Tampereen yliopisto

Jane Koivisto

ALGORITMISEN AJATTELUN KEHITTÄMINEN OHJELMOINNIN JA KIELENTÄMISEN AVULLA MATEMATIIKAN OPETUKSESSA

Informaatioteknologian ja viestinnän tiedekunta
Pro gradu
Toukokuu 2019

TIIVISTELMÄ

Jane Koivisto: Algoritmisen ajattelun kehittäminen ohjelmoinnin ja kielentämisen avulla matematiikan opetuksessa
Pro gradu
Tampereen yliopisto
Matematiikka
Toukokuu 2019

Tutkimuksessa tarkasteltiin algoritmisen ajattelun ilmenemistä ja tasoa yläkoulu- ja lukioikäisillä kielentämistehtävien ja kyselylomakkeen avulla. Opetuskokeilussa perusgeometriaa sisältäviä ohjelmointitehtäviä kielennettiin itsenäisesti työskennellen. Tutkimuksessa selvitettiin myös, miten oppilaat kokivat tehtävät, ja miten kielentäminen toi algoritmisen ajattelun esille tehtävien ratkaisuissa. Lisäksi tutkittiin, tuoko kielentäminen ohjelmointiin palautetta ja millaisilla tehtävillä algoritmista ajattelua voidaan kehittää. Algoritminen ajattelu on ajankohtainen aihe peruskoulun opetussuunnitelmauudistusten myötä, ja ohjelmointi on liitetty osaksi matematiikan oppiainetta. Algoritmisen ajattelun esiintymistä ei ole juuri tutkittu Suomessa.

Kielentämiskokeiluun osallistui viisi oppilaitosta. Oppilaita osallistui tutkimukseen yhteensä 69. Tutkimus toteutettiin osittain kehittämistutkimuksena ja osittain kyselytutkimuksena. Tutkimus kattoi kehittämistutkimuksen ensimmäisestä syklistä vain osan laajan ongelman ja sen kartoittamistarpeen vuoksi. Tutkimuksessa käytettiin pääasiassa kvantitatiivisia menetelmiä sekä tutkimustehtävien että kyselyosion analysoinnissa, mutta tuloksia täydennettiin myös kvalitatiivisin analyysein. Jatkotutkimusten tavoitteena on täydentää kehittämistutkimuksen syklit kokonaisuudessaan tuottaen kehittämismateriaaleja tulosten perusteella.

Algoritmisen ajattelun tehtävistä oppilaat saivat keskimäärin 39,2 % pisteistä ja hajonta oli suurta. Korkeampi matematiikan arvosana ja luokka-aste ennustivat parempaa menestystä tutkimuksen tehtävissä. Sukupuolella tai aiemmalla ohjelmointikokemuksella sen sijaan ei ollut merkitystä. Kokemukset ohjelmointitehtävistä ja kielentämisestä olivat vaihtelevia, mutta kielentäminen koettiin hyödylliseksi. Kyselyn perusteella oppilaat kokivat käyttäneensä tehtävissä eniten loogista ajattelua. Lisäksi kuvien piirtäminen perusteluna koettiin helpoksi.

Algoritmisen ajattelun opetusta ja arviointia tulisi kehittää, mutta tarvitaan myös konsensusta algoritmisen ajattelun määritelmästä ja sisällöistä. Opetussuunnitelmiin tulisi sisällyttää ohjelmoinnin lisäksi algoritmisen ajattelun teemoja, ja näitä tulisi sisällyttää yläkoulun lisäksi lukionkin opetussuunnitelmaan. Algoritmista ajattelua tulisi sisällyttää myös muihin oppiaineisiin kuin matematiikkaan, ja opettajia tulisi kouluttaa syvemmin algoritmisen ajattelun ymmärtämiseen ja opettamiseen. Algoritminen ajattelu on tärkeä 2000-luvun taito eri aloilla ja arkielämän ongelmanratkaisussa.

Avainsanat: algoritminen ajattelu, computational thinking, kielentäminen, laskennallinen ajattelu, matemaattinen ajattelu, ohjelmointi

SISÄLLYS

1 JOHDANTO

Tietotekniikan ja teknologian käytöstä ja kehityksestä on tullut osa kaikkien arkea, myös kouluissa. Koulutus kohtaa kehityksen mukana uusia haasteita, koska sen odotetaan vastaavan yhteiskunnan kehitykseen, hyvinvointiin ja taloudellisiin tavoitteisiin. Koulutuksen tulee tukea oppilaiden tulevaisuuden kannalta tärkeitä työelämätaitoja, edistää ympäröivän maailman tuntemista, siihen aktiivista osallistumista, sen kehittämistä sekä siinä selviytymistä erilaisin ajattelumallein, tiedoin ja taidoin. Oppilaita on ohjattava oppimaan teknisten välineiden hyötykäyttöä eri aloilla ja luomaan uusia välineitä ja menetelmiä itse.

Laskennallisten teorioiden ja laitteiden kehitys ovat mullistaneet eri tieteenaloja ja ihmisten ajattelua. Berkeley (2018, 378) arvioi, että 1940-luvulla laskenta muuttui abstraktista käsitteestä konkreettisemmaksi, koska useissa projekteissa yritettiin rakentaa mekaanisia laitteita, jotka kuuluvat nykyisin käytettyjen käsitteiden tietokone (engl. computer) ja laskenta (engl. computation) alueelle. Nykyisin tunnetun laskennan ja tietojenkäsittelytieteen kehityksen mahdollisti taustalla oleva matematiikassa luotu teoreettinen työ (Berkeley, 2018, 377; Soare, 1996a). Teoriapohjan perustamisen lisäksi matematiikkaa odotetaan sovellettavaksi sääntöjä ja menetelmiä hyödyntäen uusiin käyttötarkoituksiin sekä ilmiöiden mallintamiseen (Wake, 2016, 173). Yhteiskunta 2000-luvulla on suurelta osin tietokoneistettu ja verkotettu, ja se edellyttää kaikilta laskennallisia taitoja ja tietämystä, vaikkakin eri tasoilla (Marcelino, Pessoa, Vieira, Salvador, & Mendes, 2018, 470).

Yhteiskunnassa toimimiseksi ja teknologisen kehityksen jatkamiseksi tarvitaan osaajia. Ammattitaitoisesta työvoimasta on kuitenkin pula useilla eri tietoteknisillä aloilla (Holtgrewe, 2014; Milosz & Milosz, 2018, 16; Panko, 2008; Singh, 2012). Tässä tutkielmassa eri tietoteknisiä aloja ja käsitteitä (esimerkiksi informaatioteknologia ja informaatiotekniikka (IT), automaattinen tietojenkäsitte-

ly (ATK), tieto- ja viestintätekniikka (TVT/ICT), tietojenkäsittelytiede) ei erotella tarkoin, koska niiden sisältörajat ovat häilyviä.

Suomen yhteiskunnallista kehitystä on tarkasteltu useilla erilaisilla mittareilla. Digitaalisuuden yhteiskunnallista hyödyntämistä mittaavassa digibarometrissa 2017 Suomi sijoittui toiseksi (Liikenne- ja viestintäministeriö, Tekes, & Teknologiateollisuus ja Verkkoteollisuus, 2017, 5, 54). Euroopan 31 valtion CEPIS-tutkimuksessa Suomen vastaajien koulutus keskittyi IT-alaan kolmanneksi suurimmalla osuudella, ja osaaminen on maiden keskiarvon kanssa samaa luokkaa – kaikkialla heikkoa. Suomessa vain 22 prosentilla ICT-ammattilaisista osaamisvaatimukset vastaavat työn roolia, ja on arvioitu, että suosittuja osaamisprofiileja ei tarvittaisi tulevaisuudessa. Lisäksi ICT-ala on vahvasti sukupuolittunut, Suomessa naisten osuus on 16 % ja Euroopassa vastaava osuus on 15 %. ICT-alalle tarvittaisiin myös lisää nuoria osaajia keski-iän ollessa Suomessa ja Euroopassa yli 40 vuotta. (CEPIS (Council of European Professional Informatics Societies), 2014a, 7–8, 21–23, 32–33, 75–77, 2014b, 12.)

PISA-tutkimuksissa Suomi on menestynyt suhteellisen hyvin (esim. OECD, 2016), ja tulosten perusteella on haettu suuntaviivoja koulutuksen kehittämiseen. PISA 2015 -tulosten perusteella on huolestuttu luonnontieteiden osaamisesta ja siihen kohdistuvista asenteista ja motivaatiosta (Opetus- ja kulttuuriministeriö, 2016). Suomessa oppilaiden taustat aiheuttavat vain pieniä eroja, mutta sukupuolten väliset erot ovat suuria (Opetus- ja kulttuuriministeriö, 2017). LUMA SUOMI -kehittämisohjelmalla yritetäänkin edistää luonnontieteitä, matematiikkaa ja teknologiaa varhaiskasvatusikäisistä peruskouluikäisiin nuoriin (Opetus- ja kulttuuriministeriö, 2016). Tulevien osaajien opettaminen jo lapsuudessa ja nuoruudessa edellyttää opettajien taitojen kouluttamista. Koulujen yhteistyö korkeakoulujen ja teollisuuden kanssa olisi tärkeää opettajien taitojen lisäämiseksi (Informatics Europe & ACM Europe Working Groupon Informatics Education, 2013, 18). Kaarakaisen, Kivisen ja Vainion (2018) suomalaisten oppilaiden ja opettajien ICT-taitoja käsittelevässä tutkimuksessa miespuoliset oppilaat ja opettajat olivat naispuolisia taitavampia, ja toisen asteen koulutuksessa olevat oppilaat ja opettajat suoriutuivat peruskouluissa olevia paremmin.

Tietoteknisiin valmiuksiin on kiinnitetty opetuksessa viime vuosina enemmän huomiota. Suuntaus on näkynyt TVT:n painotuksen lisäksi ohjelmoinnin esille tuomisena, sillä opetus- ja viestintäministeri Krista Kiuru (Opetus- ja kult-

tuuriministeriön tiedote, 2014) lisäsi kaudellaan ohjelmoinnin osaksi perusopetuksen opetussuunnitelman perusteita 2014. Opetussuunnitelman (Opetushallitus, 2016) mukaan ohjelmointi toteutetaan alemmilla luokka-asteilla ohjelmointiympäristöissä ja ylemmillä luokka-asteilla ohjelmointikielillä. Koodilähettiläksikin mainittu Linda Liukas on myös tuonut vahvasti esille koodausta esimerkiksi alle kouluikäisille suuntaamassaan teoksessa Hello Ruby (Liukas, 2015). Ohjelmoinnin aloittamista sekä oppimista eri ikäisille ovat tukeneet monet eri tahot esimerkiksi sivustoin ja kerhoin (ks. esim. Mannila ym., 2014). Tietoteknisten laitteiden ja ohjelmien peruskäytön ohessa opetuksessa painotetaan myös oppilaiden luomia omia soveltavampia tietoteknisiä tuotoksia.

Ohjelmointihehkun varjoon on kuitenkin jäänyt ajattelutapa, jota tarvitaan nyky-yhteiskunnan ymmärtämiseen ja siinä toimimiseen. Seymour Papert on esittänyt käsitteen ”computational thinking” (CT) jo 1980-luvulla Mindstorms-kirjassaan (1980, 182), mutta käsite tuli tunnetuksi ja sai suurempaa yhteiskunnallista huomiota vasta Jeannette Wingin (2006) toimesta. CT:n käsitteellä on suomen kielessä useita vastineita, esimerkiksi algoritminen ajattelu tai laskennallinen ajattelu (esim. Kekäläinen, 2015). Useissa valtioissa algoritmista ajattelua on korostettu opetussuunnitelmissa jo Suomea aiemmin, ja ohjelmointi toimii yhtenä välineenä ajattelutaidon kehittämisessä (esim. Heintz, Mannila, & Färnqvist, 2016; Mannila ym., 2014). Perusopetuksen opetussuunnitelman perusteissa (Opetushallitus, 2016, 375, 379) algoritminen ajattelu on sisällytetty osaksi matematiikan oppiainetta. Lukion opetussuunnitelman perusteisiin (Opetushallitus, 2015) ohjelmointia ja algoritmista ajattelua ei ole sisällytetty (monista muista valtioista poiketen), joten perusopetuksen tiedoilla ja taidoilla on suuri merkitys esimerkiksi korkeakoulujen lähtötietojen sekä työelämän kannalta.

Digitaalisen älykkyyden osa-alueeksikin (Park, 2016) esitettyä algoritmista ajattelua on tutkittu Suomessa vasta todella vähän. Myös kansainvälisesti CT:n käsite on vasta muotoutumassa, joten ajattelutapaa tukevalle käytännön opettamiselle, algoritmisen ajattelun arvioinnille sekä kehittämiselle ollaan vasta laatimassa ohjeita ja materiaaleja (esim. S. Grover & Pea, 2013). Opettajia pitäisi kouluttaa ohjelmointiosaamisen lisäksi algoritmisen ajattelun ymmärtämiseen ja opettamiseen. Algoritminen ajattelu on lähellä monia muita ajattelumuotoja, ja erityisesti matemaattinen ajattelu sisältää sen kanssa yhteisiä osa-alueita (Sneider, Stephenson, Schafer, & Flick, 2014; Thiruvathukal, 2013). Oppilaiden ma-

temaattista ajattelua voidaan tutkia kielentämisellä (engl. languaging). Kielentämisen avulla voidaan tarkastella oppilaiden ja opiskelijoiden matemaattista ajattelua, ja saada esille oppilaiden ajatteluprosessin vahvuuksia ja heikkouksia sekä käsitteiden ymmärtämystä. (Joutsenlahti, 2003, 2009.)

Huomattavana ongelmana on, että algoritmisen ajattelun tilasta Suomessa ei ole toistaiseksi paljoa tietoa, joten algoritmisen ajattelun opetusta ei voida suoraan kehittää. Tarkoituksena on tulevaisuudessa edistää algoritmista ajattelua opetuksessa kehittämistutkimuksen avulla tuotettavan oppilaan ja opettajan materiaalien avulla. Tämän tutkimuksen kirjallisuuskatsauksessa on yritetty käsitellä algoritmisen ajattelun sisältöjä laajasti ja useista näkökulmista. Koska oppilaiden ja opettajien algoritmisen ajattelun ja ohjelmoinnin tasoa ei ole juuri Suomessa tutkittu, tämä tutkimus muodostaa kehittämistutkimukselle pohjan, jossa selvitetään oppilaiden algoritmista ajattelua ohjelmoinnin avulla. Kielentämisellä tutkitaan oppilaiden matemaattista ajattelua ja kielentämisen kaltaisella ohjelmointikoodin tulkinnalla yritetään tarkastella, miten oppilaat ymmärtävät ohjelmointikoodin logiikan.

Tutkimuksen tavoitteena on selvittää algoritmista ajattelua tutkittavilla yläkoulun oppilailla ja lukion lyhyen ja pitkän matematiikan opiskelijoilla. Jatkossa yläkoulun oppilaista ja lukion opiskelijoista käytetään yhteistä nimitystä oppilaat. Tutkimuksen osana järjestettiin geometriaa sisältäviä ohjelmointitehtäviä koskeva opetuskokeilu kirjallisen kielentämisen avulla. Tarkoituksena on selvittää algoritmisen ajattelun tilaa tutkittavilla ja tarkastella onko taustatekijöillä yhteyksiä algoritmisen ajattelun näkyvyyteen. Tutkimuksessa ollaan kiinnostuneita myös oppilaiden kokemuksista ja kielentämisen käytöstä algoritmisen ajattelun ja ohjelmoinnin yhteydessä. Lisäksi tutkimuksessa pohditaan, millaisista tehtävistä voisi olla hyötyä algoritmisen ajattelun kehittämisessä kirjallisuuden ja tämän tutkimuksen opetuskokeilun tulosten perusteella.

2 ALGORITMINEN AJATTELU

Algoritmisen ajattelun hyöty on yleisesti tunnustettu esimerkiksi matematiikan ja tietojenkäsittelytieteiden alalla. Tätä ajattelun taitoa on levitetty erityisen aktiivisesti 2000-luvulta lähtien suosiota saavuttaneen Jeannette Wingin (2006) julkaisun myötä. Wing (2006, 33) esittää algoritmisen ajattelun olevan perustavanlaatuinen taito jokaiselle, ei ainoastaan tietojenkäsittelytieteilijöille. Tämä ajattelutaito on keskeistä opettaa ottaen huomioon oppilaan oppimispotentiaali. Tutkielmassa käsitellään yleisellä tasolla algoritmisen ajattelun historiaa ja yhteyttä matematiikkaan, selvitetään tarkemmin tämän ajattelun määritelmiä sekä sisältöjä, ja tarkastellaan taidon opettamista oppilaille.

2.1 Algoritmi

Algoritmin käsite saatetaan nykyään liittää usein tietojenkäsittelytieteeseen, vaikka algoritmien käytöllä on matematiikassa pitkä historia. Soaren (1999, 4) mukaan matemaatikot ovat tutkineet laskemista ja algoritmeja jo babylonialaisten aikaan. Algoritmi on saanut nimensä persialaisen matemaatikon, al-Khwarizmin (n. 800 jaa.), mukaan (Fant, 1993, 1).[1] Al-Khwarizmi on vaikuttanut muun muassa Intiasta peräisin olevien arabialaisten lukujen leviämiseen ja käyttöön, aritmetiikkaan sekä algebran kehitykseen (Fant, 1993, 1; Folkerts, 2001; Knuth, 1997, 1; Luoma-aho, 2010, 12–35). Lisäksi al-Khwarizmin vaikutusta on arvioitu jopa nykyisen yhteiskunnan tietoteknisessä kehityksessä. Thiruvathukalin (2013, 4–5) mukaan al-Khwarizmin lukujen levityksellä on ollut merkitystä matematiikalle ja algoritmiselle ajattelulle, ja nykyistä laskennallista tehoa ei ehkä olisi käytössä ilman arabialaisten lukujen levitystä.

1 Al-Khwarizmin kansalaisuus/alue ja nimen kirjoitusasu vaihtelevat. Tässä tutkielmassa käytetään nimeä al-Khwarizmi. Mainintoina ovat esimerkiksi arabialainen (esim. Luoma-aho, 2010, 16; Soare, 1996a, 288) ja persialainen (esim. Fant, 1993, 1; Striphas, 2015, 403), ja niminä esimerkiksi al-Khowarizmi tai Al-Khowarizmi (esim. Fant, 1993, 1; Soare, 1996a, 288), al-Khwarizmi tai Al-Khwarizmi (Agarwal, Agarwal, & Sen, 2013, 11; Katz & Barton, 2007) ja al-Khwārizmī tai al-Khwārizmī (Crossley & Henry, 1990; Ghione, 2016; Striphas, 2015).

Matemaattinen työ johti vähitellen laskentaa suorittavien laitteiden kehittämiseen. Soaren (1999, 4) mukaan teoreettisten matemaattisten algoritmien kehityksen myötä kiinnostus nykyisenkaltaisiin laskennallisiin laitteisiin lisääntyi. 1600-luvulla Blaise Pascal ja Gottfried Wilhelm Leibniz keksivät laskimensa. Leibnizin työn ominaisuuksia olivat binäärijärjestelmän paremmuus suhteessa desimaalijärjestelmään mekaanisissa laitteissa sekä ongelmien symbolisen esittämisen yhdistäminen ongelmien algoritmisten ratkaisujen etsimiseen. 1800-luvulla Babbage kehitti idean koneesta, jolla voitaisiin suorittaa pitkiä ja mekaanisia laskuja. (Soare, 1999, 4.)

Matemaattiset ongelmakysymykset, niiden ratkaiseminen ja ratkaisujen lopputulokset johtivat lopulta alan osaajia keskittymään tehokkaaseen laskentaan. Fant (1993, 1–3) ja Soare (1999, 4–5) kuvailevat Leibnizin haavetta universaalisen kielen (lat. lingua characteristica) ja laskelmien perusteluiden (lat. calculus ratiocinator) löytämisestä erimielisyyksien ratkaisemiseksi. Ongelmien ratkaisemiseksi myös David Hilbert muodosti 1800-1900-luvuilla ongelmia ja kysymyksiä matemaattisen logiikan täydellisestä toimivuudesta ja ratkaisujen hyväksyttävyydestä. Matemaatikot ja loogikot, esimerkiksi Herbrand, Gödel, Church, Post, Turing, Kleene ja Markov, keskittyivät näihin Hilbertin ongelmakysymyksiin. Vaikka kysymyksiin saatiin kielteisiä vastauksia, ratkaisuprosessit johtivat tehokkaan laskettavuuden kehittämispyrkimykseen. (Fant, 1993, 2–3; Soare, 1999.) Kolaitiksen (2011, 103) mukaan laskennan ja logiikan vuorovaikutusta sekä laskentateorian kehittämistä 1930-luvulla pidetäänkin tietojenkäsittelytieteen alkutaipaleena. Viime vuosikymmeninä laskennan ja logiikan vuorovaikutus on kattanut koko tietojenkäsittelytieteen, ohjelmointikielistä tekoälyyn ja tietokantajärjestelmiin (Thiruvathukal, 2013, 103).

Eri näkökulmista laaditut laskentateorian työt saatettiin vähitellen yhteen ja algoritmin käyttö terminä alkoi yleistyä. Fant (1993, 3) arvelee, että algoritmin termiä ei käytetty yleisellä tasolla alan henkilöiden joukossa Euroopassa ja Yhdysvalloissa ennen kuin Markov yhdisti kokonaisuudeksi työt tehokkaasta laskennasta ja esitteli termin sen nykyisessä merkityksessä. Markov (1954, 1) toteaa algoritmista:

> “In mathematics, ”algorithm” is commonly understood to be an exact prescription, defining a computational process, leading from various initial data to the desired result.” (Markov, 1954, 1.)

Algoritmi tunnetaan yleisesti vaiheittaisena tai askelittaisena prosessina. Knuthin (1997, 4) mukaan algoritmia voidaan pitää melko samanlaisena kuin reseptiä, prosessia, metodia, tekniikkaa, proseduuria tai rutiinia. Wingin (2008, 3718) mukaan algoritmi on abstraktio vaiheittaisesta menettelystä, jossa otetaan syöte ja tuotetaan jokin haluttu ulostulo. Algoritmi (tietojenkäsittelyn) käsitteenä sisältää viisi tärkeää ominaisuutta (Knuth, 1997, 4–6):

1. *Finiittisyys.* Algoritmin täytyy aina päättyä äärellisen askelmäärän jälkeen.
2. *Definiittisyys*. Jokaisen algoritmin askeleen täytyy olla täsmällisesti määritetty. Suoritettavien toimenpiteiden täytyy olla täsmällisesti ja yksiselitteisesti määritelty kussakin tapauksessa. Ohjelmointikielissä jokaisella lauseella on määritelty merkitys.
3. *Syöte.* Algoritmilla on nolla tai useampia syötteitä: suureita, jotka annetaan ennen algoritmin alkua tai suureita, jotka annetaan dynaamisesti algoritmin suorituksessa. Syötteet otetaan määritellystä kohdejoukosta.
4. *Ulostulo.* Algoritmilla on yksi tai useampia ulostuloja: suureita, joilla on määritetty suhde syötteisiin.
5. *Tehokkuus.* Algoritmin oletetaan yleensä olevan myös tehokas, sen operaatioiden täytyy olla riittävän yksinkertaisia, jotta ne voidaan periaatteessa tehdä täsmällisesti ja äärellisessä ajassa paperia ja kynää käyttämällä.

Algoritmien määritelmissä on runsaasti sisällöllistä vaihtelua. Fant (1993, 4) esittää algoritmille tyypillisesti käytetyn viiden säännön määritelmän, jossa edellä kuvattuun Knuthin määritelmään verrattuna painotetaan, että algoritmin täytyy olla vaiheittainen jono operaatioita, algoritmin täytyy tuottaa tehokkaasti oikeellinen ratkaisu ja algoritmin täytyy samalla syötteellä tuottaa aina sama ratkaisu. Moschovakisin (2001, 919; ks. myös Lowrie, 2018, 351) mukaan algoritmit määritellään perusteellisesti tietojenkäsittelytieteen kirjallisuudessa vain harvoin, ja silloinkin algoritmit liitetään yleisesti abstrakteihin koneisiin, tietokoneiden matemaattisiin malleihin. Fant (1993, 4–5) kuitenkin kritisoi käsitystä, jossa oletetaan puhtaan matematiikan olevan tietojenkäsittelyn teoriaa ja antavan sopivan algoritmin määritelmän tietojenkäsittelytieteelle, koska tietojenkäsittelyssä näkökulma on erilainen kuin matematiikassa ja algoritmien ominaisuudet eivät välttämättä toteudu (esimerkiksi algoritmi voi olla ääretön ja ohjelmassa voi olla bugeja).

Tässä tutkielmassa algoritmeja käsitellään rajaamatta käsitettä. Laajemmassa näkökulmassa ajateltuna algoritmeja on kaikkialla. Matematiikan oppi-

tunneilta tutuissa algoritmeissa, esimerkiksi lukujen jakamisessa jakokulmassa ja Eukleideen algoritmissa, ratkaisut saadaan suorittamalla tiettyjä laskuvaiheita. Vaiheita suoritettaessa saattaa kehittyä myös ehtoja ja vaiheiden toistoja eli silmukoita, ja tällaisia rakenteita esiintyy myös ohjelmointikoodien rakenteissa. Lowrien (2018, 350) mukaan algoritmit automatisoivat, ja algoritminen automaatio on laajentunut puhtaalta matematiikan ja laskennan pohjalta sosiotekniseen infrastruktuuriin. Algoritmit ovat matemaattisia ja laskennallisia menetelmiä asioiden tekemiseen, kuten liikenteen reitittämiseen, syöpäkasvainten diagnosointiin MRI-kuvista, ihmisten yhdistämiseen seurustelusivustoissa, asiakkaiden analysoimiseen ja rahavirtojen muutosten seuraamiseen (Lowrie, 2018, 351). Algoritmit voidaankin arvioida merkittäväksi nykyisessä yhteiskunnassa.

Algoritmien näkökulmien lisäksi algoritmien tuottamisen taustalla olevan ajattelun mahdolliseen suppeuteen pitäisi kiinnittää huomiota opetuksessa. Thiruvathukalin (2013, 4–5) mukaan algoritmin vaiheittaisuuteen perustuva määritelmä on teknisesti korrekti, ja ohjelmointikoodi voidaan kirjoittaa joukoksi vaiheita. Algoritminen ajattelu algoritmin perustana ei kuitenkaan vaadi ilmaisemaan laskentaa vaiheina tai käyttämään ehdottomia lauseita. Algoritmin määritelmä voi muovata ajattelua algoritmeista ja haitata tehokasta opetusta tai rajoittaa mielikuvitusta ongelman ratkaisemisessa. Esimerkiksi opettajia kehotetaan päivittämään algoritmin määritelmää. (Thiruvathukal, 2013, 4–5.)

2.2 Algoritmista ajattelua kuvaavat termit

Tässä tutkielmassa käytetään Papertin (1980, 182) ja Wingin (2006, 33) käyttämästä englanninkielisestä termistä ”computational thinking” (CT) käännöstä algoritminen ajattelu. Käsitteelle CT ei ole vielä täysin vakiintunutta vastinetta suomenkielessä. Esimerkiksi MOT-sanakirja (2019) antaa sanalle ”computational” käännöksiksi 1) ”tietokone-” ja 2) ”laskennallinen”. Suomenkielisiä julkaisuja algoritmisesta ajattelusta on todella vähän.

CT-käsitteen käyttö ei ole ollut kansainvälisestikään ongelmatonta. Bocconin, Chioccariellon, Dettorin, Ferrarin ja Engelhardtin (2016, 22–23) tutkimuksen mukaan CT:hen viitataan kansainvälisesti opetuksen alalla erilaisin rinnakkaisin termein eri äidinkielissä. Käytettyjä termejä eri maissa (käännettynä englanniksi) ovat muun muassa ”algorithm”, ”algorithmic thinking”, ”coding”, ”computer

science (CS)", "computational thinking", "informatics", "informatics thinking", "language of technology", "technological literacy" ja "programming". Suomessa terminä oli käytetty algoritmista ajattelua. Syiksi termien moninaiselle käytölle on esitetty kyselyn vastausten perusteella esimerkiksi halua korostaa tiettyä sisältöä tai näkökulmaa, CT-termin pelottavuutta ja liittymistä liiaksi laskentaan sekä CT:hen sisältyvien läheisten käsitteiden parempaa ymmärrettävyyttä. Lisäksi CT voi terminä liittyä opetusta virallisempaan käyttöön (tiede, politiikka) tai CT-termi voi painottaa epäsopivaa ja merkitykseltään tarkoituksetonta sisältöä, kun se käännetään omalle äidinkielelle. (Bocconi ym., 2016, 22–23.)

Muita CT:hen viittaavia läheisiä englanninkielisiä termejä ovat esimerkiksi "computing", "digital competence/literacy", "information literacy", "information technology (IT)" ja "problem solving" (Bocconi ym., 2016, 22–23). Termien moninaisuuden taustalla voisi olla algoritmisen ajattelun käytön ja opetuksen yhteys tietoteknisiin välineisiin ja menetelmiin. Lisäksi tietoteknisiin aloihin ja käytäntöihin viittaavia termejä on useita (esimerkiksi CS, ICT, IT ja erilaiset "literacy"-termit), joita saatetaan käyttää moniselitteisellä tavalla. Tieteellisiä julkaisuja algoritmisesta ajattelusta vaikuttaisi löytyvän kansainvälisesti eniten CT:n termillä. Muita käsitteitä käyttävissä julkaisussa ei oteta välttämättä CT:n sisältöä kokonaisuudessaan huomioon.

Mahdollisia suomenkielisiä termivaihtoehtoja CT:lle on löydetty useita. Kekäläinen (2015, 27–29) pohjustaa termien sopivuutta Papertin (1980) ja myöhemmin Wingin (2006) esittämään ajattelutapaan ja ajattelutaidon merkityksiin. Lisäksi termien käytön halutaan olevan asianmukaista suhteessa Wingin (2006) kuvaamaan ymmärrykseen siitä, mitä tietoa ja ongelmia voidaan käsitellä tietokoneiden ja automaation avulla (Kekäläinen, 2015, 27–29). Olennaisinta vaikuttaisikin olevan termin ristiriidaton yhteys ajattelutaitoon ja koneiden järkevään käyttöön ongelmien ratkaisussa.

Termeissä on ollut kuitenkin sopivuuden haasteita. Kekäläisen (2015, 27–29) mukaan joillekin termeille on jo muodostunut suomenkielisiä vastineita eri käyttötarkoituksiin, termeillä on merkityksen puute tai merkitys muuttuu suhteessa CT:n sisältöön. Esimerkiksi "compute"-sanasta käännetyt laskenta-ajattelu ja laskennallinen ajattelu viittaavat liikaa laskentaan (vrt. engl. calculate), mikä ei ole alkuperäinen tarkoitus. Suomenkielinen vastine "computer"-sanalle on tietokone eikä laskentakone, ja "compute"-termi kattaakin laskentaa suuremman si-

sällön. Mallintamisajatteluun ei myöskään terminä sisälly tarpeeksi CT:n näkökulmia. (Kekäläinen, 2015, 27–29.) Laskenta saattaa vaikuttaa myös rutiininomaiselta ja tekniikasta irralliselta, mikä ei sovi algoritmiseen ajatteluun. Laskennallista ajattelua käytetään kuitenkin muun muassa Neittaanmäen, Lehdon ja Kankaanrannan (2014) raportissa sekä Laineen (2017) pro gradu -tutkielmassa.

Osa termivaihtoehdoista on rakentunut ohjelmoinnin ympärille. Kekäläinen (2015, 27–29) esittää termit ohjelmointiajattelu ja ohjelmoinnillinen ajattelu (esim. terminä käytössä: Koodiaapinen, 2019; Liukas, 2015), mutta niissäkin on ongelmia suhteessa Wingin (2006) julkaisuun, ja termit ovat jäykkiä. Lisäksi näiden termien voidaan ajatella viittaavan liiaksi ohjelmointiin, koska algoritminen ajattelu ei ole vain ohjelmointia. Kekäläinen (2015, 27–29) esittää parhaaksi vastineeksi automatisointiajattelun (esim. terminä käytössä: Koodiaapinen, 2019), koska ohjelmoidessa automatisoidaan, eikä termiä käytetä muissa yhteyksissä. Tämä on yhteydessä myös Lowrien (2018, 350) käsitykseen algoritmien käytön sitoutumisesta aina automatisointiin. Toisaalta automatisointiajattelun voidaan katsoa viittaavan tietoteknisten laitteiden käyttöön, mitä algoritminen ajattelu ei vaadi.

Jotkin termit sopisivat sisällöltään vastineeksi CT:lle, mutta ne eivät ole kielelliseltä muodoltaan tarpeeksi yksinkertaisia. Tällaisia ovat Kekäläisen mukaan algoritminen ajattelu, numeerinen ajattelu ja tietojenkäsittelyajattelu. (Kekäläinen, 2015, 27–29.) Tutkielman suomenkieliseksi CT:n vastineeksi valittiin tästä huolimatta algoritminen ajattelu, koska se viittaa suhteellisen hyvin sisällön kokonaisuuteen, algoritmit ovat käsitteenä tuttuja ja termi vaikuttaa merkitykseltään sopivalta opetusalalle. Lisäksi suomalaisen opetuksen perustana olevassa perusopetuksen opetussuunnitelman perusteissa 2014 (Opetushallitus, 2016, 374–375, 379) käytetään käsitettä algoritminen ajattelu (ks. tarkemmin luku 2.10). Algoritminen ajattelu on myös suosittu ulkomailla (Bocconi ym., 2016, 22–23). Osassa englanninkielisiä julkaisuja CT:n määritelmät ja sisällöt kattavat yhtenä osa-alueena algoritmisen ajattelun, ”algorithmic thinking” (AT). Sekaannuksen välttämiseksi tässä tutkielmassa osa-alue ”algorithmic thinking” erotetaan lyhenteellä AT kokonaisuudesta ”computational thinking” CT.

Suomenkielisen termin vakiinnuttamisen lisäksi vastineita tarvittaisiin myös CT:hen viittaaville muille käsitteille, kuten ”computation”, ”computational”, ”com-

puting” ja ”models of computation”. Näissäkin kohdataan ongelma suoran käännöksen ”laskenta” kanssa, joten tulevat vastineet olisi hyvä soveltaa paremmin algoritmiseen ajatteluun ja tietojenkäsittelyyn liittyviksi.

2.3 Algoritmisen ajattelun määritelmiä

Oppilaat tarvitsevat algoritmisen ajattelun taitoja teknologisessa maailmassa. Yadav, Hong ja Stephenson (2016, 565) esittävät algoritmisen ajattelun olevan kaikkien oppilaiden keskeinen 2000-luvun taito (engl. 21st century skill). Algoritminen ajattelu on muuttanut nykypäivän tapoja toimia. Ihmismieltä pidetään tehokkaimpana ongelmanratkaisutyökaluna, mutta tietokoneet auttavat ongelmien ratkaisemisessa arjessa ja työssä. On tärkeää ymmärtää tilanteet, joissa tietokoneet ja muut digitaaliset työkalut ovat hyödyllisiä ongelmien ratkaisussa. (D. Barr, Harrison, & Conery, 2011, 23.)

Wing (2006, 33) ilmaisee, että algoritminen ajattelu pitäisi lisätä oppilaiden analyyttisiin kykyihin lukemisen, kirjoittamisen ja laskuopin lisäksi. Denningin (2017b, 34–35) mukaan algoritmisella ajattelulla on vahva historia jo 1940-luvun tietojenkäsittelystä alkaen. George Polyan matematiikan ongelmanratkaisun menetelmiä ja teosta ”How to Solve It” (1971 alkup. 1945) pidetään algoritmisen ajattelun esiasteena. Muita algoritmiseen ajatteluun viitanneita vaikuttajia ovat muun muassa Alan Perlis, Donald Knuth, Edsger Dijkstra, Seymour Papert, Ken Wilson ja Alfred Aho. (Denning, 2017b, 34–35.)

Algoritmiselle ajattelulle on esitetty runsaasti erilaisia määritelmiä (myös tieteellisten julkaisujen ulkopuolella). Wing (2006, 33) kuvailee julkaisussaan laajasti algoritmista ajattelua. Hänen mukaansa algoritminen ajattelu rakentuu laskennallisten prosessien tehon ja rajoitusten varaan. Prosessien suorittamisen voi toteuttaa kone tai ihminen. Algoritminen ajattelu sisältää ongelmien ratkaisua, järjestelmien suunnittelua ja ihmisen käyttäytymisen ymmärtämistä hyödyntämällä tietojenkäsittelytieteen peruskäsitteitä. (Wing, 2006, 33.) Varsinainen algoritmisen ajattelun määritelmä jää kuitenkin laveaksi. Cuny, Snyder ja Wing (2011a, 20) määrittelevät käsitteen tarkemmin seuraavasti:

> “Computational thinking is the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent.” (Cuny, Snyder, Wing, 2010. Viitattu lähteestä Wing, 2011a, 20.)

Ongelmanratkaisu ja abstraktio esiintyvät usein määritelmissä. Ahon (2011, 7) mukaan matemaattiset abstraktiot, joita kutsutaan laskennallisiksi malleiksi (engl. models of computation), ovat keskeisiä laskennassa ja algoritmisessa ajattelussa. Laskenta on prosessi, joka on määritelty perustana olevan laskennallisen mallin suhteen. Algoritminen ajattelu määritellään ajatteluprosesseiksi, jotka liittyvät ongelmien muotoiluun niin, että ongelmien ratkaisut voidaan esittää laskennallisina vaiheina ja algoritmeina. (Aho, 2011, 2, 7.) Osana algoritmisen ajattelun ongelmanratkaisuprosessia on pidetty järjestelmien suunnittelua, abstraktiotasojen luomista, kykyä soveltaa matemaattisia käsitteitä kehittämään tehokkaita, oikeudenmukaisia ja turvallisia ratkaisuja sekä taloudellisten ja sosiaalisten syiden ymmärtämistä (The Center for Computational Thinking at Carnegie Mellon, ei pvm.). Myös García-Peñalvo (2017, 408) määrittelee algoritmisen ajattelun olevan korkean abstraktiotason ja algoritmisen lähestymistavan soveltamista kaikenlaisten ongelmien ratkaisemiseksi.

Osa määritelmistä korostaa ongelmien ratkaisun lisäksi esimerkiksi havainnointia, päättelykykyä, algoritmisen ajattelun käyttöä eri aloilla tai erilaisia ajatteluun liittyviä ulottuvuuksia. The Royal Societyn (2012, 29) mukaan algoritminen ajattelu voidaan kuvata prosessiksi, jossa tunnistetaan laskennan näkökohdat ympäröivässä maailmassa, ja sovelletaan tietojenkäsittelytieteen välineitä ja tekniikoita sekä luonnollisten että keinotekoisten järjestelmien ja prosessien ymmärtämiseksi ja järkeilemiseksi. Samaan tapaan Riley ja Hunt (2014, 4) esittävät, että parhaiten algoritmista ajattelua voidaan luonnehtia tavalla, jolla tietojenkäsittelytieteilijät ajattelevat ja järkeilevät. Algoritminen ajattelu on termi, joka käsittää joukon tietojenkäsittelytieteen käsitteitä ja ajatteluprosesseja, jotka auttavat ongelmien ja ratkaisujen muotoilemisessa eri aloilla (Mannila ym., 2014, 2).

The International Society for Technology in Education (ISTE) ja the Computer Science Teachers Association (CSTA) ovat muodostaneet algoritmiselle ajattelulle myös operatiivisen määritelmän K-12-opetukseen (Suomessa K-12 vastaa yleensä oppilaiden ikää noin 17-vuotiaisiin asti). ISTE:n ja CSTA:n mukaan algoritminen ajattelu on ongelmanratkaisuprosessi, joka sisältää seuraavat ominaisuudet (mutta ei ole rajoitettu niihin):

- Ongelmien muotoilu tavalla, joka mahdollistaa tietokoneen ja muiden työkalujen käytön apuna ongelmien ratkaisemisessa.

- Datan (tietojen) järjesteleminen ja analysoiminen loogisesti.
- Datan kuvaaminen abstraktioilla, kuten malleilla ja simulaatioilla.
- Ratkaisujen automatisointi algoritmisella ajattelulla AT, (sarja järjestettyjä vaiheita).
- Mahdollisten ratkaisujen tunnistaminen, analysoiminen ja toteuttaminen tavoitteena saavuttaa suorituskykyisin sekä tehokkain vaiheiden ja resurssien yhdistelmä.
- Ongelmanratkaisuprosessin yleistäminen ja siirtäminen monenlaisiin ongelmiin. (ISTE & CSTA, 2011; myös D. Barr ym., 2011, 20–21; vrt. Sysło & Kwiatkowska, 2013, 50.)

Edellä on esitetty pieni osa algoritmisen ajattelun valtavasta määritelmäjoukosta. Yhteisen määritelmän muodostaminen on vaikeaa. Kalelioğlun, Gülbaharin ja Kukulin (2016, 589) tutkimuksessa tarkasteltiin 125 CT-käsitettä koskevaa julkaisua. Tutkituissa julkaisuissa algoritmisen ajattelun määritelmää kuvattiin eniten sanoilla ongelmanratkaisu (22 %), abstraktio (13 %) ja tietokone (13 %). Muita yleisiä sanoja olivat prosessi, tiede, data, tehokas, algoritmi, käsitteet, kyky, työväline ja analysointi. (Kalelioğlu ym., 2016, 589.)

Erilaisille yksittäisille määritelmille ja määritelmien moninaisuudelle on annettu kritiikkiä. Cooperin, Pérezin ja Raineyn (2010, 27) mukaan Wingin (2006) määritelmä sisältää paljon insinööri- ja ohjelmistotuotantosanastoa. Lisäksi tiedemiesten näkemykset jakaantuvat – jotkut pitävät algoritmista ajattelua tietojenkäsittelytieteen taitojen joukkona, jotkut taas näkevät sen uutena kuvauksena perustieteestä, joka edustaa tietojenkäsittelytiedettä ja sen leikkausta muiden alojen kanssa. On ihmetelty, miten CT-käsite eroaa esimerkiksi matemaattisesta ajattelusta, algoritmisesta ajattelusta (AT), ongelmanratkaisusta, sekä monista muista ajatteluista ja matematiikan, tieteen ja tekniikan malleista. (Cooper ym., 2010, 27–28.)

Määritelmien laajuus tuottaa ongelmia myös opetukseen. Denningin (2017b, 33–34) mukaan useista määritelmistä huolimatta opettajilla on peruskysymyksiä algoritmisesta ajattelusta. Opettajien on vaikea olla tehokkaita, jos he eivät tiedä mitä opettavat ja miten arvioivat (Denning, 2017a, 15, 2017b, 34). Denning esittää määritelmien olevan sekavia ja liioittelevia. Toisaalta hänestä on kapeakatseista väittää CT:tä yksinkertaisesti ongelmanratkaisumenetelmäksi, koska se tukee laskennallisen suunnittelun päämäärää. CT:n omaksuminen tapahtuu laatimalla opetustarjontaa, joka auttaa oppimaan tehokkaammaksi omalla alalla laskennan avulla. (Denning, 2017b, 34–39.)

Algoritminen ajattelu ja sen määritelmät on otettu vähitellen huomioon eri koulutustasojen opetussuunnitelmissa. Cooperin ym. (2010, 27) mukaan K-12-opetuksessa ja STEM-aineissa (science, technology, engineering ja mathematics) määritelmät ja kuvaukset ovat hyödyttömiä ja aiheuttavat ongelmia, vaikka niiden kanssa voidaan toimia korkeakoulutasolla. CT:n määritelmät ovat hämmentäviä, kun niitä esitetään muille kuin tietojenkäsittelytieteiden ammattilaisille. (Cooper ym., 2010, 27.) Myös Grover ja Pea (2013, 42) esittävät, että on keskitytty määritelmiin ja algoritmisen ajattelun kehitystä edistäviin työkaluihin. Muutoksia on tehty valtakunnallisiin opetussuunnitelmiin, mutta silti tarvitaan paljon empiiristä tutkimusta täyttämään laajat tieteelliset aukot. (S. Grover & Pea, 2013, 42.)

Joitakin Yhdysvalloissa havaittuja opetuksen huolia (Cooper ym., 2010, 27) voidaan havaita myös Suomessa. Esimerkiksi lukiossa tietojenkäsittelytieteitä ei painoteta samalla tavalla kuin peruskoulussa, ja peruskoulussakaan tietojenkäsittelytieteille ei ole omaa oppiainetta. Valinnaiset tietojenkäsittelyyn liittyvät kurssit ovat koulu- ja aluekohtaisia. Lisäksi samalla kun jotain uutta halutaan opettaa (esimerkiksi ohjelmointia matematiikassa), jotakin muuta olisi otettava opetuksesta pois tai painotuksia muutettava. Opettajille jää suuri vastuu aihealueiden tärkeyden arvioimisessa. Opettajilla ei välttämättä ole tarpeeksi koulutusta ja tietotaitoa algoritmisen ajattelun opettamiseen. (vrt. Cooper ym., 2010, 27.)

Yhteenvetona voidaan todeta, että algoritmiselle ajattelulle ei ole vielä muodostunut vakiintunutta määritelmää. Algoritminen ajattelu on kuitenkin ajatteluprosessi, jossa olennaisena osana on mahdollisuus tietoteknisen laskentatehon sekä tietojenkäsittelijöiden käsitteiden ja toimintamallien hyödyntämiseen esimerkiksi uusien menetelmien, palveluiden ja tuotteiden suunnittelussa, luomisessa, soveltamisessa sekä ongelmanratkaisussa eri aloilla. Kaikki tarvitsevat jollakin tasolla algoritmista ajattelua 2000-luvulla. Yhteisymmärryksessä laaditun selkeän ja yksikäsitteisen määritelmän puuttuessa algoritmisen ajattelun sisällöt ja erityisesti opetustavat sekä arviointimenetelmät ovat olleet taka-alalla, mutta niihin on alettu kiinnittää aiempaa enemmän huomiota. Näkemyseroja on myös algoritmiseen ajatteluun kuuluvista sisällöistä ja yhteyksistä muihin ajattelutapoihin.

2.4 Algoritmisen ajattelun sisältöjä

Yksittäiset tutkijat ja eri tahot ovat esittäneet algoritmiseen ajatteluun kuuluvia keskeisiä taitoja ja sisältöjä yleisesti sekä opetuksen näkökulmasta. Vaihtelu on kuitenkin suurta, ja sisältöjä on myös ryhmitelty monin eri tavoin. Gander ym. (2013, 12–13) ryhmittelevät sisällöt erilaisiin ongelmanratkaisutekniikoihin ja intellektuaalisiin käytäntöihin. Denning (2010, 369–372) puolestaan ryhmittelee tietojenkäsittelyn rungon seitsemään kategoriaan: laskentaan, kommunikaatioon, koordinaatioon, muistamiseen, automaatioon, suunnitteluun ja arviointiin, jotka pitkälti vastaavat algoritmisen ajattelun sisältöjä ja taipumuksia. CT-käytäntöihin Deden, Mishran ja Voogtin (2013, 2–3) mukaan kuuluvat tietojenkäsittelyn liittäminen, laskennallisten tuotteiden kehittäminen, abstrahointi, ongelmien ja tuotteiden analysointi, kommunikaatio ja yhteistyö.

Kalelioğlun ym. (2016, 590) tutkimuksessa algoritmisen ajattelun yleisimmiksi tunnusmerkeiksi havaittiin abstraktio (20 %), algoritminen ajattelu AT (14 %) ja ongelmanratkaisu (13 %). Muut 11 havaittua näkökulmaa olivat mallin tunnistaminen, suunnitteluperustainen ajattelu, käsitteellistäminen, hajottaminen, automaatio, analyysi, testaaminen sekä virheiden etsiminen ja korjaaminen, yleistäminen, matemaattinen päättely, ratkaisujen toteuttaminen ja mallintaminen (Kalelioğlu ym., 2016, 590). Liitteessä 1 on listattu algoritmisen ajattelun sisältöjä, joita seuraavaksi käsiteltävät eri tutkijat ovat esittäneet. Wing (2006) on listannut useita CT:ssä tarvittavia ja havaittavia taitoja. Suuri osa sisällöistä liittyy vahvasti tietojenkäsittelytieteisiin ja alan termistöön, joten sisällöt ovat melko hankalia sovellettavaksi suoraan korkeakoulutasoa matalampitasoiseen opetukseen.

Barr ja Stephenson (2011, 50–54) kuvaavat National Science Foundationin (NSF), ISTE:n ja CSTA:n vuonna 2009 alkanutta projektia, jossa yritettiin selventää CT:n käsitettä. Projektissa selvitettiin, miten oppilaat oppivat algoritmista ajattelua kaikilla koulutustasoilla ja kaikissa oppiaineissa. Pitkän aikavälin tavoitteena on suositella sellaisia tapoja, että kaikilla oppilailla on mahdollisuus oppia taidot, ja varmistaa, että taidot voidaan siirtää eri ongelmiin ja kontekstei-
hin. (D. Barr ym., 2011, 23.) Barr ja Stephenson viittaavat CT:n K-12-opetuksen toimintamääritelmän (ks. luku 2.3), ja esittävät algoritmiseen ajatteluun liittyvät ydinkäsitteet ja kyvyt. Algoritminen ajattelu on lähestymistapa ongelmien ratkai-

semiseen tavalla, joka voidaan toteuttaa tietokoneella, ja sitä voidaan automatisoida, siirtää ja soveltaa eri aineissa. Oppilaat eivät pelkästään käytä työkaluja vaan myös rakentavat niitä. Lisäksi oppilaat käyttävät erilaisia käsitteitä, kuten abstraktiota, rekursiota ja iterointia, käsittelemään ja analysoimaan tietoja (dataa) sekä luomaan todellisia ja virtuaalisia esineitä. (V. Barr & Stephenson, 2011, 50–52.)

Mannila ym. (2014) jakavat Barrin ja Stephensonin (2011) esittämät algoritmisen ajattelun sisällöt kolmeen teemaan: 1) datan kerääminen, analysointi ja esittäminen; 2) ongelman hajottaminen, algoritmit, abstraktio ja rinnakkaisuus; 3) automaatio ja simulaatio. Toisen teeman sisällöt keskittyvät informaation prosessointiin, ja taitoja käytetään ongelmaan lähestymisessä ja sen ratkaisemisessa. Taidoilla voidaan hahmottaa kielellisesti ratkaisukeinoja, organisoida työskentelyä ja tutkia omaa ajatusprosessia, sekä hallita ongelman monimutkaisuutta. Kolmannen teeman automaatiossa tietojenkäsittelyyn liittyvät toistuvat tehtävät suoritetaan koneella ajan säästämiseksi ja virheiden välttämiseksi. Simulaatioita luodaan tuottamaan uutta tietoa. (Mannila ym., 2014, 13–15.) Lee ym. (2011, 32–33) esittävät puolestaan algoritmisen ajattelun sisällöiksi abstraktion, automaation ja analyysin. Jokaista osa-aluetta käsitellään käytännön suhteen. Käytännön osa-alueina ovat mallintaminen ja simulaatio, robotiikka sekä pelisuunnittelu. (Lee ym., 2011, 33.)

Brennan ja Resnick (2012, 3–11) ovat tehneet määritelmän algoritmiselle ajattelulle Scratchiä tutkiessaan. Määritelmä koostuu kolmesta ulottuvuudesta, jotka ovat laskennalliset käsitteet (käsitteet, joita käytetään ohjelmoitaessa), laskennalliset käytännöt (käytännöt ohjelmoidessa) ja laskennalliset näkökulmat (näkökulmat, joita muodostetaan ympäristöstä ja itsestä). Jokainen näistä ulottuvuuksista on jaettu tarkempiin ohjelmointia koskeviin sisältöihin. (Brennan & Resnick, 2012.) Selby ja Woollard (myös Selby, 2014; 2013) ovat sen sijaan yrittäneet kaventaa algoritmisen ajattelun määritelmää sisältöjen karsimisella. CT on heidän mukaansa lähestymistapa ongelmanratkaisuun, ja hyväksyttäviä sisältöjä kirjallisuuskriteerien perusteella ovat kognitiivinen prosessi tai ajatteluprosessi, abstraktio, hajottaminen, algoritminen suunnittelu tai algoritminen ajattelu (AT), arviointi ja yleistys. CT:n sisältötermejä on hylätty esimerkiksi huonon määrittelyn ja laajuuden vuoksi, jos kyseessä on algoritmisen ajattelun seurauksena syntynyt taito, jos termi ei ole erityisen ainutlaatuinen algoritmiselle ajatte-

lulle, tai jos termi on paremminkin kehittävä työkalu kuin määritelmään sopiva sisältö. (Selby, 2014; Selby & Woollard, 2013.)

Grover ja Pea (2013, 39–40) ovat esittäneet yleisesti hyväksytyt algoritmisen ajattelun elementit, jotka tukevat algoritmisen ajattelun oppimista ja arvioivat sen kehitystä. Toisaalta näistä yleisesti hyväksytyistä osa-alueista poiketaan usein. Esimerkiksi Englannin opetussuunnitelmaan pohjautuvassa opettajille laaditussa Csizmadian ym. (2015) tekstissä algoritmisen ajattelun piiriin kuuluvat looginen päättely, abstraktio, arviointi, algoritminen ajattelu (AT), hajottaminen ja yleistäminen. Myös Angeli ym. (2016, 50) esittävät algoritmisen ajattelun sisällöiksi melko samanlaiset ominaisuudet. Lisäksi Weintrop ym. (2016, 133) käsittelevät algoritmisen ajattelun taitojoukkoa sekä erilaisia asennoitumista kuvaavia ominaisuuksia ongelman käsittelyyn ja ratkaisuun liittyen melko yksityiskohtaisesti.

Edellä kuvatun perusteella algoritmisen ajattelun sisältöjen määrä on laaja, varsinkin, jos niiden joukkoon liitetään muitakin yleisiä taitoja, jotka voivat olla laveita ja vaikeasti määriteltäviä. Lisäksi algoritmisen ajattelun osaajilla on monia algoritmisesta ajattelusta syntyviä kykyjä, jotka usein liitetään jälleen osaksi algoritmisen ajattelun taitoa. Näitäkin ominaisuuksia voidaan pitää algoritmiseen ajatteluun kuuluvina sisältöinä, koska niiden oppiminen voi kuvastaa algoritmisen ajattelun kehittymistä ja hallintaa. Toisaalta on tärkeää muistaa, että kaikki käsitteet eivät sovi algoritmisen ajattelun määritelmään. (vrt. Selby & Woollard, 2013.)

Näihin asioihin on viitannut myös Weinberg (2013, 63–64), jonka mukaan algoritmisen ajattelun tutkimiselle on useita haasteita. Yhteinen määritelmä ja teoreettinen malli puuttuvat, mitkä tarvittaisiin algoritmisen ajattelun käsitteen tukemiseksi. Lisäksi algoritminen ajattelu on piilevä muuttuja. Sitä ei voida suoraan havaita, vaan se on pääteltävä muista muuttujista, joita havaitaan tai mitataan suoraan. Haasteena on myös tehtyjen tutkimusten vähyys, ja toisaalta kvalitatiiviset tutkimukset, joita ei voida yleistää. Tarvittaisiin määrittelyä ja tutkimusta siihen, miten algoritmista ajattelua voidaan arvioida ja mitata taitoja yleisesti. Lisäksi pitäisi tutkia tuloksia, jotka voisivat olla yleistettävissä erilaisiin tilanteisiin ja ohjelmiin. (Weinberg, 2013, 63–64.) Käsitellään seuraavaksi ongelmanratkaisun näkökulmaa ja muutamia yleisiä algoritmisen ajattelun sisältöjä tarkemmin.

2.4.1 Näkökulmana ongelmanratkaisu

Algoritmisessa ajattelussa olennaista on ongelmien ratkaiseminen. CT:llä on havaittu olevan yhteyksiä ongelmanratkaisuun, esimerkiksi Polyan ongelmanratkaisumenetelmiin (Denning, 2017b, 34–35) ja Bloomin taksonomiaan (Selby, 2012, 2014). Wingin (2006, 33–35) mukaan algoritminen ajattelu on vaikealta vaikuttavan ongelman uudelleenmuotoilua helpommaksi ongelmaksi, joka osataan jo ratkaista. CT:ssä ei tarvita aina tietokonelaitetta, sillä CT on ihmisten tapa ajatella ongelmien ratkaisemista, mutta tietokonelaitteita voidaan käyttää apuna. Tietokonelaitteiden avulla voidaan ratkaista ennen tietojenkäsittelyn aikakautta mahdottomilta vaikuttaneita ongelmia. (Wing, 2006, 33–35.)

CT kohtaa koneälyn keskeiset kysymykset: mitä ihminen voi tehdä konetta paremmin ja päinvastoin, sekä mikä on laskettavissa olevaa. Lisäksi pohditaan, miten tietokone saadaan ratkaisemaan ongelma. Huomioon pitää ottaa laskentalaitteen resurssit, kuten teho, rajat, aika, tila, tallennuskapasiteetti, ohjelman oikeellisuus ja selkeys. (Wing, 2006, 33–35.) Oppilaiden ymmärtäessä tietokoneen kyvyt ja rajat, he ymmärtävät paremmin tieteellisiä kysymyksiä ja teknisiä projekteja. Lisäksi CT:n taidot voivat joissain tapauksissa rohkaista harjoittamaan tiedettä, teknologiaa ja matematiikkaa jatkossa. (Sneider ym., 2014, 13.) Wing (2006, 33–35) kuitenkin korostaa, että CT ei ole pelkkää ohjelmointia ja tuotteiden luomista, vaan käsitteellistämistä ja ideointia ongelmia lähestyttäessä ja ratkaistaessa. Pelkkä ohjelmointitaito ei riitä, vaan tarvitaan paljon syvällisempi ajattelun taito. Ongelman ratkaisemisessa voidaan pohtia, kuinka vaikeaa ongelma on ratkaista ja mikä olisi paras tapa ratkaista se. Lisäksi pitää arvioida, onko tehty ratkaisu ongelmaan tarpeeksi hyvä. (Wing, 2006, 33–35, 2008, 3719.)

Algoritmisessa ajattelussa keskitytään ajatteluprosessiin ja soveltamiseen. Csizmadian ym. (2015, 5–6) mukaan CT:n avulla oppilaat voivat ymmärtää maailmaa sekä yleisesti että digitaalisesti syvemmällä tavalla. Algoritmisessa ajattelussa tunnistetaan tietojenkäsittely ympäröivässä maailmassa, ja sovelletaan tietojenkäsittelyn työkaluja ja tekniikoita ymmärtämään ja perustelemaan luonnollisia, sosiaalisia ja keinotekoisia järjestelmiä ja prosesseja. Painopiste keskittyy ajatteluprosessia suorittaviin oppilaisiin esineiden tuottamisen sijaan. Toisaalta CT on aktiivinen ongelmanratkaisumenetelmä, jossa oppilaat käyttä-

vät erilaisia käsitteitä prosessoimaan ja analysoimaan dataa sekä luomaan todellisia ja virtuaalisia tuotoksia, eli oppilaat eivät vain käytä työkaluja vaan tulevat työkalujen rakentajiksi (García-Peñalvo, Reimann, Tuul, Rees, & Jormanainen, 2016, 19). Algoritminen ajattelu on ajattelutaitojen kehittämistä, ja se tukee oppimista ja ymmärtämistä. (Csizmadia ym., 2015, 5–6.)

Viimeisen 20 vuoden aikana lähes kaikki tieteen ja matematiikan (STEM) osa-alueet ovat sisältäneet laskennallisuuden kasvua. Näillä onkin vastavuoroinen suhde: laskenta rikastuttaa matematiikan ja luonnontieteiden oppimista ja toisaalta taas matematiikan ja tieteen kontekstin käyttö lisää algoritmista oppimista. (Weintrop ym., 2016, 128–129.)

2.4.2 Datan kerääminen, analysointi ja esittäminen

Mannilan ym. (2014, 13) mukaan datan kerääminen (engl. data collection) on prosessi, jossa kerätään tarkoituksenmukainen informaatio. Datan analysoinnissa (engl. data analysis) datasta otetaan selvää ja yritetään ymmärtää se, etsitään malleja sekä tehdään johtopäätöksiä. Datan esittämisellä (engl. data representation) data järjestellään ja kuvataan esimerkiksi sopivilla kuvaajilla, kaavioilla, taulukoilla, sanoilla tai kuvilla. (CSTA & ISTE, 2011; Mannila ym., 2014, 13; Sysło & Kwiatkowska, 2013, 51.) Datan keräämisessä, analysoinnissa ja esittämisessä kuvataan informaation ja symbolisen datan yhteyksiä ja eroja. Näissä osa-alueissa tärkeää on lähdekriittisyys, ja miten ja millä välineillä aineistosta luodaan esitettävää informaatiota (Mannila ym., 2014, 13–15.)

Datan kerääminen, analysointi ja esittäminen voidaan nähdä tulkinnanvaraisiksi algoritmisen ajattelun sisällöiksi. Mannila ym. (2014, 13) ovat pohtineet, onko datan kerääminen CT:n toimintaa vai pitäisikö se tulkita enemmänkin digitaalisen lukutaidon (engl. digital literacy) kehittämiseksi. Suomen opetussuunnitelmien perusteissa (ks. lisää Opetushallitus, 2016 ja luku 2.9) viitataan myös tiedon keräämiseen ja analysoimiseen. Tiedon etsiminen internetistä ei riitä CT:n toiminnaksi, mikäli siihen ei liity datan analyysia ja/tai esittämistä. (Mannila ym., 2014, 5, 11–13.) Toisaalta dataan liittyvät sisällöt kuuluvat automaattisesti algoritmiseen ajatteluun ongelmia lähestyttäessä ja ratkaistaessa. Dataa tuotetaan lisää jatkuvasti (Wing, 2008, 3720). Tämän vuoksi on olennaista etsiä sopi-

va data ongelmaan, muokata data sopivilla laskennallisilla menetelmillä ja välineillä hyödylliseksi, sekä esittää laskennan tuloksena saatu ratkaisu mahdollisimman järkevällä tavalla.

Mannilan ym. (2014, 11–13) tutkimuksessa opettajat käyttivät kolmea dataan liittyvää sisältöä oppilaidensa kanssa huomattavasti enemmän kuin muita CT:n sisältöjä. Lisäksi opetuksessa käytettiin paljon enemmän internetiä, sosiaalista mediaa ja toimistotyökaluja kuin esimerkiksi ohjelmointiin, simulaatioihin ja robotiikkaan liittyviä työkaluja (Mannila ym., 2014, 11–13). Cooper ym. (2010, 28) kuitenkin esittävät, että esimerkiksi CT:lle läheiseen algoritmiseen oppimiseen (engl. computational learning) eivät sisälly välineet, kuten diaesitykset, wikit ja blogit.

2.4.3 Looginen päättely

Loogista päättelyä tai ajattelua (engl. logical reasoning/thinking) ei sisällytetä aina CT:n määritelmiin, koska se on laaja ja tulkinnanvarainen käsite (Selby & Woollard, 2013, 3,5). Looginen päättely mahdollistaa sen, että oppilaat ottavat selvää asioista analysoimalla ja tarkastelemalla faktoja. Tämä vaatii selkeää ja tarkkaa ajattelua. Oppilaat käyttävät tietojaan ja sisäisiä mallejaan, joiden avulla luodaan ja todennetaan oletuksia sekä päädytään ratkaisuun. Loogista päättelyä voidaan käyttää laaja-alaisesti muun muassa testaamisessa, virheiden etsimisessä ja algoritmien korjaamisessa esimerkiksi ohjelmointikoodista. Loogisen päättelyn harjoittaminen on yhteydessä moniin CT:n taitoihin. (Csizmadia ym., 2015, 6.)

Loogisella ajattelulla ja matematiikan oppimisella on havaittu olevan yhteyksiä lapsuudessa. Suuri osa lasten matemaattisesta tietämyksestä perustuu ymmärrykseen sen taustalla olevasta logiikasta. (Nunes ym., 2007.) Loogisen päättelyn taso kasvaa johdonmukaisella käytöllä erilaisissa konteksteissa, ja kun loogista päättelyä opitaan varhaisina vuosina. Loogisen päättelyn ja sukupuolen yhteyksiä on tutkittu useissa tutkimuksissa matematiikassa. Looginen päättelykyky ja sukupuoli ovat eri tutkimuksissa antaneet usein toisistaan poikkeavia tuloksia. (Zaman, Farooq, Hussain, Ghaffar, & Satti, 2017, 46.)

2.4.4 Abstraktio

Abstraktiota (engl. abstraction) pidetään algoritmisen ajattelun tärkeimpänä prosessina (Wing, 2011a, 20), ja abstraktio erottaa algoritmisen ajattelun muista ajattelutavoista (S. Grover & Pea, 2013, 39). Tietojenkäsittelytieteissä työskennellään usein samanaikaisesti kahden tai useamman abstraktiokerroksen kanssa (Wing, 2008, 3718). Abstraktiota käytetään mallien määrittelemisessä, erityisten tapausten yleistämisessä ja parametrisoinnissa. Abstraktiolla voidaan muuttaa mittakaavaa ja käsitellä monimutkaisuutta. (Wing, 2011b, 20.) Abstraktion taitoon voidaan liittää myös abstraktioiden välisten suhteiden tunnistaminen ja tietojen suodattaminen ratkaisuja kehitettäessä (Csizmadia ym., 2015, 15).

Abstraktiolla on useita merkityksiä CT:ssä, mutta usein sillä tarkoitetaan yleistämisprosessia. Wingin (2008, 3718; myös Csizmadia ym., 2015, 7) mukaan abstraktioprosessissa päätetään, mitä yksityiskohtia korostetaan ja mitä jätetään huomiotta. Abstraktio voidaan myös määritellä tärkeiden yhteisten ominaisuuksien tai toimintojen kokoamiseksi tai tallentamiseksi yhteen joukkoon, jota voidaan käyttää kaikkien muiden tapausten esittämiseen, ja piilottaa joukossa olevat epäolennaiset erot. (Lee ym., 2011, 32–33; Wing, 2011a, 20.) Abstraktion avulla oppilaat tekevät ongelmista tai järjestelmistä helpommin ajateltavia. Taitoon kuuluu tarpeettomien yksityiskohtien valinta piilotettavaksi niin, että ongelma muuttuu monimutkaisesta helpommaksi, menettämättä mitään tärkeää. (Csizmadia ym., 2015, 7, 15.) Toisaalta abstraktion yleistystä pidetään monimutkaisuuden vähentämisenä pääidean määrittämiseksi (CSTA & ISTE, 2011; Sysło & Kwiatkowska, 2013, 15) tai yksinkertaistamisena konkreettisesta yleiseen, kun ratkaisuja kehitetään (V. Barr & Stephenson, 2011, 52).

Yleistämisen lisäksi abstraktiossa voidaan nähdä piirteitä myös ongelman hajottamisesta, ratkaisun siirtämisestä ja esittämisestä. Ongelmanratkaisussa abstraktio voi tarkoittaa ongelman purkamista kaikkein olennaisimmiksi pidettäviin osiin (Lee ym., 2011, 32–33). Abstraktioon kuuluu kyky yleistää ja siirtää yhden ongelman ratkaisu muihin vastaavankaltaisiin ongelmiin. Lisäksi abstraktioon sisältyvät mallien kehittäminen ja esittäminen todellisesta maailmasta (esimerkiksi mallit ja simulaatiot). (Yadav ym., 2016, 567.) Keskeinen osa abstrak-

tiota onkin valita hyvä ja hyödyllinen järjestelmän esitys (Csizmadia ym., 2015, 7, 15).

Abstraktio voidaan esimerkiksi jakaa ohjelmoinnissa kahteen osa-alueeseen, jotka ovat dataan ja toimintaan liittyvät osat. Toiminnallisten abstraktioiden aktivointi johtaa johonkin käyttäytymiseen. Abstraktiolla voidaan piilottaa kohteen täysi monimutkaisuus (vähennetään kohteen toiminnallista monimutkaisuutta) tai datan monimutkaisuus (luodaan ja käsitellään esimerkiksi tietorakenteita, kuten listaa tai taulukkoa). (Csizmadia ym., 2015, 15.) Abstraktiot koetaan yleensä haastaviksi (dataa käsittelevät abstraktiot toiminnallisia haastavammiksi), ja tähän vaikuttaa aikaisemman kokemuksen puute. (Selby, 2014, 192–193, 208.)

2.4.5 Arviointi

Prosessin, algoritmin tai järjestelmän arvioinnilla (engl. evaluation) varmistetaan, että ratkaisu on hyvä ja sopii tarkoitukseensa. Lisäksi arvioinnilla arvioidaan ratkaisun ominaisuuksia. Nämä koskevat esimerkiksi oikeellisuutta, nopeutta, resursseja, käyttöä ja kokemusta. Valintoja on tehtävä, koska harvoin on olemassa yhtä ihanteellista ratkaisua kaikkiin tilanteisiin. Esimerkiksi ohjelmoinnin tapauksessa ohjelman ja sen sisältämien algoritmien tehokkuutta, turvallisuutta, vaikuttavuutta, taloudellisuutta ja käytettävyyttä on arvioitava. Prosessin, algoritmin tai koodin arviointi voidaan jakaa opetuksessa osiin, esimerkiksi toiminnalliseen oikeellisuuteen, testaamiseen, suorituskykyyn, vastaavanlaisten kohteiden vertailuun, käytettävyyden osatekijöihin, heuristiikkojen toimivuuteen ja ohjelman eri osien toimintojen vaiheittaiseen selvittämiseen. (Csizmadia ym., 2015, 7, 15.)

Arviointi on tuttua, ja sitä voidaan harjoittaa eri oppiaineissa, esimerkiksi kielten esseiden tai kemian käytännön kokeiden arvioinnissa. Algoritmisen ajattelun sisällöistä arviointi koetaan yhdeksi helpoimmista taidoista hallita. Arviointi ei silti välttämättä ole johdonmukaista ja syvällistä, ratkaisuihin voi jäädä huomaamattomia virheitä. Arvioinnin tulisikin olla kriittistä ratkaisun heikkouksien suhteen, ja lisäksi pitäisi pohtia, onko ratkaisu parempi tai huonompi kuin jokin toinen vaihtoehto. (Selby, 2014, 173–174, 189–190.) Wingin (2006, 33) mukaan CT:ssä arvioidaan ohjelmaa oikeellisuuden ja tehokkuuden lisäksi myös estetii-

kasta, ja järjestelmän suunnittelua yksinkertaisuudesta ja tyylikkyydestä. CT vaatiikin selkeän ymmärryksen algoritmisen prosessin suunnittelusta ja luomisesta, jotta arviointi olisi mahdollista ja se hallittaisiin (Selby, 2014, 199).

Selby (2014, 173–174) jakaa termit analysointi (engl. analysis) ja arviointi niin, että analysointi käsittelee ongelmaa ja arviointi ratkaisua. Lisäksi arviointi sijoittuu korkeammalle tasolle kuin analysointi (Selby, 2014, 173–174). Analysointiin kuuluu piirteitä hajottamisesta, abstraktiosta, algoritmeista ja yleistyksestä. Analysointia käytetään alustavaan ongelman pohdintaan, ja se sisältää loogisen ajattelun käyttämistä asioiden parempaan ymmärtämiseen sekä tarkoituksenmukaisuuden arvioimiseen. (Csizmadia ym., 2015, 9.) Leen (2011, 32–33) mukaan analysointia voidaan kuitenkin pitää myös reflektoivana käytäntönä, joka viittaa tehtyjen abstraktioiden oikeellisuuden validointiin. Analysointiin sisältyy seuraavan tyyppisiä reflektiivisiä kysymyksiä: "Tehtiinkö oikeita oletuksia, kun ongelma kavennettiin olennaisimpiin osiin?", "Jätettiinkö tärkeitä tekijöitä ulkopuolelle?" tai "Oliko abstraktion tai automaation toteutus virheellinen?" (Lee ym., 2011, 32–33).

2.4.6 Algoritminen ajattelu (AT)

Algoritmista ajattelua voidaan pitää CT-käsitteen suomenkielisen vastineen lisäksi myös yhtenä CT-käsitteen osa-alueista tai CT:tä edeltävänä muotona. Denningin (2009, 28) ja Weinbergin (2013, 53) mukaan englanninkielinen CT-käsite on tunnettu tietojenkäsittelytieteessä 1950- ja 1960-luvuilla käsitteenä ”algorithmic thinking”, ja se on tarkoittanut mentaalista suuntautumista tai prosessia ongelmien muotoilemiseksi (syötteen muuntamista tulosteeksi) ja algoritmien etsimistä (muuntamisen suorittamista) käyttäen vaiheittaisia menetelmiä. Nykyinen CT-käsite on siis alkuperäistä käsitettä laajempi. Hemmendingerin (2010, 6) mukaan laskentateoria on tuonut algoritmisen ajattelun monimutkaisuuteen täsmällisiä käsitteitä, ja tämä on tullut osaksi matematiikan algoritmien tutkimusta vasta äskettäin.

CT-käsitteen osa-alueena algoritminen ajattelu (engl. algorithmic thinking, AT) tarkoittaa tapaa päätyä ongelman ratkaisuun määriteltyjen vaiheiden, algoritmien, avulla (Csizmadia ym., 2015, 7, 14; CSTA & ISTE, 2011; Sysło & Kwiatkowska, 2013, 51). Ajattelutapaa hyödyntäen yhteen ongelmaan kehitetään rat-

kaisualgoritmi, ja samaa algoritmia voidaan edelleen kehittää ja soveltaa. Tarvitaan ratkaisu, joka toimii ongelmaan joka kerta. Tällöin samankaltaisia ongelmia pystytään ratkaisemaan jo käytetyin algoritmein ja metodein. Kun algoritmi on kerran ymmärretty, sitä ei tarvitse kehittää tyhjästä jokaiseen uuteen ongelmaan. AT on kykyä ajatella sarjojen ja sääntöjen suhteen. Se on ydintaito, jota oppilaat kehittävät oppiessaan kirjoittamaan omia tietokoneohjelmia. (Csizmadia ym., 2015, 7, 14.)

Ohjelmien kirjoittamisen näkökulmasta AT:hen kuuluu sääntöjen muodostaminen aritmeettisia ja loogisia operaatioita käyttäen. Lisäksi varastoidaan ja manipuloidaan dataa esimerkiksi muuttujien avulla. Algoritmeissa käytetään silmukoita ja iteraatiota toistotehtävien suorittamiseen. Tähän liittyy myös nimeäminen ja funktioiden ja metodien tekeminen (aliohjelmat). Luotaessa algoritmeja kuvataan todellista maailmaa mallintamalla. (Csizmadia ym., 2015, 14.) Yadavin ym. (2016, 567) mukaan muita CT:n osa-alueita saadaan toteutettua tavallisessa luokkahuoneessa, mutta automaation opetus mallintamisen ja simulaatioiden avulla vaatii tietojenkäsittelytieteen välineitä.

Selbyn (2014, 213) mukaan AT:ta voidaan kehittää, ja tämä taito voidaan osoittaa ohjelmoinnin avulla. Ohjelmointiprosessista syntyvän ohjelmointituotteen lisäksi osoitusta AT:sta voidaan saada muillakin logiikan muodostuksen keinoilla, esimerkiksi vuokaavion tai pseudokoodin avulla. On raportoitu, että oppilaille logiikan muodostaminen on todella haastavaa. Prosessi sisäisestä logiikasta vuokaavion ja pseudokoodin kautta valmiiksi koodiksi vähentää kognitiivista kuormaa eri vaiheissa. Edellä kuvattua prosessia vaativien ongelmien systemaattinen esittäminen askeleittain voi auttaa kehittämään AT:n taitoa. (Selby, 2014, 213.)

2.4.7 Hajottaminen ja yleistäminen

Hajotusta (engl. decomposition) hyödyntämällä laaja ongelma pilkotaan helpommin käsiteltäviksi osaongelmiksi, jotka voidaan ratkaista erikseen. Osaratkaisut yhdistämällä saadaan koko ongelman ratkaisu. Hajottamisella on helpompi ratkaista ja ymmärtää monimutkaiset ongelmat sekä suunnitella suurempia kokonaisuuksia. (V. Barr & Stephenson, 2011, 52; Csizmadia ym., 2015, 8; CSTA & ISTE, 2011, 14; Sysło & Kwiatkowska, 2013, 51.)

Hajottamista harjoitellaan eri ikätasoilla ja monissa oppiaineissa, erityisesti matematiikassa. Ongelmien hajottaminen on melko haastavaa oppilaille. Tämä johtuu kokemuksen puutteesta, epätäydellisestä ratkaistavan ongelman ymmärtämisestä ja ohjelmointitaitojen opetusjärjestyksestä ohjelmointia opeteltaessa. Oppilaat pystyvät käyttämään hajottamistaitoa paremmin tilanteissa, joissa he tietävät ratkaisun etukäteen tai ymmärtävät ongelman hyvin. Ohjelmoinnin opetusjärjestyksessä haasteita tuottaa, jos ohjelmoinnin perusasioiden opettelun jälkeen siirrytään suoraan ongelmiin, joissa tarvitaan hajottamista osiin. Ongelman hajottamisen taito on edellytyksenä muille algoritmisen ajattelun osa-alueille, kuten abstraktiolle, algoritmien suunnittelulle ja arvioinnille. (Selby, 2014, 195–197.)

Yleistäminen (engl. genaralisation) liittyy kaavojen ja mallien tunnistamiseen, yhteyksien löytämiseen ja ratkaisujen laajentamiseen (Csizmadia ym., 2015, 8, 14; Selby, 2014, 213). Tunnistettuja ominaisuuksia, aiempia kokemuksia ja opittuja asioita hyödyntämällä voi ratkaista ongelman helpommin, kun ongelmasta tunnistaa yhtäläisyydet aikaisempiin ratkaisuihin. Ominaisuuksien tunnistamiseen liittyvät pohdinnat: ”Olenko ratkaissut vastaavan ongelman aikaisemmin?” ja ”Millä tavoin ongelma on erilainen kuin aiemmat ratkaisemani?”. Tähän liittyvät olennaisesti datan käsittely ja ratkaisustrategiat. (Csizmadia ym., 2015, 8, 14.)

Algoritmeja, jotka ratkaisevat joitakin kapea-alaisia ongelmia, voidaan soveltaa ratkaisemaan koko samanlaisten ongelmien luokka. Kun havaitaan uusi ongelma kyseisestä luokasta, voidaan soveltaa yleistä ratkaisua. Yleistämiseen liittyy myös ideoiden ja ratkaisujen siirtäminen tilanteesta toiseen. (Csizmadia ym., 2015, 8, 14.) Yleistämistä voidaan pitää yleisenä ongelmanratkaisutaitona, ja sitä voidaan kehittää harjoittelemalla. Yleistämisen taitoja voidaan kehittää vaikeuttamalla tasaisesti ongelmia, joissa on tunnistettavia piirteitä. (Selby, 2014, 213.)

Selbyn (2014, 191–192) tutkimuksessa yleistämistä ei pidetty laajassa näkökulmassa erityisen vaikeana taitona hallita. Vastauksista kuitenkin ilmeni, että oppilaat tarvitsevat neuvoja yleistämisen strategioihin ongelmia ratkaistessaan. Yleistämistä pidetään strategisena taitona, mutta sen soveltamistilanteita ei aina heti tunnisteta. Yleistämisen tunnettuudesta huolimatta termiä harvoin käytetään täsmällisesti. (Selby, 2014, 191–192.)

2.5 Algoritmisen ajattelun harjoittaminen

Erilaisilla käytännöillä, tekniikoilla ja taipumuksilla voidaan auttaa CT:n harjoittamista. Osa tutkijoista myös erottaa algoritmisen ajattelun ja siihen liittyvän toiminnan toisistaan. Denningin (2009, 30) mukaan CT:n sijaan pitäisi keskittää huomio algoritmiseen toimintaan (engl. computational doing), joka keskittyy eri alojen ongelmien ratkaisemismahdollisuuksiin. Csizmadian ym. (2015, 9) mukaan algoritmiseen toimintaan kuuluvat reflektio, koodaus, suunnittelu, analysointi ja soveltaminen.

Algoritmisen ajattelun taitoja tuetaan ja edistetään asenteilla ja taipumuksilla. Näihin asenteisiin ja taipumuksiin kuuluvat muun muassa luottamus monimutkaisuuden käsittelemiseen, kyky käsitellä monimerkityksisyyttä, sinnikkyys työskennellessä vaikeiden ongelmien parissa, sietokyky monimerkityksisyyden suhteen ja kyky käsitellä avoimia ongelmia. Lisäksi muiden kanssa työskenneltäessä tärkeitä ovat kommunikointikyky, erimielisyyksien kumoaminen, yksilöiden vahvuuksien ja heikkouksien tunteminen ja työskentely yhdessä muiden kanssa yhteisen tavoitteen tai ratkaisun saavuttamiseksi. (V. Barr & Stephenson, 2011, 51; ISTE & CSTA, 2011; vrt. intellektuaaliset käytännöt Gander ym., 2013, 13.) Neuvottelussa tiimin sisäiset ryhmät työskentelevät yhdessä yhdistääkseen ratkaisun osia kokonaisuudeksi. Yhteisymmärryksen rakentamisessa työskennellään ryhmän solidaarisuuden rakentamiseksi idean tai ratkaisun tueksi. (V. Barr & Stephenson, 2011, 52.)

Algoritmista ajattelua ja hyödyllistä oppimiskokemusta edistävään luokkahuonekulttuuriin sisältyvät seuraavat asiat: Opettajien ja oppilaiden laskennallisen sanaston lisääntynyt käyttö, jos se on tarkoituksenmukaista ongelmien ja ratkaisujen kuvaamiseksi; opettajien ja oppilaiden epäonnistuneiden ratkaisuyritysten hyväksyminen ymmärtäen, että varhainen epäonnistuminen voi usein johtaa onnistumiseen; opiskelijoiden tiimityö, jossa käytetään nimenomaisesti hajottamista, abstraktiota, neuvottelua ja yhteisymmärryksen rakentamista. (V. Barr & Stephenson, 2011, 52.)

Lähestymistavoiksi on myös kuvattu kokeellisuus, pelaaminen, luomistyö sekä virheiden etsiminen ja korjaaminen (debuggaus) (Csizmadia ym., 2015, 11). Algoritmisen ajattelun opetuksessa voidaan käyttää monipuolisesti erilaisia välineitä ja opetusmenetelmiä. Mannilan ym. (2014) tutkimuksen perusteella

opettajat käyttävät algoritmisen ajattelun sisältöjen opettamiseen verkkomateriaalia ja sosiaalista mediaa (alle 85 % vastaajista), toimistotyökaluja ja -ohjelmistoja (noin 80 % vastaajista), graafisia ohjelmointiympäristöjä, kuten Scratch (alle 35 % vastaajista), ohjelmointikieliä, kuten Python, Java (noin 20 % vastaajista), simulaatioita, kuten NetLogo (alle 10 % vastaajista) ja robotiikkaa, kuten Lego Mindstorms, Bee-Bot (alle 10 % vastaajista). Opettajat käyttävät myös muita välineitä. (Mannila ym., 2014, 13.) On huomioitava, että tutkimus oli suunnattu K-9 opettajille, joten mukana tutkimuksessa on myös alakoulu- ja päiväkoti-ikäisten opettajia, mutta ei lukio-opettajia.

Mannila ym. (2014) antavat esimerkkejä erilaisista opetukseen sopivista CT:tä hyödyntävistä tehtävistä ja aktiviteeteista. Niissä käytetään esimerkiksi ilman elektroniikkaa tehtäviä aktiviteetteja, kinesteettisiä aktiviteetteja, oppiaineiden rajat ylittäviä projekteja, ohjelmointia, tarinankerrontaa, opetusrobotteja, interaktiivisten korttien ja julisteiden luomista, kokeilemista ja simulointia, opetuspelien pelaamista, yleistyksien testaamista tekstistä ja tehokasta muistiinpanojen tekemistä. (Mannila ym., 2014, 15–22.) Bers, Flannery, Kazakoff ja Sullivan (2014, 145) esittävät, että algoritmista ajattelua kehittäviksi laitteiksi kelpaavat tietokoneet, älypuhelimet sekä tabletit, ja sitä voidaan harjoittaa myös ilman tietokonetta. Lisäksi on olemassa erilaisia sähköisiä oppimisleluja. Yhä nuorempien lasten käyttäessä näitä välineitä haasteeksi nousee sopivasti kehittävien toimintojen ja sisältöjen valinta opetuksessa. (Bers ym., 2014, 145.)

Repenning, Webb ja Ioannidou (2010, 266) tarkastelevat algoritmista ajattelua pelisuunnittelun näkökulmasta. Heidän mukaansa algoritmisen ajattelun työkalujen pitäisi toteuttaa opetussuunnitelmien puitteissa tietyt ehdot, joita ovat 1) alhainen kynnys (toimivan pelin voi tuottaa nopeasti), 2) korkea katto (oikea pelattava peli, mahdollisuus monimutkaisuuteen), 3) rakentamisen virtaus (asteittain monimutkaistuvat pelisuunnittelun haasteet ja taidot), 4) siirtämisen mahdollistaminen (toimivuus pelisuunnittelussa ja myöhemmissä laskennallisissa tiedesovelluksissa ja tuki näiden välisissä siirroissa), 5) oikeudenmukaisuuden tukeminen (saatavuus ja motivoivuus sukupuolesta ja etnisestä taustasta riippumatta), 6) yhdenmukaisuus ja kestävyys (kaikki opettajat voivat opettaa kaikkia oppilaita, sopivuus kaikille ja eri tarkoituksiin, esim. opettajankoulutukseen). (Repenning ym., 2010, 266–268.) Monet ohjelmointityökalut ja -ympäris-

töt täyttävät edellä mainitut kriteerit, esimerkiksi Scratch, Alice, Game Maker, Kodu ja Greenfoot (S. Grover & Pea, 2013, 40).

Basawapatna, Koh, Repenning, Webb ja Marshall (2010, 225; 2011, 246–250) ovat käyttäneet pelisuunnittelun, laskentatieteen ja robotiikan osa-alueita sisältäviä algoritmisen ajattelun malleja (engl. computational thinking patterns, CTP) pelien yhteydessä ja havainneet, että ymmärrys ja oivallukset ovat siirrettävissä myös peliohjelmoinnin kontekstin ulkopuolelle.

Esimerkiksi pelisuunnittelu ja robotiikka lisäävät motivaatiota ja innostavat tietojenkäsittelyyn ja CT:n tutkimiseen. Visuaalisista ja aistillisista ohjelmointikokemuksista siirrytään korkeatasoisempiin ohjelmointikieliin (Python, Java). Koodaaminen voi olla vetovoimaista nuorille opiskelijoille ja hyvä käytäntö tai kokemus (García-Peñalvo & Mendes, 2017, 407). Kuitenkin mielenkiintoisempaa voisi olla kehittää opiskelijoiden loogista ajattelukykyjä ja ongelmanratkaisutaitoja ohjelmointilähestymistavoilla tai algoritmisella ajattelulla (García-Peñalvo & Mendes, 2017, 407). Lisäksi voidaan käyttää e-tekstiileitä. On myös kehitetty puhelimelle visuaalisia ohjelmointiympäristöjä (Massachusetts Institute of Technology, 2018). Tärkeää olisi luoda ja käyttää välineitä sukupuolineutraalisti. Algoritmisessa ajattelussa on myös alihyödynnetty erilaisia lupaavia alustoja, oppimisympäristöjä ja videopelejä. (S. Grover & Pea, 2013, 40–41.)

Oppilaat voivat jakaa osaongelmia tai tehtäviä keskenään, jotta he pystyisivät työskentelemään rinnakkain (engl. parallellization). Rinnakkain työskentelemisessä organisoidaan tehtäviä niin, että voidaan suorittaa tehtäviä samanaikaisesti yhteistä tavoitetta varten (CSTA & ISTE, 2011, 9; Sysło & Kwiatkowska, 2013, 51). Esimerkiksi Scratch-ohjelmointiympäristössä on mahdollista ajaa erilaiset ohjelmakoodit rinnakkain. (Mannila ym., 2014, 13–14.)

Lee ym. (2011, 32–35) esittävät algoritmisen ajattelun sisällyttämistä nuorille esimerkiksi mallinnukseen ja simulaatioon, robotiikkaan ja pelisuunnitteluun ja pelinkehitykseen. Lisäksi verkkosivujen suunnittelussa ja ohjelmoinnissa, älypuhelinsovelluksissa ja monissa muissa voi olla potentiaalia nuorten algoritmisen ajattelun kehittämiseen. Esimerkeissä on keskeistä algoritmisen ajattelun käsitteiden aktiivinen käyttö. Algoritminen ajattelu voidaan helposti upottaa toimintoihin, jotka kannustavat nuoria luovuuteen, innovaatioihin ja ongelmanratkaisuihin. Projekteissa lopputuloksena on oppilaiden luoma tuote. (Lee ym., 2011, 35.)

Lisäksi Lee ym. (2011, 35) ehdottavat kolmivaiheista etenemistä nuorten osallistumiselle algoritmiseen ajatteluun laskennallisissa ympäristöissä. Oppimisen eteneminen noudattaa ”Use-Modify-Create”-vaiheita. Käyttövaiheessa oppilaat käyttävät jonkun toisen luomusta. Muokkausvaiheessa oppilaat alkavat vähitellen muokata valmista luomusta yhä kehittyneemmällä tasolla. Luomisvaiheessa oppilaalla on tarve muokata luomusta niin, että tarvitaan esimerkiksi uuden koodiosan kehittämistä. Muutos edellyttää ymmärrystä ainakin mallissa, simulaatiossa ohjelmassa tai pelissä olevan abstraktion ja automaation osajoukosta. Useiden muutosten ja iteratiivisten parannusten avulla kehitetään uusia taitoja ja ymmärrystä, ja samalla muokatusta luomuksesta tulee oppilaan oma. Luomisvaiheeseen sisältyvät kaikki algoritmisen ajattelun keskeiset osa-alueet: abstraktio, automaatio ja analyysi. (Lee ym., 2011, 35.)

Visualisointi on tärkeä osa laskennassa, ja sen avulla voi ymmärtää paremmin tieteellisiä kysymyksiä ja laskennallisia periaatteita ja prosesseja. Visualisointia ei useinkaan käytetä tehokkaasti laskennallisten käsitteiden opetuksessa. VPython voisi olla hyvä visualisoinnissa, koska sen oppii nopeasti tekemällä. (Hambrusch, Hoffmann, Korb, Haugan, & Hosking, 2009, 184, 187.) Aiken ym. ovat käyttäneet ohjelmointiympäristönä VPythonia mallinnusopetuksen fysiikan kurssilla. VPython perustuu Pythoniin, ja VPythonilla voidaan luoda esimerkiksi 3D-animaatioita (vpython.org, ei pvm.). Opetuksen jälkeen noin kolmasosa oppilaista kykeni suorittamaan onnistuneesti tehtävän, jossa rakennettiin malli uudelle fysiikan systeemille. (Aiken ym., 2013, 46, 49.)

Ohjelmointi ei riitä algoritmisen ajattelun opettamiseen, mikäli ohjelmointia pidetään pelkkänä "koodauksena", rutiininomaisena tehtävänä, jossa annetaan ohjeita tietokoneelle perinteisten ongelmien ratkaisemiseksi. Koodaus on murto-osa ohjelmointiprosessista. Se on analysoinnin, hajottamisen ja suunnittelun vaiheiden jälkeinen tehtävä, joka suoritetaan ratkaisun saamiseksi. Kaikki vaiheet ovat tärkeitä algoritmisessa ajattelussa. Jos ohjelmointi nähdään laajana ja monipuolisena prosessina, niin sitä voidaan pitää työkaluna kaikkien algoritmisen ajattelun näkökohtien kehittämisessä. Keskusteluissa koodaus olisikin vaihdettava ohjelmointiin, ja muutettava käsitykset ohjelmoinnista ongelmanratkaisutoiminnaksi, jota voidaan käyttää käsittelemään algoritmisen ajattelun eri osa-alueita. (Mannila ym., 2014, 4.)

Crow (2014) perustelee ohjelmoinnin opettamista sillä, että ohjelmistot hallitsevat elämää nykyisin ja tulevaisuudessa maailma on yhä enemmän verkossa ja digitaalisempi. Tulevaisuudessa tietokoneiden kielen oppimattomuus on yhtä haastava tilanne kuin lukutaidottomuus tai laskutaidottomuus nyt. Vaikka jokaisessa työssä ei ohjelmoida, olisi tärkeää saada kaikki lapset oppimaan ohjelmointia algoritmisen ajattelun edistämiseksi, koska algoritminen ajattelu opettaa uudenlaisen tavan ajatella maailmaa. (Crow, 2014.) Koodaus kuuluu valikoimaan työkaluja, joita voidaan käyttää algoritmisen ajattelun opettamiseen (García-Peñalvo ym., 2016, 3). Ohjelmointia voidaan oppia esimerkiksi yhdistämällä elektroniikkaa, ohjelmointia, robotiikkaa, tekstiilikäsityössä ja teknisessä käsityössä tai puunjalostuksessa. (Tuomi, Multisilta, Saarikoski, & Suominen, 2018, 421.)

Basogainin ym. (2016) tutkimuksessa tarkasteltiin peruskoulun ja lukion algoritmisen ajattelun kuuden kuukauden kestoista kurssia. Käytössä olivat Scratch 2.0 -ohjelmointiympäristö ja Moodle-oppimisalusta. Kurssi sisälsi algoritmisen ajattelun ja ilmaisun, abstraktion (monimutkaisuuden yksinkertaistus ja ongelmien hajotus), multimediasisällön integroinnin (teksti, kuvat, ääni, ym.), ohjelmien ja toiminnallisten lohkojen kehittämisen, interaktiivisten ohjelmien (tapahtumat) ja ohjelmoinnin peruskäsitteiden (muuttujat, silmukat, funktiot, rinnakkaisuus, ym.) elementit. Tulokset kurssilta arvioitiin positiivisiksi. Lisäksi Basogain ym. esittävät, että algoritmiseen ajatteluun koulutetut opiskelijat ovat paremmin valmistautuneita arkisiin tehtäviin ja ammattimaiseen työhön lähitulevaisuudessa. (Basogain ym., 2016, 1–4.)

Hambruschin ym. (2009) tutkimuksessa arvioitiin CT:n johdantokurssia yliopistossa, jossa opiskelijoista 27 % ei ollut ohjelmointikokemusta, 40 % oli kokeillut itsenäisesti ohjelmointia ja 33 % oli osallistunut lukion ohjelmointikurssille. Kurssilla käytettiin Pythonia opettamaan CT:tä perusohjelmointikäsitteiden, tiedonhallintakäsitteiden, simuloinnin ja visualisoinnin kautta. Arvioinnin mukaan ongelmanratkaisuperustainen tieteellisiin löytöihin ja laskentaperiaatteisiin keskittyvä lähestymistapa, kyky kirjoittaa nopeasti merkityksellisiä ohjelmia ja visualisointityökalujen käyttö ovat ratkaisevia opiskelijoiden kiinnostuksen lisäämisessä. (Hambrusch ym., 2009.)

Perinteistä ohjelmointia on pidetty liian teknisenä ja liian tylsiä opettamaan ja oppimaan oppilaita koulussa. Nykyään ohjelmoinnissa voidaan käyttää mo-

nenlaisia ympäristöjä (esim. lautapelit, visuaaliset ja lohkoperustaiset ohjelmointityökalut) ja laitteita (esim. robotit ja paperikortit), jotka ovat hyödyllisiä algoritmisen ajattelun oppimisessa. Painotus ohjelmointiin tai koodaukseen on vahvaa myös kansainvälisesti. Monesti keskitytään enimmäkseen ohjelmointiin yleisen algoritmisen ajattelun sijaan. (Mannila ym., 2014, 4.)

Ohjelmoinnin ei pitäisi olla välttämätöntä algoritmisen ajattelun opettamisessa. Ohjelmointi tulisi aloittaa kaikille opiskelijoille vasta sen jälkeen, kun heillä on ollut huomattavaa harjoittelua ajattelua laskennallisesti. Ohjelmointi on tietojenkäsittelytieteille samankaltainen prosessi kuin todistaminen matematiikassa. (Lu & Fletcher, 2009, 260–261.) Ohjelmointi on tietotekniikassa keskeinen toiminta, mutta vain työkalu (Tucker ym., 2006, 2).

Burtonin (2010) mukaan algoritmiselle ajattelulle on järjestetty myös kynäpaperikilpailuja, joissa mahdollistetaan nopea vastauksiin vastaaminen (monivalinnat ja lyhyet vastaukset), vastausten tarkistaminen (oikein tai väärin), eritasoiset tehtävät ja helppo lähestyttävyys ilman aiempaa ohjelmointitietoa (kysymykset eivät turvaudu ohjelmointi- tai laskentatietämykseen, eivätkä ne sisällä koodia tai pseudokoodia). On haastavaa tuottaa tehtäviä, jotka keskittyvät algoritmeihin ja algoritmiseen ajatteluun, mutta eivät liity ennalta oletettuun tietoon (esim. ohjelmointikieliin), ja eivät ole pelkkiä perinteisiä matematiikan ongelmia ja pulmia. Monivalinnoissa voi olla 1) algoritmisia tehtäviä, joissa kannustetaan oppilaita kehittämään algoritmia ongelman ratkaisemiseksi; 2) logiikkatehtäviä, joissa edistetään täsmällistä päättelyä ja tapausanalyysiä; 3) jäljitystehtäviä, joissa noudatetaan määriteltyjä ohjeita; 4) tutkimustehtäviä, joissa otetaan selvää algoritmin tai ongelman vahvuuksista ja heikkouksista. Kolmivaiheisissa tehtävissä ratkaistaan koko ajan samaa ongelmaa, mutta datajoukko kasvaa jatkuvasti. Näissä houkutellaan oppilaita algoritmien laatimiseen ja siirtymään arvailutekniikoista järjestelmällisiin menetelmiin. (Burton, 2010, 4-6,11.)

Simulaatio sisältää prosessien mallien kuvauksen ja niiden kokeellisen ajamisen (CSTA & ISTE, 2011, 9; Sysło & Kwiatkowska, 2013, 51). Luonnontiedepainotteisissa aineissa ja teknologiassa voidaan hyödyntää esimerkiksi simulaatioita, tiedonlouhintaa (engl. data mining) ja automaattista tiedonkeruuta ja analysointia. Simulaatioissa tulee olla dynaamisia tietokonemalleja, joissa oppilaat voivat muuttaa olosuhteita ja tehdä muutoksista havaintoja. Algoritmista ajattelua voidaan kehittää vielä lisää kysymällä, miten simulaatio auttaa visuali-

soimaan tilanteen ja vastaako simulaatio oppilaiden mielestä todellista tilannetta. Lisäksi oppilaat voivat käyttää algoritmista ajattelua pohtiessaan, miten simulaatio toimii, ja rakentaa simulaatio-ohjelmistoilla omia simulaatioita. (Sneider ym., 2014, 11–12.)

Automaatio on prosessi, jossa tietokone on ohjeistettu suorittamaan toistuvien tehtävien joukko nopeasti ja tehokkaasti. Automaatiossa ihmistyövoimaa säästetään turhalta tai tehottomalta työltä. (CSTA & ISTE, 2011, 9; Lee ym., 2011, 32–33; Sysło & Kwiatkowska, 2013, 51.) Automaattisessa tiedonkeruussa ja analysoinnissa voidaan hyödyntää tietokoneeseen tai älypuhelimeen kytkettyä laitetta tai anturia, joista kerätään ja analysoidaan tieto sopivan ohjelmiston tai sovelluksen avulla. Tällöin algoritmista ajattelua tarvitaan laitteiden hyötykäytössä ja tiedon tuottamisessa välineiden avulla sekä tiedon muotoilussa tarkoituksenmukaiseen muotoon. Samalla oppilaat ymmärtävät tutkittavaa ilmiötä. (Sneider ym., 2014, 13.)

Weintropin (2018) mukaan virtuaalitodellisuudella (virtual reality, VR) on potentiaalia tehokkaaksi välineeksi tuomaan CT luokkahuoneisiin, myös STEM-aineiden ulkopuolelle. Virtuaalitodellisuus osoittaa laskennallisen ajattelun laajuutta ja näyttää, kuinka CT:tä voidaan kokea tietokoneluokkien ulkopuolella. VR:n yhdistämisellä algoritmiseen ajatteluun oppilailla olisi myös mahdollisuus kehittää algoritmista ajattelua luonnontieteiden lisäksi myös humanistisilla aloilla. VR:ssä on myös haasteita ja rajoitteita käyttöönottoon luokkatilassa. Korkeahkot kustannukset rajaavat käyttöä ja opettajilla ei ole osaamista suunnitella omia VR-sovelluksiaan. (Weintrop, 2018, 87–88.)

Epämuodollisina CT:n sisältämiin taitoihin ja käsitteisiin johdattelevat erilaiset kerhot ja kilpailut. Kerhoja ovat muun muassa CoderDojo (CoderDojo Foundation, ei pvm.) ja Code Club (Raspberry Pi Foundation, ei pvm.) sekä eri maiden ja yritysten kerhot. Lisäksi on olemassa kilpailuja kyvykkäille oppilaille, kuten International Olympiads in Informatics (International Olympiad in Informatics, ei pvm.). Projekteista esimerkiksi *The Value of Computational Thinking across Grade Levels* (VCTAL) (National Science Foundation, ei pvm.) kannustaa lukioikäisiä algoritmiseen ajatteluun (Weinberg, 2013, 57). CT:tä käsittelevät myös muut ohjelmat, kuten CS Unplugged (Bell, Alexander, Freeman, & Grimley, 2009; Bell, Witten, & Fellows, 2015; Computer Science Education Research Group, ei pvm.) ja CS4FN (Curzon, McOwan, Cutts, & Bell, 2009; Queen Mary

University of London, ei pvm.), Hour of Code (Hour of Code, ei pvm.) ja European CodeWeek (Europe Code Week, ei pvm.).

2.6 Algoritmisen ajattelun yhteydet muihin käsitteisiin

Algoritmisella ajattelulla on yhteyksiä useisiin ajattelutapoihin. Weinbergin (2013, 53) mukaan CT liittyy esimerkiksi proseduraaliseen ajatteluun. Proseduraalinen lukutaito tarkoittaa tiettyjen laskennallisten työkalujen rajoitusten ja mahdollisuuksien ymmärtämistä. Rekursiivisella ajattelulla ratkaistaan mittavia ongelmia hajottamalla ne pienemmiksi ongelmiksi, joilla on identtiset muodot. (Weinberg, 2013, 53.) Deden ym. (2013, 1) mukaan algoritmisella ajattelulla on yhteyksiä 2000-luvun kykyihin, kuten ongelmanratkaisuun, kriittiseen ajatteluun, tuottavuuteen ja luovuuteen.

Barrin, Harrisonin ja Coneryn (2011, 22–23) mukaan monet algoritmisen ajattelun käsitteistä, taidoista ja taipumuksista eivät ole uusia. Algoritminen ajattelu kuitenkin eroaa kriittisestä ajattelusta ja matemaattisesta ajattelusta seuraavin perustein (tosin tästä ei ole laajaa yhteisymmärrystä): CT on ainutlaatuinen yhdistelmä ajattelutaitoja, jotka yhdessä käytettynä muodostavat perustan uudelle ja tehokkaalle ongelmanratkaisumuodolle. CT on työkalupainotteisempi, ja siinä hyödynnetään ongelmanratkaisutaitoja, kuten yritystä ja erehdystä, iterointia ja arvaamista tilanteissa, joissa ne olivat aiemmin epäkäytännöllisiä. Nyt ne ovat mahdollisia, koska ne voidaan automatisoida ja toteuttaa paljon suuremmilla nopeuksilla. (D. Barr ym., 2011, 22–23.)

Hun (2011, 225) mukaan algoritminen ajattelu ja matemaattista ajattelu ovat lähellä toisiaan, vaikka tietojenkäsittelyssä on erilainen näkökulma. Matemaattiseen ajatteluun kuuluvat algoritmista ajattelua vastaavat taidot, kuten analyyttisyys, hajottaminen, samankaltaisuuksien ja yhteyksien tunnistaminen sekä mallien ja algoritmien käyttäminen. Lisäksi yhteisiä taitoja ovat esimerkiksi looginen, abstraktinen ja rekursiivinen ajattelu, käsitteellistäminen, suunnittelu sekä erilaiset esittämismuodot ongelmille ja tiedolle. Myös ohjelmoinnin rakenteita voidaan pitää matemaattisina kokonaisuuksina. (Hu, 2011, 225.) Algoritminen ajattelu täydentää ja yhdistää matemaattisen ja teknisen ajattelun, koska tietojenkäsittelytieteen formaalit perustat perustuvat matematiikkaan ja rakennetaan todellisen maailman kanssa vuorovaikuttavia järjestelmiä. Laskennallisuut-

ta tarvitaan matematiikan ohessa laskentalaitteiden rajoitusten huomioimiseen. (Wing, 2006, 33–35.) Algoritmiseen ajatteluun oletetaan liittyvän myös visuaalis-spatiaalisen päättelyn, joustavan älykkyyden ja työmuistin, joiden ajatellaan olevan kytköksissä ohjelmointiin (Ambrósio, Xavier, & Georges, 2014, 3–4, 6–7).

On tärkeää erottaa algoritminen ajattelu ja tietojenkäsittelytiede toisistaan (García-Peñalvo ym., 2016, 17). Tietojenkäsittelytiede viittaa tietokoneisiin ja algoritmisiin prosesseihin, joihin kuuluvat niiden periaatteet, laitteistojen ja ohjelmistojen suunnittelu, sovellukset ja niiden vaikutus yhteiskuntaan (Wilson, Sudol, Stephenson, & Stehlik, 2010, 10).

Perković, Settle, Hwang ja Jones (2010, 123) kuvaavat kolme tärkeää tietokonesovellusten ja tekniikoiden tehokkaaseen käyttöön tarvittavaa taitoa, joita tarvitaan eri aloilla. Ensimmäinen taito on tietotekninen lukutaito (engl. computer literacy), joka sisältää tietokoneiden perussovellusten, kuten editorien ja selainten käytön. Toinen taito on sujuvuus (engl. computer fluency), joka liittyy tietojärjestelmien toiminnan korkean tason ymmärtämiseen. Kolmas taito on algoritminen ajattelu. Se on kriittinen älyllisten taitojen ja päättelytaitojen joukko, joita ammattilaisen tarvitsee hallita soveltaakseen laskentatekniikoita tai tietoteknisiä sovelluksia oman alansa ongelmiin ja projekteihin. (Perković ym., 2010, 123.)

Vastaavasti Sysło ja Kwiatkowska määrittelevät tietoteknisen sujuvuuden vaatimuksiksi nykyaikaiset taidot (taito käyttää nykyaikaisia tietokonesovelluksia), perushallinta (tietokoneiden, verkkojen ja informaation perusteiden osaaminen) ja tiedollinen kyvykkyys (taito soveltaa informaatioteknologiaa monimutkaisissa tilanteissa). Lisäksi sujuvuus sisältää taidon käyttää algoritmista ajattelua (AT) ja ohjelmointia ongelmien ratkaisemisessa. (Sysło & Kwiatkowska, 2008, 6.) Yleensä algoritminen ajattelu ei ole sama asia kuin ajatteluprosessi, joka johtaa ohjelmointiin. Algoritminen ajattelu on joukko ajattelutaitoja, joita voidaan käyttää tietokoneohjelmien luomiseen. Syslon ja Kwiatkowskan mukaan algoritmisen ajattelun pitäisi keskittyä ohjelmointitaitojen sijaan enemmän laskennan periaatteisiin. (Sysło & Kwiatkowska, 2013, 50.) Ohjelmointi on Groverin ja Pean (2013, 40) mukaan kuitenkin perustavanlaatuinen taito tietojenkäsittelytieteissä ja algoritmisen ajattelun kognitiivisia tehtäviä tukeva työkalu.

Erona tieto- ja viestintätekniikan (ICT) ja algoritmisen ajattelun välillä on, että siirrytään TVT-työkalujen ja informaation käytöstä niiden luomiseen. Tämä erottaa myös TVT:n ja tietojenkäsittelytieteen toisistaan. (N. C. C. Brown ym.,

2013, 272–273; Sysło & Kwiatkowska, 2013, 49.) Digitaalinen lukutaito (engl. digital literacy) sisältää perustaidot tietokoneen ja perusohjelmistojen ymmärtämisestä ja käytöstä sekä tehokkaan, turvallisen ja eettisen käytön (esimerkiksi asiakirjan muokkaaminen, kommunikointi eri medioiden kautta, toisen työn laillisen käytön erottaminen plagioinnista sekä mittayksiköiden, kuten koko ja nopeus ymmärtäminen). (Gander ym., 2013, 8.)

2.7 Algoritmisen ajattelun integrointi eri oppiaineisiin

Wing (2006, 34) esittää, että tietojenkäsittelyä ja algoritmista ajattelua voidaan hyödyntää useilla eri aloilla, ja laskennallisuus muuttaa eri tiedeyhteisöjen ajattelutapaa. Matematiikka on aloista ilmeisin, mutta tietojenkäsittelyprosesseja tarvitaan myös luonnontieteissä ja yhteiskuntatieteissä. Tarkoituksenmukaiset mahdollisuudet pitäisi tunnistaa ja integroida ne jo korkeakoulutasoa alempaan opetukseen. Algoritminen ajattelu on integroitavissa matemaattisluonnontieteellisten aineiden lisäksi muihinkin aineisiin (Mannila ym., 2014, 15), vaikka integrointi matematiikkaan ja luonnontieteisiin on luonnollisinta (Weinberg, 2013, 61). (Lu & Fletcher, 2009, 262.) Barrin ja Stephensonin (2011, 52) mukaan algoritmista ajattelua voidaan hyödyntää tietojenkäsittelytieteissä, matematiikassa, luonnontieteissä, yhteiskuntaopissa, kielissä ja taiteissa, ja algoritmista ajattelua onkin kokeiltu peruskoulu- ja lukioikäisiä vastaavilla ikäryhmillä tietojenkäsittelyn, latinan, englannin, grafiikan ja historian oppikursseilla (Settle ym., 2012, 23–26).

Myös Guzdial (2008, 25, 27) opastaa tieteenalan mukaiseen ohjelmoinnin ja laskennan opetuksen tukemiseen. Mikäli jokaiselle opiskelijalle opetetaan ohjelmointia ja tietojenkäsittelyn teoriaa heidän alalleen järkevällä tavalla, niin kaikkialla olevan tietojenkäsittelyn ymmärtäminen edistää koko korkeakoululaitosta (Alan Perlisin 1962 tekemän ennusteen mukaan) (Guzdial, 2008, 25, 27). Tietojenkäsittelytieteilijän ajattelun sijaan pitäisi opettaa oman alan asiantuntijan ajattelua ja auttaa ymmärtämään, miten laskentaa voidaan käyttää oman alan ongelmien ratkaisemisessa, luomisessa ja uusien kysymysten löytämisessä. Algoritmisen ajattelun ominaisuuksien kertomisen sijasta pitäisi keskittyä laskennalliseen tekemiseen uudella tavalla käyttämällä tietojenkäsittelyn työkaluja. (Hemmendinger, 2010, 6–7.)

Lisäksi laskennalliset käsitteet tulisi opettaa alan kielen mukaan, esimerkiksi kemian opiskelussa integroituna kemiaan (Hambrusch ym., 2009, 184). Tietojenkäsittely ja algoritminen ajattelu käsitteineen yritetään integroida sujuvasti oppiaineiden opetukseen esimerkiksi tutkijoiden ja opettajien yhteistyöllä. Samalla oppilaat yritetään saada kiinnostumaan tietojenkäsittelystä. Tarkoituksena ei ole lisätä uutta sisältöä, vaan yhdistää tärkeitä laskennan osuuksia ja aktiviteetteja olemassa oleviin oppiaineiden aiheisiin. Useat käytetyt välineet perustuvat Python-ohjelmointikieleen. (Allan, Barr, Brylow, & Hambrusch, 2010, 390–391.)

2.8 Algoritmiseen ajatteluun vaikuttavia taustatekijöitä

Tutkijat ovat väitelleet siitä, minkä ikäisenä algoritmisen ajattelun ja ohjelmoinnin opetus pitäisi aloittaa. Wing (2008, 3720–3721) kannattaa aikaista algoritmisen ajattelun opetusta kertomalla, että mikäli halutaan varmistaa kaikille algoritmisen ajattelun ymmärtämisen ja soveltamisen yleinen ja vankka perusta, oppimisen pitäisi tapahtua varhaisessa lapsuudessa. Esimerkiksi ensimmäisillä luokilla voidaan hyödyntää tehokasta visualisointia sekä animaatiota, ja myöhemmillä luokilla oppilaat voivat itse ohjelmoida. (Wing, 2008, 3720–3721) Myös Lun ja Fletcherin (2009, 261) mukaan algoritmisen ajattelun opettaminen on aloitettava aikaisin ja usein. Painopisteen pitäisi olla laskennallisten prosessien (manuaalisen suorituksen) ymmärtämisessä, eikä ohjelmointikielissä. Tärkeitä taitoja ovat algoritmiset käsitykset, abstrahointi, tiedon esittäminen ja prosessien ominaisuuksien arvioiminen. (Lu & Fletcher, 2009, 261.) Dorlingin ym. (2015, 15) mukaan algoritminen ajattelu ei ole iästä riippuvaista, joten algoritmisen ajattelun käsitteitä ei ole määritetty esimerkiksi vuosiin, mutta käsitteet ovat sen sijaan riippuvaisia kyvyistä. Korkeakoulutettavien alhaisen määrän vuoksi algoritminen ajattelu pitäisi saada ja painottaa alemmille koulutustasoille (Settle ym., 2012, 22).

Sukupuoli ja muut taustat eivät saisi vaikuttaa kielteisesti alavalintoihin. Esimerkiksi luovuus on tärkeä osa CT:n taitojen valikoimaa. Näyttämällä, miten oikeat eri ammateissa toimivat algoritmisen ajattelun asiantuntijat, sekä miehet että naiset ja eri etnisen taustan henkilöt, ovat löytäneet luovia ratkaisuja reaalielämän ongelmiin, rikkoutuu stereotypioita ja ala houkuttelee myös naisia (Cur-

zon, Peckham, Taylor, Settle, & Roberts, 2009, 201). Myös García-Peñalvon, Reimannin, Tuulin, Reesin ja Jormanaisen (2016, 13) mukaan on tärkeää tunnistaa esimerkiksi naisten roolimalleja STEM-aineissa mahdollisimman varhaisessa vaiheessa. Tytöt pitäisi tutustuttaa varhaisessa vaiheessa tietojenkäsittelyyn jo ennen kuin he ymmärtävät, mitä tietojenkäsittelytiede on. Tämä auttaisi muuttamaan ennakkoluuloisen käsityksen siitä, että tietojenkäsittely ei sovi naisille. (Wilson ym., 2010, 11.) Myös Settlen ym. (2012, 22) mukaan pitäisi erityisesti keskittyä tyttöihin ja aliedustettuihin vähemmistöihin jo varhain.

Espinon ja Gonzálezin (2015, 1–2) tutkimuksessa opetusrobottien kilpailussa FLL:ssä (First Lego League) miespuolisia oppilaita oli huomattavasti enemmän kuin naispuolisia, vaikka tiedon prosessoinnissa ja oppimisessa ei havaittu sukupuolieroa. Heidän (2016, 1–2) mukaansa CT on otettu huomioon opetuksessa ja kasvatuksessa kansainvälisesti, mutta varsinaisia menetelmiä CT:n opettamiseen on harvoin, ja missään ei ole otettu sukupuoliperspektiiviä huomioon. CT-taitojen osaajia tarvitaan runsaasti ja tämän vuoksi olisi tärkeää saada laajennettua osallistumista CT-opintoihin etnisyydestä, sukupuolesta ja taloudellisesta asemasta riippumatta (Association for Computing Machinery (ACM) & IEEE Computer Society, 2013, 47).

Lapsia ja erityisesti tyttöjä voidaan rohkaista eri tavoin matemaattisten aineiden ja insinööritieteiden pariin. García-Peñalvon ym. (2016, 12–13) mukaan oppimisympäristöt voivat tarjota monimuotoisia materiaaleja, muotoja, värejä ja medioita, joita voidaan hyödyntää kiinnostuksen lisäämiseksi teknologiaa kohtaan. Tähän liittyy esimerkiksi lasten mielikuvitus. Lasten on saatava käyttöönsä sopivat työkalut tekniikan kokemiseen eri tavoilla. Uusien oppimisympäristöjen avulla voidaan vahvistaa lasten mielikuvitusta. (García-Peñalvo ym., 2016, 12–13.)

Raportti 46 erilaisesta eri aihepiirejä käsittelevästä sukupuolierojen meta-analyysista osoittaa, että matematiikan osalta (esimerkiksi mathematics computation, mathematics concepts, mathematics problem solving, mathematics, mathematics self-confidence ja mathematics anxiety) osalta havaitut erot olivat lähellä nollaa tai pieniä. Kuitenkin iän mukaisessa monimutkaisessa ongelmanratkaisussa erot kasvoivat poikien ja miesten hyväksi iän kasvaessa, vaikka erot olivat yhä pieniä. Tietokoneiden käytön (computer use: current, computer self-efficacy) osalta erot olivat pieniä tai kohtalaisia miespuolisten hyväksi. Raportis-

ta paljastuu myös, että itsetunnon tulokset olivat lähes aina miehille lievästi suotuisammat. Edellä mainitut tutkimukset ovat vuosilta 1988, 1990, 1995, 1997 ja 1999. (Hyde, 2005, 583, 585, 586, 588)

Hyden ja Mertzin (2009) mukaan sukupuolten välinen kuilu matematiikassa on kuitenkin kaventunut ajan mittaan eri tasoilla, ja heidän mukaansa se kaventuu edelleen tulevaisuudessa. Lisäksi miespuolisia suosiva sukupuolten välinen suhde ei esiinny kaikkialla. Tutkijoiden mukaan suhde korreloi hyvin maan sukupuoliyhdenvertaisuuden kanssa, joten erot johtuvat suurelta osin sosiokulttuurisista tekijöistä ja muista ympäristötekijöistä, ei biologiasta tai sukupuolesta sinänsä. Ongelmanratkaisutaito ja korkeatasoinen matemaattinen päättely ovat kuitenkin olennaisia taitoja menestymiseen elämässä ja STEM-aineiden uralla. (Hyde & Mertz, 2009.)

Myös Spelken (2005) mukaan poikien ja tyttöjen suorituskyky standardoiduissa testeissä heijastaa todennäköisesti monimutkaista sekoitusta yhteiskunnallisista, kulttuurisista ja biologisista tekijöistä. Kognitiivisen kehityksen ja biologisen perustan tutkimukset eivät selitä miesten enemmistöä matematiikan ja tieteen akateemisissa tiedekunnissa. Mies- ja naispuolisten kognitiivisten kykyjen tutkimus ei tue väitettä, jonka mukaan miehillä olisi suurempi sisäinen kyky matematiikkaan ja tieteisiin. Pienet tyttö- ja poikalapset eivät eroa kognitiivisissa kyvyissä matemaattisen ja tieteellisen ajattelun osalta, vaan heillä on yhtäläiset kyvyt kuvata ja oppia esineistä, numeroista, kielestä ja tilasta. Lukiotasolla laskentaluokilla voi olla enemmän poikia kuin tyttöjä, mutta sukupuolikuilu on kuitenkin suljettu. Korkeakoulutasolla miehet ja naiset osoittavat yhtäläistä matemaattista kyvykkyyttä esimerkiksi uuden haastavan matemaattisen materiaalin hallitsemisessa pitkällä aikavälillä. Miehet ja naiset näyttävät hieman erilaisilta kognitiivisilta profiileilta, mikäli ne esitetään monimutkaisten useilla strategioilla ratkaistavien tehtävien kautta. Sukupuolilla on kuitenkin samanlainen suorituskyky tehtävissä, joiden ytimessä on matemaattinen ajattelu. Lisäksi miehillä ja naisilla on yhtäläiset kyvyt oppia kehittynyttä, korkeakoulutason matematiikkaa. Siltä osin kuin matemaattinen kyky on keskeisessä asemassa opiskelijoiden edistykselle tieteissä, miehet ja naiset näyttävät olevan yhtä kykeneviä tieteiden oppimiseen. (Spelke, 2005, 955–956.)

Matemaattisen suorituskyvyn sukupuolieroja on myös havaittu, ja eroille on löydetty syitä. Ganleyn ja Vasiylevan (2014) tutkimuksessa tarkasteltiin mah-

dollisia mekanismeja, jotka ovat perustana sukupuolten välisiin eroihin matemaattisessa suorituskyvyssä. Suurempi ahdistus ja huoli voivat verottaa työmuistiresursseja, mikä voi johtaa matemaattisesti heikompaan suorituskykyyn. Tulokset osoittivat huomattavaa, naisille heikompaa, sukupuolten välistä eroa matemaattisessa suorituskyvyssä, ahdistuksessa ja visuospatiaalisessa työmuistissa. Matemaattisten ahdistuneisuusongelmien ratkaiseminen naisilla voisi tutkijoiden mukaan olla kauaskantoisia vaikutuksia. (Ganley & Vasilyeva, 2014.)

län, sukupuolen ja muiden taustatekijöiden lisäksi opettajien taidoilla on nähty olevan merkitystä algoritmiseen ajatteluun ja sen lähisisältöihin. Sysłon ja Kwiatkowskan (2013, 46) mukaan opettajat eivät ehdi käsitellä algoritmisia aihealueita. Opettajat pelkäävät algoritmisia aiheita, koska he eivät ole hyvin valmistautuneita alalla, eivätkä luota riittävästi algoritmisiin tietoihinsa ja taitoihinsa sellaisten opiskelijoiden kanssa, joilla on kokemusta ohjelmoinnista ja omista ohjelmista. (Sysło & Kwiatkowska, 2013, 46.)

Opettajankoulutuksella voisi olla merkitystä algoritmisen ajattelun kehittämisessä. Wilsonin ym. (2010, 13) mukaan opettajille suunnatun työpajan jälkikyselyn tuloksista havaittiin, että monet opettajat muuttivat tietojenkäsittelyn määritelmänsä yksinkertaisesta ohjelmoinnista laskennan perusteisiin ja muuttivat myöhemmin opetustaan. Ahamedin ym. (2010) tutkimuksessa kolmen päivän työpajassa 9-12 luokka-asteiden opettajille esiteltiin CT:tä eri STEM-aineissa. Työpajassa käsiteltiin työkaluja, esimerkiksi simulaatioiden kehittämistä Python-ohjelmointikieleen perustuvalla visuaalisella laajennoksella, VPythonilla. Tarkoituksena oli sisällyttää laskennallisia konsepteja suoraan tieteellisiin oppiaineisiin. Opettajilla ei ollut juurikaan aiempaa ohjelmointikoulutusta. Työpajan päätteeksi jokaisella opettajalla oli suunnitelma CT:n sisällyttämisestä oppiaineisiinsa, ja suurin osa oli valinnut VPython-simulaatiot. Lisäksi ennen ja jälkeen työpajan tehdyn kyselyn mukaan tutkijat arvioivat opettajien CT:n opetuksen ja CT-työkalujen käytön parantuneen. Vähäisestä ohjelmointikokemuksesta ja työpajan lyhyydestä huolimatta opettajat menestyivät Pythonin ja VPythonin osalta. (Ahamed ym., 2010.)

Yadavin, Hongin ja Stephensonin (2017) mukaan olisi otollista liittää opettajankoulutukseen algoritminen ajattelu. He suosittelevat 1) kehittämään opettajankoulutuksen opetussuunnitelmaa niin, että opettajat ovat valmistautuneita algoritmisen ajattelun sulauttamiseen opetuksessaan; 2) opettamaan opettajille

ydinsisällöt CT:stä suunnittelemalla teknologiakurssit uudelleen; 3) käyttämään menetelmäkursseja kehittämään ymmärrystä algoritmisesta ajattelusta oppiaineen kontekstissa; 4) pitämään yhteistyötä tietojenkäsittelyn ammattilaisten ja opettajankouluttajien kesken, jotta CT-opetussuunnitelma menisi ohjelmointia pidemmälle; 5) käyttämään olemassa olevia lähteitä ja opetussuunnitelmastandardeja CT:n omaksumiseen opettajankoulutuksessa. (Yadav ym., 2017, 60.)

2.9 Algoritmisen ajattelun oppimisen arviointi

Algoritmisen ajattelun oppimisen arvioinnissa on huomattavia puutteita. Groverin ja Pean (2013) mukaan algoritmista ajattelua koskeva tutkimus on keskittynyt pääasiassa määritelmiin ja algoritmista ajattelua kehittäviin keinoihin ja työkaluihin. Opetussuunnitelmauudistusten myötä on tehty kehitysaskeleita arviointimenetelmien luomiseksi, mutta suuria aukkoja on vielä olemassa. (S. Grover & Pea, 2013, 42.) Vastaavasti Denning (2017b) esittää ongelmakohdiksi algoritmisessa ajattelussa määritelmän lisäksi arviointimenetelmät ja yleistä etua koskevat väitteet. Ei ole selkeää yhteisymmärrystä arviointikriteereistä, ja nykyiset kriteerit ovat epäluotettavia. CT ei välttämättä ole hyödyllinen kaikille, koska muille kuin laskennallisia menetelmiä suunnitteleville ei ole saatu selkeää näyttöä hyödystä. (Denning, 2017b.)

Gouws, Bradshaw ja Wentworth (2013) esittävät kaksiulotteisen taulukon, jota voidaan käyttää algoritmisen ajattelun materiaalien suunnittelussa. Vertikaalisena ulottuvuutena taulukossa ovat keskeiset CT:n taidot (prosessit ja muunnokset, mallit ja abstraktiot, mallit ja algoritmit, työkalut ja resurssit, logiikka, arvioinnit ja parannukset). Horisontaalisena ulottuvuutena taulukossa kuvataan erilaisia tasoja, joissa CT:n taitoja voidaan harjoittaa (tunnistaminen, ymmärtäminen, soveltaminen ja omaksuminen). Kyseisen kehikon avulla tutkijat suorittivat myös arvioinnin Light-Bot-opetuspelissä. (Gouws ym., 2013, 10–14.)

Brennan ja Resnick (2012) ovat arvioineet määrittelemiään kolmea ulottuvuutta projektiportfolioanalyysillä, tuoteperustaisilla haastatteluilla ja suunnittelunäkymillä. He ovat esittäneet myös kuusi ehdotusta ohjelmoinnin avulla tapahtuvan CT:n arvioimiseksi. Näitä ovat 1) oppilaan oppimisen tukeminen jatkossa (esimerkiksi arvioinnin tulisi olla merkityksellistä ja arvokasta oppilaalle), 2) tuotteiden sisällyttäminen arviointiin (esimerkiksi oppilaan kehityksen arviointi useita

tuotoksia tarkastelemalla), 3) oppilaan oma prosessin esille tuominen (esimerkiksi kommentoimalla koodia, tekemällä ohjelmointiprojektista erilaisia esityksiä ja opettamalla muita oppilaita – samalla kehitetään oppilaiden metakognitiivista toimintaa ja oman ajattelun ajattelua), 4) oppilaan arviointi useissa kohdissa prosessia ja oppilaan kehityksen kuvailu, 5) useiden tietämisen tapojen arvostaminen arvioinnissa (esimerkiksi erilaisten kykyjen monipuolinen arviointi) sekä 6) useiden näkökulmien sisällyttäminen arviointiin (esimerkiksi itse- ja vertaisarviointi, opettajien, perheen, ammattilaisten ja yhteisöjen antama arviointi). (Brennan & Resnick, 2012, 1, 22–24.)

Webb (2010) on arvioinut CT:tä AgentCubes-ohjelmointiympäristössä virheenetsintään perustuvan autenttisen ongelmanratkaisun avulla. Webbin mukaan ongelmallista on, että ennakkoarviointi on hankalaa, koska oppilaiden pitää tutustua riittävästi käytettävään ohjelmointiympäristöön, ja tämän vuoksi loppuarviointi on järkevämpää (Webb, 2010, 904). Ongelmanratkaisulla, ohjelmoinnilla ja CT:llä on yhteisiä taitoalueita, mutta tutkimuksesta ei käy ilmi mitä keskeisiä CT:n taitoja arvioitiin, ja oliko tutkimuksen tehtävissä myös soveltavampia ongelmia suhteessa oppilaiden aikaisemmin käsittelemiin tehtäviin. Arviointi oli tehty yleisellä tehtävätasolla, eikä arvioinnissa kuvattu CT:n eri osa-alueita.

Werner, Denner ja Campe (2012) toteuttivat arvioinnin peliohjelmoinnissa 10-14-vuotiaille. Tutkimuksessa luotiin Alice-ohjelmaan arviointimetodi (the Fairy Assessment), jonka avulla analysoitiin algoritmista ajattelua (AT) sekä abstraktioiden ja mallien tehokasta käyttöä. Kaikki osa-alueet arvosteltiin asteikolla 0-10 osatulosten mukaan. Pisteet eivät korreloineet sukupuolen, iän, luokassa läsnäoltujen tuntien mukaan tai sen mukaan kuinka usein oppilaat tekevät asioita tietokoneella luokan ulkopuolella. Löydökset viittaavat siihen, että the Fairy Assessment on lupaava strategia CT:n arviointiin. Lisäksi arviointi on motivoiva ja havaitsee CT:n osa-alueita ja osaamisen vaihtelua. (Werner ym., 2012, 215–219.)

Pariohjelmoinnissa oppilaat saivat merkitsevästi korkeammat arvosanat. Mitä enemmän parityöskentelyä oli, sitä parempi oli CT-suoriutuminen. Lisäksi molemmat parin oppilaista, riippumatta lähtötasosta, saivat paremmat pisteet kuin yksin työskennelleet. Oppilaat, jotka käyttivät enemmän tietokonetta tai ovat saaneet parempia arvosanoja, olivat myös parempia CT:ssä. Itseluottamus tietokoneiden käyttämiseen oli tärkeä suoriutumista ennustava tekijä. Tutkijat ar-

vioivat, että tämä voisi johtua sietokyvystä kohdata haastavia tehtäviä. (Werner ym., 2012, 215–219.)

Basun ym. (2014) tutkimuksessa käytettiin CTSiM-opetusympäristöä (Computational Thinking using Simulation and Modeling), jonka avulla oppillaat loivat omia malleja. Opetusympäristöön perehdytettiin ennen tutkimusta. Arvioinnit tehtiin kurssia ennen ja sen jälkeen. Tutkimuksessa käytetyn arvioinnin avulla havaittiin merkitsevä paraneminen sekä tieteellisessä tiedossa että CT-taidoissa. Runsasta CT-käsitteiden käyttöä suositaan, mutta sen todellista vaikutusta lopullisiin tuotoksiin, kuten peleihin, tai oppimistavoitteiden ja lopullisten tuotosten väliseen suhteeseen harvoin otetaan huomioon. (Basu ym., 2014, 476–477, 481.)

Algoritmisen ajattelun malleja (computational thinking patterns, CTP) käytettäessä on myös toteutettu automaattinen arviointi, jolla erilaisia ohjelmoituja pelituotoksia analysoidaan, kuvataan graafisesti ja verrataan malliin, ja pisteytetään tuotokset käytettyjen menetelmien mukaan. Tarkoituksena on ollut huomioida arvioinnissa sekä syntaktinen että semanttinen muoto. (Koh, Basawapatna, Bennett, & Repenning, 2010, 60–66.) Basawapatnan ym. tutkimuksessa luotiin malli CTP:n arviointiin. Tutkimuksessa suunniteltiin pelejä ja tämän jälkeen osaamista testattiin eri mallinnuskontekstissa. Esimerkiksi tositilanteista kuvatuista videoista haettiin yhteneväisyyksiä ohjelmointiin. Näin pyrittiin arvioimaan siirtovaikutusta. (Basawapatna ym., 2011; Marshall, 2011.)

Dr. Scratch on analyyttinen arviointityökalu, jolla voidaan arvioida Scratch-projekteja ja niiden sisältämää algoritmista ajattelua ja koodia. Oppilaat voivat oppia parantamaan ohjelmointitaitojaan arviointityökalun avulla. (Dr. Scratch: drscratch.org, 2014.) Lisäksi Dr. Scratch tukee opettajia arviointitehtävissä. Dr. Scratchiä on myös arvioitu 10-14-vuotiailla oppilailla, joilla oli aiempaa Scratchin käyttökokemusta. Oppilaat arvioivat Dr. Scratchia positiivisesti, ja se edisti lisäksi heidän haluaan parantaa ohjelmointitaitojaan. Lisäksi oppilaat kykenivät parantamaan projektiaan Dr. Scratchin palautteen perusteella, mutta Dr. Scratch edisti vähemmän niiden oppilaiden projekteja, jotka olivat jo ennen arviointia hyviä. Lisäksi 10-14-vuotiaat oppilaat saivat parannettua projektejaan nuorempia oppilaita enemmän. (Moreno-León, Robles, & Román-González, 2015, 1, 8–17.) Dr. Scratchin arviointimekanismia on myös arvioitu eri ohjelmistojen moni-

mutkaisuusmittareilla, ja on saatu merkitsevä positiivinen korrelaatio (Moreno-Leon, Robles, & Roman-Gonzalez, 2016, 1041, 1044).

Dr. Scratch arvioi ohjelman koodia ja antaa pisteet eri osaamisalueista. Ohjelma etsii virheitä ja huonoja ohjelmointitapoja, esimerkiksi koodia, jota ei koskaan suoriteta. Dr. Scratch myös sisältää selitysohjeen virheistä ja huonoista ohjelmointitavoista, ja miksi kyseisiä tapoja on syytä välttää. (Moreno-León & Robles, 2015b, 132.) Dr. Scratchin heikkouksina on pidetty esimerkiksi seuraavia asioita: yhden Scratch-projektin arviointi ei riitä antamaan kokonaiskuvaa oppilaan CT-taidoista, tiettyjen lohkojen tai lohkoryhmien käyttö ei riitä vahvistamaan tietyn CT:n käsitteen sujuvuutta ja joitain keskeisiä CT:n pätevyyksiä ei voida mitata analysoimalla projektin koodia, kuten virheenkorjaus- tai uudelleenmiksaustaitoja (engl. remix). Lisäksi yksinkertainen sopivat lohkot sisältävä projekti voi saada korkean CT-pistemäärän, vaikka sen toiminnot olisivat hyödyttömiä. Myöskään omaperäisyyttä tai luovuutta ei arvioida. Opettajien ei pitäisi luottaa vain Dr. Scratchin antamiin pisteisiin. (Moreno-León & Robles, 2015a, 5–6.)

Tietojenkäsittelytieteessä ja erityisesti CT:ssä yhtenäinen arviointi on yhä ongelma. Käytännössä arviointimetodia tietojenkäsittelytieteen opetukselle ei ole olemassa (Wilson ym., 2010, 14). CT:hen liittyvät arvioinnit keskittyvät usein tiettyyn työkaluun sekä CT:n tiettyihin määritelmiin ja sisältöihin, jolloin arviointimenetelmä ei ole yleistettävissä. Lisäksi arviointimenetelmiä tulisi kehittää pidemmälle. Niissä on eri painotuksia, esimerkiksi ne voivat arvioida ohjelmointirakenteiden käyttöä, käsitteiden määrää ja kategorioita. On ongelmallista, että eri maissa eri ikäluokille CT ja tietojenkäsittely on määritelty opetussuunnitelmissa eri tavoin. Eri koulujärjestelmät ovat myös erilaisia ja koulua käydään eri iässä, joten vaatimukset eri opetussuunnitelmissa eivät kohtaa. Tämän vuoksi on myös vaikea arvioida minkä tasoiset tehtävät sopivat tietyn ikäisille oppilaille.

2.10 Algoritminen ajattelu ja ohjelmointi opetussuunnitelmissa

Tarkastelussa ovat Suomen vuosien 2004 ja 2014 opetussuunnitelmat luokille 7-9 sekä lukion opetussuunnitelman perusteet 2015. Perusopetuksen osalta on tarkasteltu vanhempaa ja uudempaa opetussuunnitelmaa, koska tutkielman opetuskokeilun aikana myös vanha vuoden 2004 opetussuunnitelma on ollut

vielä porrastetusti voimassa. Seitsemännelle vuosiluokalle on otettu uusi opetussuunnitelma käyttöön 1.8.2017 ja kahdeksannelle vuosiluokalle 1.8.2018. Yhdeksännelle vuosiluokalle opetussuunnitelma otetaan käyttöön 1.8.2019. Vanhaa ja uutta perusopetuksen opetussuunnitelmaa on verrattu toisiinsa. Lukion osalta käsitellään vain opetussuunnitelman perusteita 2015, koska tutkielman opetuskokeilussa olivat mukana 1. ja 2. vuosiluokan lukion opiskelijat, joten kaikilla oli voimassa lukion uusi opetussuunnitelma. Huomiota on kiinnitetty opetussuunnitelmien kohtiin, joiden voidaan tulkita liittyvän algoritmiseen ajatteluun ja ohjelmointiin.

Kaikkien opetussuunnitelmien oppimiskäsityksissä painotetaan oppilaan aktiivista toimintaa. Oppiminen sisältää ongelmanratkaisua, joka voidaan suorittaa itsenäisesti tai yhdessä muiden kanssa. Yhdessä oppimisen ja ongelmanratkaisun nähdään palvelevan algoritmiselle ajattelulle läheisiä ajattelumuotoja, kuten luovaa ja kriittistä ajattelua. (Opetushallitus, 2004, 18, 2015, 14, 34, 2016, 17, 19.) Ryhmässä toimimista, projektityöskentelyä ja verkostoitumista suositaan opetussuunnitelmissa. Yksittäisen oppilaan muodostama tehtäväosuus on osa ryhmän työn kokonaisuutta. (Opetushallitus, 2016, 23–24.) Perusopetuksen uuden opetussuunnitelman (Opetushallitus, 2016, 17, 19) mukaan oppilaan ajattelemista, suunnittelua ja tutkimista pitäisi edistää, ja lisäksi oppilasta tulisi kannustaa luomaan uutta. Oppilaan oppimisen suunnittelulle, reflektoinnille ja prosessien arvioinnille on myös annettu tilaa. Erityisesti matematiikalle ominainen tiedon kumuloituminen on otettava huomioon, ja opetuksessa opittavat asiat sekä käsitteet yritetään liittää osaksi aiempaa tietoa. Perusopetuksen uusi opetussuunnitelma sekä lukion opetussuunnitelma korostavat myös joitakin algoritmiselle ajattelulle tyypillisiä taipumuksia, kuten sinnikkyyttä ja pitkäjänteisyyttä (Opetushallitus, 2015, 129, 2016, 128). Lisäksi opetuksessa on otettava huomioon esimerkiksi oppilaan itsetunto ja minäpystyvyys. (Opetushallitus, 2016, 17, 19.)

Vanhan perusopetuksen opetussuunnitelman (Opetushallitus, 2004, 19) mukaan työtapojen tulisi kehittää tiedon hankkimisen, soveltamisen ja arvioimisen taitoja. Uusi opetussuunnitelma lisää edellisiin vielä tiedon käsittelyn, analysoimisen, esittämisen, yhdistelemisen ja luomisen (Opetushallitus, 2016, 30). Nämä periaatteet sisältyvät myös algoritmiseen ajatteluun. Uuteen opetussuunnitelmaan (Opetushallitus, 2016, 30) on työtapoihin lisätty myös ongelmalähtöi-

syys, leikki, mielikuvitus ja taiteellisuus. Uuden opetussuunnitelman työtavoissa otetaan vanhaa opetussuunnitelmaa voimakkaammin esiin sukupuolineutraalisuus ja eriyttäminen. (Opetushallitus, 2016, 30.) Eriyttämistä voidaan toteuttaa algoritmisessa ajattelussa esimerkiksi valitsemalla erilaisia ohjelmointikieliä eritasoisille oppilaille, antamalla vaihtelevan laajuisia ja sisältöisiä algoritmisen ajattelun ongelmatehtäviä tai projekteja.

Uudessa opetussuunnitelmassa (Opetushallitus, 2016, 20–21, 281–282) mainitaan myös algoritmiselle ajattelulle tyypilliset suunnittelu, prosessin arviointi ja oman toiminnan arviointi, tavoitteellinen työskentely sekä tiedonkäsittelyyn liittyvät osa-alueet. Olennaisia ovat oppilaiden ideointi, päättely, ongelmanratkaisu, uuden tiedon rakentaminen, perustelutaidot sekä luovat ja vaihtoehtoiset ratkaisut. Asioiden välisten vuorovaikutussuhteiden ja yhteyksien havaitseminen on tärkeä osa ajattelutaitoja. (Opetushallitus, 2016, 20–21, 281–282.)

Vanha perusopetuksen opetussuunnitelma kehottaa tietokoneiden ja mediatekniikan käyttöön, ja uusi perusopetuksen opetussuunnitelma korostaa tieto- ja viestintätekniikan käyttöä ottaen huomioon luovat ratkaisut, asioiden tutkimisen ja oppimisympäristöjen monipuolisuuden (Opetushallitus, 2004, 18, 2016, 29). Kaikki opetussuunnitelmat tuovat esiin teknologian kehityksen ja vaikutuksen eri elämänalueilla ja eri ympäristöissä – arjessa, opiskelussa, työssä ja yhteiskunnassa (Opetushallitus, 2004, 42, 2015, 39, 2016, 22–23). Vanhassa perusopetuksen opetussuunnitelmassa korostetaan, että oppilaat oppivat käyttämään välineitä, laitteita, ohjelmia, tietoverkkoja ja tietotekniikkaa. Lisäksi tuetaan teknologisten ideoiden kehittämistä, mallintamista ja arviointia. (Opetushallitus, 2004, 42–43.)

Uudessakin perusopetuksen opetussuunnitelmassa oppilasta tuetaan tieto- ja viestintätekniikan ymmärtämisessä ja sen käytön opettelussa, mutta huomioon otetaan myös ideointi, ajattelutaidot, luovuus, oivaltaminen ja motivaatio (Opetushallitus, 2016, 23). Pelkästä tekniikan käytöstä siirrytään tekniikan valjastamiseen ja uuden luomiseen. Painopisteessä on tieto- ja viestintätekniikan hyödyntäminen, sopivien välineiden valinta, tiedonhallinta, mallintaminen, digitaalisten asioiden jakaminen ja digitaalisten tuotosten laatiminen. Ohjelmointi sisältyy uudessa opetussuunnitelmassa osaksi eri oppiaineita. (Opetushallitus, 2016, 283–284.) Lukion opetussuunnitelma keskittyy myös uuden tiedon ja teknologian luomiseen, sillä opetussuunnitelmassa mainitaan teknologian kehittä-

mismahdollisuudet ja teknologian merkitys suhteessa yhteiskuntaan, ympäristöön, ongelmanratkaisuun, tieteisiin ja luovuuteen (Opetushallitus, 2015, 15, 39).

Matematiikan oppiaineen osalta ongelmanratkaisu ja ajattelutaidot ovat keskeisessä osassa opetussuunnitelmia. Oppilaan tulisi tehdä suunnitelma ongelman ratkaisemiseksi, ratkaista ongelma jäsennellysti ja perustellusti, esittää ratkaisuvaihtoehdot järjestelmällisesti ja tarkistaa ratkaisun oikeellisuus arvioimalla ratkaisua kriittisesti (Opetushallitus, 2004, 165, 2015, 131, 2016, 374). Oppimisen kannalta tärkeitä ovat luova, looginen ja täsmällinen matemaattinen ajattelu sekä opetuksen yhteys konkretiaan ja arjen ongelmiin (Opetushallitus, 2004, 158, 163, 2015, 129, 2016, 374, 376). Oppilaan tulisi kyetä havaitsemaan yhteyksiä, riippuvuuksia ja säännönmukaisuuksia sekä soveltamaan erilaisia menetelmiä tiedonkäsittelyn eri vaiheissa. Tärkeää olisi myös edistää oppilaiden päättelykykyä ja perustelutaitoa. (Opetushallitus, 2004, 163, 2016, 374–375.) Matematiikan oppiaineessa tarvittavaa kyvykkyyttä ja algoritmisen ajattelun taipumuksia yhdistävät keskittyminen, pitkäjänteisyys sekä yhteistyötaidot. Matematiikan opetuksessa tulisi vahvistaa muun muassa oppilaiden motivaatiota, positiivista minäkuvaa ja itseluottamusta. (Opetushallitus, 2004, 163, 2015, 129, 131, 2016, 374–375.)

Matematiikassa tulee hyödyntää tarkoituksenmukaisesti tieto- ja viestintäteknologiaa. Esimerkiksi mallinnusta voidaan käyttää apuna. (Opetushallitus, 2004, 158, 163–164, 2016, 374, 376.) Lukion opetussuunnitelman mukaan oppilaan teknisten apuvälineiden käyttöä tuetaan matemaattisissa ongelmissa ja lisäksi välineitä arvioidaan. Matematiikan opetuksen tavoitteisiin kuuluu oppilaan kyky mallintaa ongelmatilanteita ja käyttää erilaisia menetelmiä ja ratkaisustrategioita. Tavoitteisin kuuluvat myös positiiviset oppimiskokemukset ja ajatus matematiikasta apuvälineenä esimerkiksi ilmiöiden mallintamisen muodossa. Suuria eroja pitkän ja lyhyen matematiikan tavoitteiden välillä ei ole algoritmisen ajattelun sisältöjen näkökulmasta. (Opetushallitus, 2015, 129, 131, 136.)

Vanha perusopetuksen opetussuunnitelma ei sisällä terminä algoritmista ajattelua, toisin kuin muut opetussuunnitelmat. Vain perusopetuksen uusi opetussuunnitelma sisältää ohjelmoinnin käsitteen. Ohjelmointi kuuluu erityisesti osaksi 3-6-luokkien matematiikan oppiainetta, ja opetuksessa on tarkoitus käyttää graafisia ohjelmointiympäristöjä (Opetushallitus, 2016, 235, 239). Opetus-

suunnitelmassa 7-9-luokkien opetuksessa on puolestaan mainittu ohjelmointi (mutta ei enää graafisten ohjelmointiympäristöjen käyttöä). Matematiikan opetuksen tavoitteisiin kuuluu: *”– – ohjata oppilasta kehittämään algoritmista ajatteluaan sekä taitojaan soveltaa matematiikkaa ja ohjelmointia ongelmien ratkaisemiseen”*. (Opetushallitus, 2016, 375, 379.) Matematiikan tavoitteisiin liittyviin keskeisiin sisältöalueisiin sisältyy:

> ”Syvennetään algoritmista ajattelua. Ohjelmoidaan ja samalla harjoitellaan hyviä ohjelmointikäytäntöjä. Sovelletaan itse tehtyjä tai valmiita tietokoneohjelmia osana matematiikan opiskelua.” (Opetushallitus, 2016, 375.)

Matematiikan päättöarvioinnin yhtenä kriteerinä hyvälle osaamiselle on algoritmisen ajattelun ja ohjelmointitaitojen osa-alue, ja arvosana 8 täyttyy, jos: *”Oppilas osaa soveltaa algoritmisen ajattelun periaatteita ja osaa ohjelmoida yksinkertaisia ohjelmia”* (Opetushallitus, 2016, 378–379). Lisäksi ohjelmointi on mainittu uuden opetussuunnitelman 7-9-luokkien käsitöissä: *”Käytetään sulautettuja järjestelmiä käsityöhön eli sovelletaan ohjelmointia suunnitelmiin ja valmistettaviin tuotteisiin” (Opetushallitus, 2016, 431)*.

Lukion matematiikan pitkä oppimäärä sisältää syventävän kurssin ”*Algoritmit matematiikassa (MAA12)*”. Tavoitteena on algoritmisen ajattelun syventäminen, algoritmien toiminnan tutkiminen ja selittäminen, iteroinnin ymmärtäminen sekä teknisten apuvälineiden käyttö algoritmien tutkimisessa. (Opetushallitus, 2015, 135.) Tulevaankaan lukion opetussuunnitelmaan 2019 (astuu voimaan vuonna 2021) ohjelmointia ei ole ainakaan toistaiseksi sisällytetty matematiikan syventävää kurssia (MAA11) lukuun ottamatta (Opetushallitus, 2019). Suuria eroja nykyisen lukion opetussuunnitelman ja tulevan opetussuunnitelmaluonnoksen välillä ei vaikuttaisi algoritmisen ajattelun osalta olevan.

Lisäksi oppiainerajat ylittävä työskentely, eheytys ja kokonaisuuksien integrointi on ilmaistu opetussuunnitelmissa usein. Näihin saattaa sisältyä yhteistyötä, pidempiaikaista työskentelyä ja projektiluonnetta sekä tieto- ja viestintätekniikan hyödyntämistä. Toteutuskeinoiksi on mainittu esimerkiksi teemapäivät ja -opinnot, tapahtumat ja opintokäynnit. (Opetushallitus, 2015, 34, 220, 2016, 19, 31, 282, 389.) Algoritmisessa ajattelussa nähdään tärkeäksi ammattilaisten kanssa tehtävä yhteistyö, autenttisuus ja yhteys tieteisiin. Uuden perusopetuksen opetussuunnitelman (Opetushallitus, 2016, 390–391, 394, 396) fysiikan ja

kemian ainekokonaisuuksissa on viittauksia myös luontevaan tieto- ja viestintätekniikan käyttöön sekä ilmiöiden mallinnukseen ja simulaatioihin.

2.11 Eri valtioiden opetussuunnitelmat

Vuonna 2015 tehdyn 21 Euroopan maata koskevassa raportissa (Balanskat & Engelhardt, 2015) käsitellään koodaamisen ja algoritmisen ajattelun sisällyttämistä opetukseen. Termejä on käytetty eri maissa eri tavalla ”coding” (noin 6 maata), ”programming” (noin 13 maata), ”computing” (1 maa) ja ”computational thinking” (noin 6 maata) algoritmeihin ja robotiikkaan viittaavia termejä (noin 4 maata) ilmaistaessa taitojen sisällyttämisestä opetukseen (Balanskat & Engelhardt, 2015, 27). Eri maiden opetuksessa painotetaan eri suuntauksia. Jotkin maat painottavat ICT:n ja digitaalisten taitojen ja käytön kehittämistä (yli 10 maata), jotkin koodaamista (10 maata) ja toiset algoritmista ajattelua (5 maata) (Balanskat & Engelhardt, 2015, 8).

Koodaaminen on myös useissa maissa sisällytetty opetussuunnitelmaan (18 maata) (Balanskat & Engelhardt, 2015, 9–10). Esimerkiksi Venäjä, Etelä-Afrikka, Uusi-Seelanti ja Australia ovat liittäneet tietojenkäsittelytieteitä K12-ikäisten opetussuunnitelmiinsa (S. Grover & Pea, 2013, 40). Kahdessa valtiossa ja Belgian Valloniassa ei ole suunnitelmia lisätä koodausta opetussuunnitelmaan. Tavoitteena on ollut lisätä loogista ajattelun ja ongelman ratkaisun taitoja, sekä kehittää 2000-luvun taitoja. Monissa maissa tietojenkäsittelytieteet ovat saatavissa koulussa jo varhaisessa iässä. Koodaaminen on sisällytetty yleensä toiseen asteeseen (secondary level), mutta useat myös peruskouluasteelle (primary level), kuten Suomessa, joissakin molempiin, ja jopa päiväkoti-ikäisille (esimerkiksi Puolassa). (Balanskat & Engelhardt, 2015, 9–10.)

Osassa maista koodaaminen on valinnaista ja osassa pakollista. Koodaaminen tai algoritminen ajattelu voivat esiintyä yhdistettynä eri aineisiin (esimerkiksi matematiikka, fysiikka, kemia, tietotekniikan kaltaiset aineet) tai muodostaa oman aineensa. Pakollisuus ja valinnaisuus voivat vaihdella myös tietyillä koulutustasoilla (Heintz ym., 2016, 9). Jotkin maat arvioivat koodaamistaidot osana oppilaiden yleistä arviointia (13 maata), toiset taas oppiaineen taitojen osana (Suomi ja kaksi muuta maata). Lisäksi eri maissa painotetaan eri sisältöjä opetuksessa. Suomi on raportin mukaan ensimmäinen maa, jossa koodaaminen on

huomioitu koko opetussuunnitelmassa kaikkien aineiden osalta. (Balanskat & Engelhardt, 2015, 10–12, 24–25.) Suomessa valinnainen tietotekniikan oppiaine on korvattu integroimalla tieto- ja viestintäteknologiaa kaikkiin oppiaineisiin (Mannila ym., 2014, 4). Joissakin kouluissa tietotekniikkaa tarjotaan kuitenkin esimerkiksi kursseina tai koulun jälkeen järjestettävänä harrastustoimintana.

Eri maissa koodaamisen sisällöt on liitetty opetukseen myös hyvin eri aikoina. Israel on liittänyt koodauksen opetukseen jo vuonna 1976, kaksi maata 80-luvulla ja kolme 90-luvulla ja seitsemän maata 2012-2016 vuosien välillä. (Balanskat & Engelhardt, 2015, 36–37.)

Yhdysvalloissa opetussuunnitelma vaihtelee merkittävästi eri osavaltioissa, mutta vuonna 2009 on tehty yhdenmukaistuksia. K-12 opetussuunnitelmassa ovat englannin kielen taide/lukutaito ja matematiikka. Lisäksi on standardeja historiaan ja yhteiskuntatieteisiin, tieteisiin ja teknisiin aineisiin. Mainintoina ovat muun muassa median integrointi, teknologian ja internetin käyttö, digitaalisten lähteiden hakeminen ja kriittinen arviointi. Lisäksi esitetään CT:hen viittaavia näkökulmia, kuten mittaaminen ja datan esittäminen, sekä ongelmien ratkaiseminen. Tietojenkäsittelytieteen kursseja on pyritty saamaan lukioon valittaviksi kursseiksi. Lisäksi on kehitetty vaihtoehtoinen tietojenkäsittelytieteiden perusteiden kurssi niille, jotka eivät ole ohjelmoinnista kiinnostuneita. Kurssilla käydään abstraktiota, big dataa, algoritmeja ja internetiä, ja ensimmäinen koe järjestettiin 2016. (Mannila ym., 2014, 8–9.)

Yhdysvalloissa tietojenkäsittelyn oppimisstandardit ikäluokittain (Wilson ym., 2010, 27–29) ovat tarkoin määritellyt, ja sisällöt vaikuttavat haastavilta verrattuna Suomen opetussuunnitelman perusteiden löyhiin oppimistavoitteisiin ja sisältöjen kuvauksiin. Lisäksi tietojenkäsittelyä on Yhdysvalloissa myös lukioikäisillä. Standardit ja niiden osa-alueet (konseptit, kyvyt, taidot) on kuitenkin saavutettu Yhdysvalloissa vaihtelevasti eri osavaltioita tarkastelemalla (Wilson ym., 2010, 34–45).

Jonesin (2011) maavertailun mukaan Iso-Britannian alueella ja Yhdysvalloissa on eri tasoisia kursseja, kokeita ja kokonaisuuksia (esim. GCSE:t ja A-tasot, The International Baccalaureate, Advanced Placement exams, AP's). GCSE:t ja International Baccalaureate saavuttaneet myös suosiota Intiassa. Israelissa erityistä ovat tietyt vaatimukset tietojenkäsittelyn opettajille. Uudessa-Seelannissa keskitytään opettajankoulutukseen ja materiaaleihin, Kreikassa

puolestaan tietojenkäsittelytieteiden opettajien laatu on yleisesti korkea, mutta pedagoginen osaaminen ei ole niin vahvaa. Etelä-Koreassa on voimakas painotus etiikkaan ja verkkorikollisuuteen kaikilla koulutasoilla, ja ohjelmointi kuuluu osaksi opetusta noin yläasteikäisestä alkaen. (Jones, 2011, 3–11.) Myös Neittaanmäen, Lehdon ja Kankaanrannan raportissa on tarkasteltu neljän eri valtion TVT-osaamistaitoja. Osaamistaitojen aihealueet ja sisällöt poikkeavat eri valtiossa toisistaan. (Neittaanmäki ym., 2014, 23–34.)

Jonesin mukaan tietojenkäsittelytiede on paljon muutakin kuin ohjelmoinnin oppimista. Algoritmien perusteiden, tietorakenteiden ja laskennallisen ajattelun osaaminen ovat perustavanlaatuisempia ja kestävämpiä taitoja. On suurta tarvetta opettajien koulutukselle tietojenkäsittelytieteissä. (Jones, 2011, 2.)

Suomessa ohjelmointi on sisällytetty erityisesti matematiikan oppiaineeseen, mutta sitä pitäisi integroida myös muihin oppiaineisiin. Luokilla 1-2 on esimerkiksi ohjelmointiin liittyvää toisten oppilaiden ohjeistusta, luokilla 3-6 voidaan käyttää graafisia ohjelmointiympäristöjä ja luokilla 7-9 ohjelmointikieliä. Ohjelmoinnin opetuksen kehittämiseksi on myös julkaistu Koodi2016 ja Opetus- ja kulttuuriministeriön tukema LUMA SUOMI -projekti. (Mannila ym., 2014, 5.) Vertailu kansallisten opetussuunnitelmien välillä on hankalaa, koska koulutusjärjestelmien koulutustasot ovat limittyneet eri ikäryhmille eri tavoin. Suomen opetussuunnitelmassa ei määritellä tarkasti algoritmisen ajattelun sisältöjä, vaikka TVT-taidot on otettu huomioon. Eri valtioiden opetussuunnitelmien sisältöjen sopivuutta vaatimustasoltaan Suomen koulujen 7-9-luokkalaisille ja lukiolaisille on haastavaa selvittää.

3 OHJELMOINTI

Ohjelmoinnilla tarkoitetaan tietokoneohjelman laatimisprosessia. Tietokoneohjelman perustana on ongelma tai tehtävä, jota yritetään ratkaista tietokoneen avulla. Ongelma on tärkeää määritellä ensin formaalimpaan muotoon, ja tehdä suunnitelma ongelman ratkaisemiseksi. Tiettyyn ongelmaan voidaan suunnitella ja luoda algoritmi, jolla ratkaistaan kyseinen ongelma. Usein voidaan käyttää myös valmiita algoritmeja, joita voidaan soveltaa ongelmaan. Algoritmien toiminta ongelman ratkaisussa tulee analysoida ja validoida, jotta algoritmin toiminta vastaa ratkaistavaa ongelmaa.

Kokonaisen tietokoneohjelman laatiminen saattaa olla monimutkainen prosessi. Yksittäisen laajan ja monimutkaisen ongelman ratkaisemisen sijaan ongelma on järkevää hajottaa osaongelmiksi. Tietokoneohjelmassa on yleensä useita toimintoja, jotka voidaan ratkaista osaongelmina. Osaongelmat ratkaistaan algoritmeilla, joten tietokoneohjelma koostuu useista algoritmeista. Algoritmit puolestaan sisältävät useita funktioita (palauttavat arvoja) ja proseduureja (lausesarja), jotka ratkaisevat tai toteuttavat algoritmin osia. Tietokoneohjelma sisältää siis algoritmeja, jotka puolestaan koostuvat vaiheittaisesta sarjasta käskyjä ongelman ratkaisemiseksi.

Jotta tietokone kykenee ratkaisemaan ongelman luodun algoritmin avulla, algoritmi tulee suunnitella tietokoneelle ymmärrettävään muotoon. Tietojenkäsittelyn alkuvaiheessa tietokoneelle ohjelmoitiin komentoja konekielellä (engl. machine code), joka koostuu binäärikoodista (0 ja 1). Binäärikoodi korvattiin assemblerilla, joka on symbolinen konekieli. Myöhemmin kehitettiin kehittyneempiä ohjelmointikieliä, jotka kääntäjän (engl. compiler) avulla käännetään konekieliseen muotoon. Osa käännetään valmiiksi konekieliseen muotoon ennen suoritusta (esimerkiksi kielet C ja C++) ja osa vasta ohjelmaa suoritettaessa (esimerkiksi JavaScript).

Konekielen avulla ohjelman toimintoja saadaan suoritettua koneen eri fyysisten osien yhteistoimintana. Prosessorin laskentatehoa käytetään tehokkaasti ja oikea-aikaisesti eri laskentatehtävien suorittamiseen, ja muuttujien arvoja ylläpidetään muistissa tehokkaasti jatkokäyttöä varten. Muistia saadaan vapautettua poistamalla tarpeettomia säilytettyjä muuttujien arvoja.

Tietokoneohjelman testausta ja sen toiminnan analysointia tehdään monella tasolla. Algoritmeja testataan erikseen suhteessa ongelman ratkaisuun, ja tietokoneohjelmaa testataan sekä osina että kokonaisuutena. Lisäksi testausta tehdään esimerkiksi ei-toiminnallisuuden, toiminnallisuuden ja käytettävyyden tasolla. Testaukseen liittyy erityisesti toiminnallisuuden osalta virheiden etsiminen ja korjaaminen. Testauksen lisäksi dokumentointi on tärkeää, ja sitäkin tehdään monella tasolla. Ohjelmointikoodia selitetään kommentein, luokkien toimintaa luokkakuvauksin, rajapintoja dokumentoidaan rajapintakuvauksin ja testauksen kulkua sekä löydettyjä virheitä dokumentoidaan korjauksia varten. Dokumentoinnin muoto yritetään saada noudattamaan yleisiä periaatteita ja käytäntöjä, jotta dokumentointi olisi helposti lukijan ymmärrettävissä. Järkevän ja kattavan dokumentoinnin avulla ohjelmointikoodin muokkaaminen ja ylläpitäminen on myös turvallisempaa.

Tietokone suorittaa ihmisen luomaa ohjelmointikoodia niin kuin se on kirjoitettu. Tietokone ei ajattele, onko algoritmi järkevä ongelman kannalta. Tehtävän ratkaisemiseen sopivan algoritmin laatiminen ja kirjoittaminen tietokoneohjelmaksi jää ihmisen vastuulle. Tietokone ei siis ajattele niin kuin ihminen. Tietokone ei esimerkiksi algoritmia suorittaessaan sovella toimintaansa tilanteen mukaan, vaan jos esimerkiksi erityistapausta ei ole ohjelmointikoodissa huomioitu, tietokone joko toimii virheellisesti tai antaa virheilmoituksen.

Tietokoneen suorituskapasiteetti on myös rajallinen riippuen laskentatehosta, muistin ja tallennustilan määrästä. Isommissa ohjelmistoprojekteissa rajallinen suorituskyky vaatii ohjelmointikoodin, käytetyn ohjelmointikielen tai kääntäjän optimointia, lisäresurssin hankkimista kapasiteettiin tai toiminnan rajoittamista. Algoritmien kehittäminen ja optimoiminen ovat tärkeitä taitoja, vaikka tietokoneiden suorituskyky onkin kasvanut selvästi. Vaikka resurssit olisivat rajattomat, algoritmien tulee tuottaa ratkaisu haluttuun ongelmaan. Toisaalta suorituskyvyn kasvaessa myös ongelmat monimutkaistuvat ja monimutkaisiinkin ongelmiin on löydettävä ratkaisu resursseja tehokkaasti hyödyntäen. Ohjelmoin-

ti onkin prosessi, johon sisältyy huomattava määrä muitakin taitoja kuin koodaus.

Linnin (1985, 15) mukaan tietokoneohjelmointi vaikuttaa ihanteelliselta ongelmanratkaisun edistämiseksi. Ohjelmoinnin opettelu voi olla aluksi kaiken ikäisillä erittäin haastavaa. Jo 1960-luvun alkupuolelta on yritetty kehittää ympäristöjä ja ohjelmointikieliä, joiden avulla ohjelmointia kykenisi opettelemaan suurempi joukko ihmisiä (Kelleher & Pausch, 2005, 83).

3.1 Ohjelmointi opetuksessa ja CT:n kehittämisessä

Soare (1996b) kuvaa, miten tietokoneen termistö on muuttunut. Vielä vuoteen 1946 saakka kaikki Turingin ohella käyttivät ”computer”-termiä ihmisestä, joka käytti apuvälineitä kuten kynää ja paperia laskemisen apuna. Terminologisesti Soare päätyi käyttämään ”computor”-termiä mekaanisesti laskevasta ihmisestä ja ”computer”-termiä koneesta, kuten Turingin koneesta tai nykymaailmassa tuntemistamme tietokoneista. (Soare, 1996b, 291–292.)

Yliopistoissa tietojenkäsittelyn ja tietotekniikan opetus alkoi 1960-luvulla, ja joitakin peruskursseja tuli tarjolle jo 1960- ja 1970-lukujen vaihteessa (Saarikoski, 2011, 2). Aluksi ATK:ta ei pidetty peruskoulun oppiaineena, vaikka asiantuntijat tätä jo alkuvaiheessa suosittelevatkin (Saarikoski, 2011, 4). ATK:n hyödyntäminen osaksi koulujärjestelmää alkoi varsinaisesti 1980-luvulla (Saarikoski, 2011, 3–4). Tällöin virallisissa raporteissa ja opetussuunnitelmassa nostettiin tietotekniikka vahvasti esiin ja koulut pakotettiin hankkimaan tietokoneita ja muuta laitteistoa. ATK oppiaineena aloitettiin ensin lukioissa vuonna 1982. Yläkouluissa ja lukioissa tämä otettiin opetussuunnitelmaan vuosina 1987-1988 ensin vapaaehtoisena aineena. Käyttöönotossa oli kuitenkin ongelmia. Opettajia koulutettiin hätäisesti (Saarikoski, 2011, 4; Tuomi ym., 2018), kurssimateriaalia oli niukasti (Saarikoski, 2011, 6), ohjelmistotuki oli heikkoa (Saarikoski, 2011, 4; Tuomi ym., 2018) ja BASIC-ohjelmoinnilla oli hyvin suuri rooli opetussuunnitelmassa. (Tuomi ym., 2018, 426,428.)

Vuonna 1994 ATK alkoi menettää paikkansa itsenäisenä aineena. (Tuomi ym., 2018, 428.) Ohjelmointi säilyi opetussuunnitelmassa 1990-luvulle, jonka jälkeen pääpaino alkoi siirtyä tietokoneavusteisiin opetusmenetelmiin ja ohjelmistojen käyttöön (García-Peñalvo ym., 2016, 8; Tuomi ym., 2018, 6). 1990-lu-

vulla myös tietokoneiden käytettävyys parani selkeästi, kun toimisto-ohjelmat, grafiikat ja viihdekäyttö (musiikki, pelit) yleistyivät (Saarikoski, 2011, 7).

Digitaalinen lukutaito ja tietotekniikka eroavat kuitenkin selkeästi toisistaan. Lukutaidossa opitaan perustaidot ja käyttämään laitteita, sovelluksia sekä internetiä, kun taas tietotekniikalla voidaan ymmärtää taustalla oleva moninainen tiede, sekä tekniikan luominen ja kehittäminen. Lisäksi tietotekniikkaan kuuluvat keskeisesti esimerkiksi algoritmit ja algoritminen ajattelu. (European Commission, 2014, 53.) Ohjelmointia käytetään tietokoneiden algoritmien toteuttamisessa ja ohjelmointi on tietotekniikassa keskeinen toiminta, mutta vain työkalu (Tucker ym., 2006, 2). Tietotekniikan pitäisikin olla pakollinen osa opetusta digitaalisen lukutaidon ohella (Informatics Europe & ACM Europe Working Groupon Informatics Education, 2013, 18).

Opettamisen avuksi on tullut välineitä myös opettajien perehdytykseen. Esimerkiksi Koodiaapinen on suomalaisten opettajien ja opetustutkijoiden aloite, joka auttaa opettajia oppimaan ohjelmointitaitoja. (Tuomi ym., 2018, 429.) Ohjelmointi koulussa ei ole vain oikean ohjelmiston valitsemista vaan myös opetettavien taitojen ja käytettävien pedagogisten ja motivoivien mallien valitsemista (Repenning ym., 2010, 266).

1960-luvun lopulla Papert oli kehittämässä ensimmäistä lapsille tarkoitettua ohjelmointikieltä LOGO:a (Stager, 2016, 308). LOGO-ympäristössä kilpikonnalla on tietty sijainti ja suunta. Erilaisilla käskyillä se saadaan liikkumaan ja piirtämään liikkeensä. (Papert, 1985, 67–68.) Tietokoneen avulla suoritetussa opetuksessa, jossa tehtävät ovat valmiina, tietokone määrittelee oppilaan toimintaa. LOGO-ympäristössä oppilas taas ohjelmoi tietokonetta ja tarkastelee omaa ajatteluaan. (Papert, 1985, 28.)

> ”Kun – – lapsi oppii ohjelmoimaan, oppimisprosessi muuttuu. Siitä tulee aktiivisempi ja itseohjautuva. Tietoa hankitaan nimenomaan itse tunnistettuihin tarkoituksiin.” (Papert, 1985, 29.)

LOGO teki ohjelmoinnista ja monista matemaattisista aihealueista, kuten aritmetiikasta, kulmista ja geometriasta, helpommin ymmärrettävää. Lisäksi ohjelmointikielen avulla matematiikasta tuli pelillistä ja luovaa. (Stager, 2016, 308.) LOGO:n kilpikonnageometrian lisäksi voidaan käyttää muitakin visuaalisia ohjelmointitapoja. Lyen ja Kohin (2014, 53) mukaan visuaaliset ohjelmointikielet

sopivat K12-ikäisten opettamiseen, koska ne edistävät algoritmisen ajattelun oppimista ilman, että kielen syntaksiin on kiinnitettävä huomiota. Visuaalisissa ohjelmointiympäristöissä (esim. Scratch) ohjelmointikoodin osat ovat yleensä valmiina raahattavina rakenteina. Visualisaatio tekee myös testaamisesta ja virheiden etsimisestä ja korjaamisesta vähemmän kuormittavaa. (Lye & Koh, 2014, 53.)

Wolz ym. (2011) tekivät 7-8-luokkalaisille suunnatun tutkimuksen journalistisesta prosessista, jossa algoritminen ajattelu oli mukana. Tulokset ensimmäisestä kahdesta vuodesta osoittivat opettajien ja oppilaiden itseluottamuksen lisääntyneen algoritmisessa ajattelussa ja laskennallisten taitojen parantuneen. Suurin osa 7- ja 8-ryhmäläisistä piti Scratchiä hauskana, mutta 7-luokkalaiset pitivät Scratchia selkeästi hauskempana kuin 8-luokkalaiset (7-luokka 89 % ja 8-luokka 65 %). Oppilaiden kyselytulokset paljastavat, että matematiikan osaamista ei pidetä välttämättömänä Scratch-ohjelmoinnissa. Parasta Scratchin kanssa työskentelyssä olivat Scratchin oppimis- ja käyttöprosessi sekä uuden oppiminen ja luova prosessi. Huonoimmiksi asioiksi koettiin ongelmat ajan kanssa, Scratchin työskentelyn vaikeus tai sekavuus ja Scratchin oppimis- ja käyttöprosessi. (Wolz ym., 2011, 2, 17–18.)

Tutkimuksissa on käytetty menestyksekkäästi visuaalisia ohjelmointiympäristöjä ohjelmoinnin perusteiden opettamiseen. Aiemmin Carnegie Mellonin yliopistossa oli käytetty ainoastaan Javaa ohjelmoinnin alkeiden opettamiseen. Tutkimuskohortille (vuosi 2009) otettiin mukaan Alice 3 ennen Java-ohjelmointiin siirtymistä. Tuloksiin saatiin vähintään 10 % parannus, kun taustalla oli Alice 3 - Java tyyppinen lähestymistapa. Erityisesti kontrollirakenteiden (ehtolauseet ja iteraatiot) hallinta parani. Oppilaat kykenivät siis siirtämään Alicen avulla opittuja taitoja Java-ohjelmointiin. Vuonna 2010 saatiin toistettua tulos eri kohortille. (Dann, Cosgrove, Slater, Culyba, & Cooper, 2012, 144–145.)

Syntaksittomat ohjelmointiympäristöt vetoavat erityisesti aloittelijoihin, koska tällöin vältetään haasteet syntaksin virheiden kanssa. Oppilaat saavat myös välittömästi tekemisestään palautetta usein visuaalisesti. Myös erilaiset hahmot ja tapahtumapaikat auttavat esimerkiksi pelien tai tarinoiden esittämisessä. (Mannila ym., 2014, 17.)

Moskalin, Lurien ja Cooperin tutkimuksessa tietojenkäsittelytieteen ensimmäisen vuoden pääaineopiskelijoille tarjottiin Alice-kurssi johdatuskurssin CS1-

kurssin lisäksi. Tutkijat arvioivat olio-ohjelmointikielissä (esim. Java, C++) olevan liikaa ydinasioiden ulkopuolista hallittavaa sisältöä suhteessa käytettävään aikaan. Opiskelijat luokiteltiin ohjelmointi- tai tietojenkäsittelykokemuksen ja matemaattisten taitojen suhteen eri riskiryhmiin niin, että matematiikassa heikommin suoriutuvat ja vähän ohjelmointikokemusta omaavat luokiteltiin korkeampaan riskiin. Keskikorkeaan ja korkeaan riskiin kuuluvat jaettiin kahteen ryhmään, joista toinen osallistui Alice-kurssille. Alice-ryhmään kuuluvat suoriutuivat paremmin CS1-opinnoistaan kuin kontrolliryhmän opiskelijat, erityisesti korkean riskin alaryhmä. Lisäksi Alice-ryhmään kuuluvat pysyivät muita ryhmiä paremmin opinnoissa ja kyselyiden perusteella kaikki asennearvot paranivat. Kontrolliryhmässä luovuuden arvossa oli selkeä lasku CS1-kurssin jälkeen toisin kuin Alice-ryhmässä. (Moskal, Lurie, & Cooper, 2004, 75–79.)

Kuviossa Virhe: Viitteen lähdettä ei löydy on esitetty yleisiä ohjelmointirakenteita. Näihin kuuluvat sijoitukset, ohjausrakenteet (for-silmukka ja ehtorakenne) sekä muuttujan ja merkkijonon tulostus Scratchin, JavaScriptin ja Pythonin avulla. Erilaisista syntakseista huolimatta voidaan havaita yhtäläisyyksiä koodin rakenteissa. Scratch-ohjelmoinnin avulla ohjausrakenteiden opettelu onnistuu melko autenttisesti ja JavaScript tai Python-ohjelmointiin siirtymisen kynnys pienenee. Kaikki kolme kieltä sopivat hyvin opetukseen, erityisesti Python ja JavaScript yläkouluun.

Scratch

JavaScript

```
var summa = 0;
var pii = Math.PI;
for (var x = 1; x <= 10; x++) {
    summa += pii;
}
if (Math.sqrt(summa) > 8) {
    document.write(summa);
} else {
    document.write("Summan neliöjuuri ei ole yli 8");
}
```

Python

```
import math

summa = 0
pii = math.pi
for x in range(10):
    summa += pii
if (math.sqrt(summa) > 8):
    print(summa)
else:
    print("Summan neliöjuuri ei ole yli 8")
```

KUVIO 1. Koodirakenteiden vertailua Scratch-, JavaScript- ja Python-kielillä.

Haastavana piirteenä ohjelmointikielissä on, että pohjakielenä on usein englanti. Tämä pitää ottaa huomioon myös opetuksessa. Alemmilla luokilla esimerkiksi Scratchin koodipalat on käännetty suomeksi, jolloin englannin kielen osaaminen ei muodostu rajoitteeksi.

Hambruschin ym. (2009) mukaan opetuksellisessa käytössä olevan ohjelmointikielen pitäisi heti mahdollistaa keskittyminen laskennallisiin periaatteisiin, opiskelijoiden tulisi pystyä kirjoittamaan tarkoituksenmukaisia ohjelmia lyhyessä ajassa ilman keskittymistä kielen tarpeettomiin yksityiskohtiin ja saatavana pitäisi olla laajoja ohjelmointikielen kirjastoja, joita tieteellinen yhteisö käyttää. Python täyttää Hambruschin ym. mielestä ohjelmointikielen odotukset ja on erinomainen ensimmäiseksi ohjelmointikieleksi. Lisäksi Python on hyvä vuorovaikutteisen ympäristön vuoksi. Python mahdollistaa myös suurempien tietomäärien ja ongelmien käsittelyn. Se antaa myös mahdollisuuden opetella nykyaikaisia ohjelmointikäsitteitä, käyttö on vuorovaikutteista, erilaisten rakenteiden kokeilu helppoa ja se antaa opiskelijoille välitöntä palautetta. (Hambrusch ym., 2009, 184–187.)

Ohjelmointi vaatii useita haastavia taitoja. Seppälä (2012, 4) esittää väitöskirjassaan ohjelmoijan taidoiksi ohjelmointikielen kieliopin ja kuvitteellisen suorittavan koneen ymmärtämisen, yleiset käsitteet ja algoritmit sekä ohjelmointikielen käytön ja soveltamisen. Lisäksi ohjelmoijalta vaaditaan erilaisten tietorakenteiden käyttöä, dokumentointia, testausta, ohjelmointityökalujen ja kirjastojen osaavaa hyödyntämistä sekä virheenetsintää ja -korjausta. (Seppälä, 2012, 4.) Ohjelmoijan on myös hallittava ala, jolle hän tekee ohjelmointityötä, ja osattava käyttää tehokkaasti ohjelmointiin ja siihen liittyvään työhön tarvittavia laitteita ja ohjelmia.

Selby on koonnut eri lähteistä oppimisen haasteita. Näitä ovat koodin seuraaminen ja koneen mallien ymmärtäminen. (Selby, 2012, 74, 76.) Vaikka ohjelmointi on hyvä harjoittelumuoto algoritmiseen ajatteluun, se vaatii rutiinia (Loidl, Mühlbacher, & Schauer, 2005, 104). Repenningin (2006, 169) havainnon mukaan ohjelmoinnin opettelu ei motivoi kaikkia lapsia, mutta pelien tekeminen ohjelmoinnin avulla kiinnostaa lapsia. Pelien teon ohessa voidaan siis opettaa ja oppia ohjelmointia.

Ohjelmoinnin opettelun tiedetään olevan kognitiivisesti haastavaa erityisesti alkuvaiheessa (Lau & Yuen, 2009, 697). Kokonaisuudessaan ohjelmointia

pidetään tietojenkäsittelytieteen haastavimpana alueena (Sentance & Csizmadia, 2017, 471–472). Van Gorp ja Grissom (2001) esittelevät tapoja, joilla osittain ryhmämuotoisesti voidaan opettaa ohjelmoinnin perusteita. Olemassa olevan koodin läpikäynnissä (”code walkthrough”) oppilaat koodia tarkastelemalla ennustavat tuloksen (engl. output). Tämän avulla oppilaat oppivat ymmärtämään paremmin koodin suoritusjärjestystä. Samalla ryhmissä koodi voidaan helposti jakaa vastuualueisiin. Koodin kirjoittamisessa ryhmä ratkaisee pienen ongelman kirjoittamalla yhdessä koodia tai ryhmän yksilöt ovat muodostaneet ongelmasta ensin omat ratkaisunsa, jotka puretaan osiin ja muodostetaan niitä vertailemalla ryhmän yhteinen ratkaisu. Koodin täydentämisessä (engl. scaffolding) joko annetaan valmis koodi, johon pareittain lisätään kommentit, tai päinvastoin annetaan algoritmia kuvaavat kommentit, johon tulee tuottaa sopiva koodi. Tavoitteena myöhemmin on myös yleistää koodi ja miettiä tehokkaita ratkaisuja. Virheiden etsimisessä ja korjaamisessa oppilaat etsivät valmiista virheellisestä koodista syntaktisia ja loogisia virheitä. (Van Gorp & Grissom, 2001, 249–250.) Toisaalta ohjelmointi voi muokata ihmisten ajattelumaailmaa rakentavasti:

> ”Hyväksi ohjelmoijaksi oppiminen merkitsee kehittymistä taitavaksi virheiden, so. ohjelman toimimista estävien osien, etsinnässä ja poistamisessa. Ohjelmasta ei pidäkään kysyä, onko se oikein tai väärin, vaan onko se korjattavissa. [– –] Jos tietokone vaikuttaisi tällä tavoin muuttavasti mustavalkoiseen käsitykseemme menestymisestä ja epäonnistumisesta, se olisi esimerkki tietokoneen käyttämisestä ”ajatteluvälineenä”.” (Papert, 1985, 32)

Myös Papertin (1985, 43) mukaan ohjelmointitaitoon vaaditaan ohjelmointikielen oppiminen. Jo 1995-1996 tehdyssä kolmen kuuden tunnin pituisen jakson Lego/logo-projektissa 5.-6.-luokkalaisille havaittiin ohjelmoinnin ongelmat, luovuuden ja omaperäisyyden mahdollisuudet, ongelmanratkaisutaitojen tärkeys, ryhmätyöskentelyn vuorovaikutushyödyt sekä matemaattisten ja teknisten aineiden yhteensopivuus. (Järvinen, 1998, 196–210.)

Kaarakaisen, Kivisen ja Tervahartialan (2013) käsittelemässä Turun yliopiston Koulutussosiologian tutkimuskeskuksen ReadIt-tutkimuksen (Kivinen & Kaarakainen, 2012) aineiston osassa yhdeksännen luokan oppilaita (N=442) vastasi kyselyyn, jossa kartoitettiin muun muassa erilaisia nuorten vapaa-ajan käyttäjäryhmiä, arviota IT-osaamisesta ja teknologiataitojen oppimismyönteisyydestä. Tutkimuksessa havaittiin nuorten käyttävän tietokonetta myös opiskelua

ja työelämää kehittäviin taitoihin vapaa-ajallaan. Taidoiksi esitettiin esimerkiksi tekstien tuottaminen, ohjelmointi ja tiedonhaku internetistä. (M.-T. Kaarakainen ym., 2013.)

Lister ym. (2004) raportoivat ITiCSE 2004 työryhmän tuloksista. Yleinen selitys oppilaiden heikoista kyvyistä ovat oppilaiden ongelmanratkaisutaidon ongelmat. Työryhmä kuitenkin osoitti, että oppilailta puuttuu tietoa ja taitoja, jotka ovat ongelmanratkaisun edellytyksiä. Nämä vaikuttavat enemmän oppilaiden kykyyn lukea koodia kuin kirjoittaa sitä, eikä heillä ollut taitoa analysoida koodia. (Lister ym., 2004, 143.) Koodin seuraamistaidolla ja kirjoittamistaidolla on havaittu vahva korrelaatio erityisesti silmukkarakenteisiin liittyen. Lisäksi on havaittu yhteys koodin selittämis- ja kirjoittamistaitojen välillä. (Lopez, Whalley, Robbins, & Lister, 2008, 109.)

Pariohjelmoinnilla on havaittu olevan myönteisiä vaikutuksia esimerkiksi ohjelmointitehtävistä suoriutumiseen (McDowell, Werner, Bullock, & Fernald, 2003, 603–604, 607). Sen sijaan merkitseviä sukupuolten välisiä eroja ei löytynyt keskimääräisistä ohjelmointipisteistä, ja loppukokeessa miehien paremmat tulokset arvioitiin matematiikan SAT-pisteistä johtuviksi (McDowell ym., 2003, 604–605). Miehillä oli kuitenkin merkitsevällä erolla parempi itseluottamus ratkaisuihin kuin naisilla, samoin pariohjelmoivilla oli parempi itseluottamus kuin yksin tekevillä (McDowell ym., 2003, 605). Vastaavasti pariohjelmoijat ja miehet olivat tyytyväisempiä (McDowell ym., 2003, 605–606). Tutkijat arvioivat pariohjelmoinnin olevan menetelmä, jolla naiset voisivat kiinnostua ja edistyä paremmin tietojenkäsittelytieteissä (McDowell ym., 2003, 607). Myös Lau ja Yuen ehdottavat pariohjelmoinnin lisäämistä eri oppimistyyleillä oppivien ohjelmointitaitojen parantamiseksi (2009, 704–705).

Linn kuvaa ohjelmointikurssien kognitiivisten kykyjen ketjun. Ketju sisältää kielen ominaispiirteiden ymmärryksen, suunnittelutaidot ja ongelmanratkaisutaidot. Kielen ominaispiirteiden ymmärryksellä hahmotetaan kielen peruselementit ja toiminnot, ja oppilas kykenee tekemään pieniä muutoksia koodiin. Suunnittelutaidot sisältävät mallit ja menettelytapataidot, ja niiden avulla yhdistetään kielen toimintoja ongelmien ratkaisuun. Suunnittelutaitojen avulla oppilaat myös kykenevät ratkaisemaan ongelman käyttäen valmiita aikaisempia malleja ja näin ohjelmoimaan tehokkaammin sekä testaamaan ja korjaamaan suunnitelmiaan. Ongelmanratkaisutaidoissa opitaan yleistämistä eri järjestelmiin ja muodollisiin

järjestelmiin, jolloin taidot ovat siirrettävissä uuteen järjestelmään. Ongelmanratkaisutaitojen testaaminen voi vaatia siirtymistä uuteen järjestelmään, johon aikaisempia yleisiä malleja tulisi osata soveltaa. (Linn, 1985, 15–16, 25.)

Laskennan opetuksen strategioita ja haasteita selvittävässä tutkimuksessa suurin osa vastaajista oli ensimmäisen tai toisen asteen opettajia. Strategioita oli useita, ja niihin sisältyi esimerkiksi ohjelmointia ja algoritmista ajattelua, sekä näiden lähisisältöjä. Opettajien haasteina olivat aineenhallinta, eriyttäminen, ajan puute, lähestymistavat opetettaviin aiheisiin, tuen puute, arviointi ja ”kuivien” aiheiden opettaminen. Oppilaisiin liittyviä haasteita olivat aiheiden ymmärtäminen, ongelmanratkaisutaidot, sinnikkyys, motivaatio, matemaattiset kyvyt ja lukutaidot. Teknisiin haasteisiin liittyivät verkot ja ohjelmien asentaminen. (Sentance & Csizmadia, 2017, 478–480, 493.)

Useimmat opetusstrategiat voidaan luokitella oppimiseen ilman tietokonetta, yhteistyöhön, algoritmiseen ajatteluun, oppimisen kontekstualisointiin ja ohjelmointitehtävien rakenteisiin. Ohjelmointiin liittyy vahvasti virheistä oppiminen, sinnikkyys ja vastoinkäymisten hyväksyminen. Virheiden etsiminen, korjaaminen ja virheistä oppiminen ovatkin tärkeimpiä keinoja edistyä ohjelmoinnissa. Oppilaille tämä ajatusmalli on melko vieras, koska oppilaat opetetaan pyrkimään onnistumisiin virheiden sijaan. Vain pienellä osalla opettajista oli strategioita tukea tällaista ajatusmallia. (Sentance & Csizmadia, 2017, 484, 491.)

Selby on tutkinut ohjelmoinnin käyttämistä välineenä opettamaan CT:tä ja ongelmanratkaisua. Tiedot on kerätty opettajilta, opiskelijoilta ja eri alojen ammattilaisilta. Tutkimukseen osallistujat löysivät monia yhteisiä piirteitä ongelmanratkaisutaidoista, ohjelmointitaidoista ja CT-taidoista sekä niiden opettamisesta. (Selby, 2012, 74–76.) Selby on myös laatinut mallin, jossa on esitetty hierarkisesti monimutkaisuuden mukaan muutamia keskinäisiä suhteita CT:n taidoista, ohjelmointitaidoista ja Bloomin taksonomiasta (Bloom, Engelhardt, Furst, Hill, & Krathwohl, 1956). Bloomin taksonomian ja ohjelmoinnin opetuksen hierarkiat vastaavat toisiaan (esim. ongelman hajottaminen ongelmanratkaisun ja ohjelmoinnin opetuksen alkuvaiheessa), mutta järjestys on käänteinen suhteessa CT:n osa-alueiden haastavuuteen.

Sysło ja Kwiatkowska (2008, 10) käyttävät tietojenkäsittelyoppia opettaessaan algoritmista ongelmanratkaisumenetelmää, johon kuuluu kuusi vaihetta. Vaiheiden avulla saadaan ratkaisu, joka on ymmärrettävä, oikeellinen ja teho-

kas. Kuusi vaihetta ovat ongelmatilanne (ongelman analysointi, rajoitteet ja ymmärtäminen), määrittely (syötteet, tulosteet ja niiden suhteet), suunnittelu (ohjelman ja algoritmin suunnittelu), koodaus (luodaan suunnitelman mukainen ratkaisu ja arvio), testaus (systemaattinen ratkaisun testaaminen) ja esittäminen (kirjoitetaan dokumentaatio, käyttöohje ja esitys). (Sysło & Kwiatkowska, 2008, 10.) Edellä mainittu malli on todella lähellä ohjelmointia, mutta mallin vaiheissa on myös algoritmista ajattelua.

Useimpien ohjelmointikurssien tarkoituksena on opettaa ongelmanratkaisua ja päättelyä yhtä lailla kuin ohjelmointia. Tietokoneen ohjelmointi on ongelmanratkaisun muoto, ja ainakin oppilaat, jotka oppivat ohjelmoimaan, oppivat ongelmanratkaisua. (Linn, 1985, 14.) Ohjelmoijien pitää hajottaa monimutkaiset ongelmat osaongelmiksi, ja sitten luoda kokoelma käskyjä vaihe vaiheelta osaongelmien ratkaisemiseksi (Linn, 1985, 14–15). Lisäksi ongelman ratkaisua testatessa saadaan palautetta ratkaisun tehokkuudesta, ja tämän tiedon avulla ratkaisua voidaan muuttaa (Linn, 1985, 14–15).

Vaikka tietotekniikkaa integroidaan kaikkiin oppiaineisiin, ohjelmointia on pääasiassa vain matematiikkaan sisällytettynä. Ohjelmointia tulisi Mannilan (2014) mukaan integroida myös muihin aineisiin. Ohjelmointia tulisi opettaa luokka-asteesta riippuen eri tavoin: luokilla 1-2 ohjelmointityyppistä toisten oppilaiden ohjeistusta, luokilla 3-6 graafiset ohjelmointiympäristöt ja luokilla 7-9 ohjelmointikielet. (Mannila ym., 2014, 5.)

Mannila (2014) esittää kattavasti, miten ohjelmointi tarjoaa työvälineitä algoritmisen ajattelun eri osa-alueiden opettelemiseen. Tiedonkeruu onnistuu suodattamalla tietoa suuresta tekstimäärästä, hakemalla tietoa verkkosivustoilta tai tietokannoista tai keräämällä oman aineistonsa erilaisten sensorien välityksellä. Tiedon analysointia voi toteuttaa luomalla malleja erilaisilla graafisilla ohjelmointiympäristöillä tai kehittämällä ohjelmoiden omat analysointimetodit. Tiedon esittäminen onnistuu erilaisin visualisoinnein ja kuvaajin. Isoja ongelmia joudutaan hajottamaan osiin, jotta niitä saadaan ratkaistua. Algoritmeja käytetään suunnittelemaan ongelmiin vaiheittain etenevät ratkaisut. Abstrahointia käytetään ohjelmointiosioiden luomiseen ja käyttöön toistotehtävissä. Simulaatioita saa luotua graafisissa ohjelmointiympäristöissä arkisistakin toiminnoista. Automaatiolla luodaan lyhyitä ohjelmia yksinkertaisiin ja usein esiintyviin tehtä-

viin. Rinnakkaisuuden avulla useampi saa työskenneltyä yhdessä saman ohjelmistokoodin parissa. (Mannila ym., 2014, 16–17.)

Suomessa ohjelmointia on edistänyt Linda Liukas teoksellaan Hello Ruby (2015), jonka alle kouluikäisille suunnatussa kirjassa opetellaan ja harjoitellaan algoritmista ajattelua satujen avulla. Tavoitteena on ollut lisäksi saada pääosin miesvaltaisella alalla myös tytöt tietokoneiden äärelle. Liukkaan kirjassa on tulkittavissa algoritmista ajattelua esimerkiksi ideoinnin muodossa. Lisäksi kirjassa käsitellään laajasti algoritmisen ajattelun osa-alueita ikään ja osaamistasoon sopivassa muodossa: esimerkiksi parhaan ratkaisun valitseminen useammasta ratkaisuvaihtoehdosta, aiempien ratkaisujen hyväksikäyttö ja soveltaminen, ratkaisujen suunnittelu, ongelmien hajottaminen ja vaiheittaiset järjestykset. Kirja sisältää myös muuttujia, tietotyyppejä, symboleita, tietorakenteita, toistorakenteita sekä virheiden etsintää ja korjaamista. Yhteys visuaalisiin ohjelmointiympäristöihin on luotu käyttämällä piirrettyjä koodipalikoita. Osa-alueita kuvataan paljon arkielämän esimerkkien kautta. Kirjassa käytetään myös paljon ryhmätyöskentelyä, jossa jokaisella on ryhmässä jotakin annettavaa, vaikka yksittäinen ryhmän jäsen osaisi vain vähän. (Liukas, 2015.)

Vahvaa panostusta ohjelmointiin ja koodaukseen on myös kansainvälisesti. Monet aloitteet kuitenkin keskittyvät enimmäkseen ohjelmointiin yleisten CT-taitojen sijaan. Perinteistä ohjelmointia on pidetty liian teknisenä ja tylsänä koulussa opettajalle opettaa ja oppilaiden oppia. Ohjelmointi nähdään usein pelkkänä koodauksena, vaikka ohjelmointi sisältää paljon muutakin, kuten ongelmien analysointia ja suunnittelua. Näin käsitykset ohjelmoinnista voisivat muuttua enemmän ongelmanratkaisutoiminnaksi, joka on työkaluna kaikkien CT:n osa-alueiden kehittämiseen. (Mannila ym., 2014, 4).

Osallistumisen lisäämiseksi tulisi CT:n perusteita opettaa jo kauan ennen kuin oppilaat kohtaavat ensimmäisen ohjelmointikielen. Ohjelmoinnin tulisi olla enemmän keino tietojenkäsittelytieteiden osaamisen syventämiseen eikä ensimmäinen kosketus siihen. Toisaalta tietojenkäsittelytieteiden syvempi ymmärtäminen vaatii ohjelmointiosaamista, joten näiltä osin ohjelmointiopetuskin puolustaa paikkaansa. Ohjelmointi ei kuitenkaan ole välttämätön väline opettaessa algoritmista ajattelua tai tietojenkäsittelyn perusteita ja ohjelmointiopiskelu tulisikin aloittaa opiskelijoille vasta, kun heillä on harjaantumista laskennalliseen ajatteluun. Lu ja Fletcher (2009) vertaavat analogisesti, että ohjelmointi on tietojen-

käsittelytieteille kuin kirjallisuusanalyysi on englannin kielelle keinona syvempään tieteelliseen ymmärrykseen. (Lu & Fletcher, 2009, 260–261.)

3.2 Ohjelmointisuoriutumisen ja -kokemusten yhteyksiä taustatekijöihin

Pojat käyttävät tietokonetta ohjelmointiin ja tiedonhakuun tutkimuksen mukaan merkitsevästi enemmän kuin tytöt. Kymmenesosa pojista osallistui koulun ulkopuolisille atk-kursseille tai harrastuksiin. Lisäksi tutkimukseen osallistuneet arvioivat IT-taitonsa hyviksi, vaikka intensiivikäyttäjien ja passiivisten käyttäjien välillä on suuria eroja. Myös IT-taitojen lisäoppimiseen suhtaudutaan tutkimuksen perusteella myönteisesti. Tietoteknologiataidot ovat useimmiten tutkittavien itseoppimia. Vajaa kolmasosa koki koulun opetuksen merkitykselliseksi tietoteknologian käytössä. (M.-T. Kaarakainen ym., 2013.)

Tietojenkäsittelytieteisiin ja ohjelmointiin liittyy epävarmuutta. Hjelmin (2016) pro gradu -tutkimuksessa kolmivaiheisen kyselytutkimuksen tuloksena tyttöjen käsitykset omista taidosta, asenteista, näkemyksistä ja käsityksistä tietojenkäsittelytieteistä ovat matalampia muuhun populaatioon verrattuna. (Hjelm, 2016.) Ennakkoasenteena on, että oppilasryhmät, jotka ovat aliedustettuina CS:ssä eivät ymmärrä sitä. Aliedustettujen oppilasryhmien oppilailla saattavat tuntea itsensä häkeltyneiksi ja epävarmoiksi CS:n oppimiseen. Tällaiset ennakkoasenteet tuhoavat mahdollisuudet CS:n oppimiseen heti alussa. (Margolis ym., 2012, 77.) Ohjelmoinnista eristäytyneet opiskelijat saattavat työntyä pois myös tietojenkäsittelytieteistä. Jos koulussa ei opeteta ohjelmointia, toisaalta annetaan ennakkoasenteiden mukaista viestiä ohjelmoinnin epämiellyttävyydestä. (Curzon, Peckham, ym., 2009, 202.) Aliedustettujen ryhmien tulisi myös viihtyä STEM-aineiden parissa (Wolz ym., 2011, 7).

Pioron (2004, 14, 22) tutkimuksessa tarkasteltiin C++-koodauksen loppukoetta, alussa tehtyä kyselylomaketta taidoista ja GPA-arvosanan (engl. Grade Point Average) seurantaa. Todettiin, että naisten minäpystyvyys oli heikompi kuin miesten, mutta pisteet olivat naisilla paremmat sekä ohjelmointitehtävissä että monivalinnassa, ja naiset pitivät paremmin tasonsa kahden vuoden jälkeen. Laun ja Yuenin (2009) tutkimuksessa 14-19 -vuotiaiden toisen asteen opiskelijoiden ohjelmointisuoriutumisista ja oppimistyyleistä sukupuolieroja ei löydetty.

Naiset suoriutuivat paremmin, mutta selittäväksi tekijäksi arvioitiin sukupuolen sijaan akateeminen kyvykkyys, koska tutkimuksessa akateemisesti paremmin menestyvät suoriutuivat myös ohjelmoinnista paremmin. (Lau & Yuen, 2009, 700–701, 703, 705)

Yläkoulua/keskikoulua koskevassa tutkimuksessa aiempi tietokoneen käyttökokemus oli yleisempää pojilla kuin tytöillä. Asenteissa eri sukupuolten välillä ei kuitenkaan ollut eroa. Kiinnostus tietokoneisiin liittyi ohjelmointisuoritukseen. Johdonmukaista eroa sukupuolten välillä ohjelmoinnin loppuarvioinnin suorittamisessa ei löydetty. Huomiota herättävää tutkimuksessa kuitenkin oli, että ohjelmoivista oppilaista tyttöjä oli 37 %, mutta tyttöjen osuus kyvykkäimmän prosentin joukossa oli 60 %. (Linn, 1985, 28)

Kiss (2010) on tutkinut Unkarissa ohjelmointia ja tietojenkäsittelytieteitä sukupuolinäkökulmasta. Tutkimuksessa otettiin huomioon eri koulutyypit ja kouluasteet (9-12), ja kysymysten aihepiiri ja määrä räätälöitiin vastaavan oppilaan mukaan. Lisäksi vastauksille oli kysymyksen ymmärrykseen asetettu aikaraja ja vastausaikaraja. Unkarissa tietojenkäsittelyn oppiaine on vapaaehtoista luokka-asteilla 11-12. Ohjelmointia osataan useilla luokka-asteilla vain vähän verrattuna muihin tietojenkäsittelyn aihepiireihin. Ohjelmointi alkaa vasta 11. asteella, ja pojat osaavat ohjelmointia selvästi tyttöjä paremmin. Ero oli selkeä kaikissa testin ohjelmointikysymyksissä. Suuressa osassa tietojenkäsittelyn aihepiirien tiedoissa ei ole havaittavaa eroa sukupuolten välillä. (Kiss, 2010.)

Tutkijan mukaan pojat oppivat korkeammilla opetusasteilla ohjelmointia helpommin kuin tytöt, ja tytöillä on ongelmia löytää syntaksivirheitä lähdekoodista tai ymmärtää koodia lainkaan. Opettajilla epäillään olevan liian vähän aikaa opettaa eri tietojenkäsittelytieteen aihepiirejä, ja kaksi tuntia viikossa ei riitä. Koska ero on suuri juuri ohjelmoinnissa sukupuolten välillä, tutkija epäilee, että sitä pitäisi opettaa eri tavoin tytöille ja pojille. Opettajat käyttävät Pascalia, Delphiä, C:tä tai Javaa opettaessaan ohjelmointia, ja koodi on olennaisessa osassa. Tämä metodi näyttäisi tutkijan mukaan suosivan poikia. Tutkijan mielestä tyttöjä olisi parempi opettaa eri työkaluilla, kuten Lego-Mindstormsilla, jotta syntaksivirheitä ei syntyisi niin paljon. (Kiss, 2010, 24–29.)

Tietojenkäsittelytieteissä on havaittu alalle ilmoittautuneiden vähentyneen ja havaittu myös kasvavat kuilut naisten ja vähemmistöjen osallistumisessa. Huolena on myös valmiuksien puute. (Ahamed ym., 2010, 42, 44). Myös Allan,

Barr, Brylow ja Hambrusch (2010, 390) ovat todenneet saman ongelman, joka tapahtuu samaan aikaan, kun urakehitysmahdollisuudet tietojenkäsittelytieteissä kasvavat, ja alan osaajat ovat ratkaisevan tärkeitä maailmanlaajuisesti kilpailukykyisen työvoiman rakentamisessa 2000-luvulla.

Ohjelmointikilpailuja järjestetään eri suoritustavoilla, tietokoneella tai kynäpaperitehtävinä. Monissa maissa STEM-aineet näkyvät vahvasti opetussuunnitelmissa, mutta on vain harvoja oppilasohjelmoijia, jotka osallistuvat ohjelmointikilpailuihin. Opettajien voi myös olla vaikea havaita näitä harvoja kyvykkäitä oppilaita. Heikkoina puolina teknologiaa vaativissa kilpailuissa ovat mahdolliset teknologian ja ohjelmoinnin aiheuttamat haasteet. Jos oppilaat eivät tuota toimivaa ohjelmaa, tuloksena ovat alhaiset pisteet tai nollapisteet, mikä lannistaa kokemattomia oppilaita osallistumaan kilpailuun sekä lannistaa kilpailuihin osallistuvia. Kynä-paperitehtävinä järjestetyt kilpailut ovatkin lisääntyneet viime vuosina. (Burton, 2010, 3–4.)

Tietojenkäsittelytieteeseen liittyviä sukupuolieroja tarkasteltiin 358 kreikkalaisella 17-18-vuotiaalla lukion opiskelijalla (177 poikaa ja 181 tyttöä). Molemmilla sukupuolilla havaittiin virheellisiä käsityksiä. Opiskelijoista 16,5 % oli päättänyt pyrkiä CS-opintoihin, pojat merkitsevästi enemmän kuin tytöt. CS-opinnot valittiin hyvien työllisyysnäkymien ja henkilökohtaisen kiinnostuksen vuoksi, eikä suurimmalla osalla ollut erityistä henkilöä, joka olisi kannustanut CS-opintoihin. CS-opintoja ei valittu muista ammattisuunnitelmista johtuen (vaikka CS:n ja IT:n merkitys nyky-yhteiskunnassa tunnustettiin) tai koska haluttiin työskennellä mieluummin ihmisten kuin koneiden parissa tai ohjelmoinnin vaikeudesta ja lapsuudessa tietokoneisiin tutustumisen vähyydestä johtuen. (Papastergiou, 2008, 594–600.)

Tyttöjen prosenttiosuus on merkitsevästi korkeampi kuin pojilla kokemuksissa, että kotona tai koulussa ei ole aikaisemmin ollut mahdollisuuksia tutustua tietokoneisiin. Suuri osa opiskelijoista kertoi CS:n sisältävän ohjelmointia ja algoritmeja. Tytöt määrittelivät CS:n perinteisemmin (laitteisto, algoritmit ja ohjelmointi) kuin pojat, ja pojat esittävät CS:n inhimillisempänä ja sovelluslähtöisempänä kuin tytöt. Myös opettajan sukupuolella voi olla merkitystä, sillä miespuolinen opettaja saattaa tukea alan kokemista maskuliiniseksi ja naispuolinen opettaja taas lisätä tyttöjen tietokoneminäpystyvyyttä. Pojat käyttävät tyttöjä enemmän ja useammin tietokonetta kotona. Tutkija arvioi, että kotona tapahtuva tieto-

koneen käyttö lisää tietokoneminäpystyvyyttä, ja näin ollen poikien tietokoneminäpystyvyys voi olla suurempi. Tietokoneminäpystyvyyden todettiin liittyvän opiskelijan aikeeseen opiskella CS:ää. Opiskelijoilla, joilla oli aikomus opiskella CS:ää, oli korkeampi tietokoneminäpystyvyys. (Papastergiou, 2008, 600–604.)

IT-ammattilaisuutta luonnehdittiin ohjelmointia vaativaksi, vaikeaksi ja luovaksi. Pojilla oli IT-ammattilaisuudesta positiivisemmat käsitykset kuin tytöillä. Tyttöjen käsityksissä olivat vaikeus ja ohjelmointisuuntautuneisuus. Opiskelijat eivät kokonaisuutena usko, että CS ja IT-ammattilaisuus soveltuvat paremmin miehille kuin naisille, mutta pojat näkevät ne kuitenkin enemmän maskuliinisina kenttinä kuin tytöt. Suuri osa on ottanut tutkimuksen aikaan CS-kursseja koulussa, mutta pojat tyttöjä enemmän. Poikia oli opetettu myös vuosissa mitattuna tyttöjä enemmän. Opiskelijoista 19,8 % totesi tietävänsä jonkin ohjelmointikielen, mutta poikien osuus (24,9 % pojista) oli huomattavasti suurempi kuin tytöillä (14,9 % tytöistä). Tyttöjen CS-tason keskiarvo oli merkitsevästi poikien tasoa korkeampi. (Papastergiou, 2008, 601–602.)

4 KIELENTÄMINEN

Tässä tutkielmassa kielentämisen tarkastelu keskittyy erityisesti matematiikkaan sekä opetus- ja oppimisnäkökulmaan. Joutsenlahden (esim. 2003, 2009) mukaan matematiikan kielentämisellä (engl. languaging) tarkoitetaan kielen avulla tapahtuvaa matemaattisen ajattelun ilmaisemista. Kielentäminen voi olla suullista tai kirjallista (Joutsenlahti, 2009; Joutsenlahti, Kulju, & Tuomi, 2013). Kielentämistä on käytetty esimerkiksi kielten, filosofian, psykoterapian ja matematiikan aloilla (Joutsenlahti & Kulju, 2015, 59; Lankiewicz & Wąsikiewicz-Firlej, 2014).

Kielentämistä on tarkasteltu myös useilla eri koulutustasoilla. Kielentämistä on käytetty ja tutkittu alakoululaisilla, esimerkiksi ensimmäisen luokan oppilailla monilukutaidon kehittämistä (Laitinen, Rantamäki, & Joutsenlahti, 2015) sekä millaisia merkityksiä matemaattisen symbolikielen lausekkeille annetaan luonnollisella ja kuviokielellä (Joutsenlahti ym., 2013). Lukio- ja yliopistoikäisillä on teetetty kielentämistehtäviä matematiikkaan (Sarikka, 2014), ja yliopisto-opiskelijoilla on tehty tutkimusta joukko-opin tehtävillä, miten he suoriutuvat kielentämisestä ja miten he suhtautuvat siihen (Ojanen, 2016). Ala- ja yläkouluikäisillä sekä opettajilla on myös tutkittu suhtautumista kielentämiseen, ja tutkittu sen hyödyllisyyttä oppimisessa matematiikassa (Mäcklin & Nikula, 2010).

Kielentämisen ymmärtämiseksi käsitellään matemaattista ajattelua ja ajattelua suhteessa kieleen, matematiikan ominaisuuksia kielen näkökulmasta sekä käsitteiden muodostusta. Lisäksi pohditaan lyhyesti haasteita matematiikan ja kielen oppimisessa, ja tarkastellaan kielentämisen ja matemaattisen ajattelun esiintymistä opetussuunnitelmissa. Tutkielman opetusosuuden tehtävien muodostamisen apuna on käytetty erilaisten kielentämistehtävätyyppien tarkastelua.

4.1 Matemaattinen ajattelu

Matemaattinen ajattelu (engl. mathematical thinking) voidaan määritellä usealla eri tavalla lähestymistavasta riippuen (esim. Joutsenlahti, 2003, 190; Näveri, Ahtee, Laine, Pehkonen, & Hannula, 2012, 83). Määritelmien moninaisuuden lisäksi matemaattisen ajattelun oppimisen kehittämisestä on useita erilaisia käsityksiä. Opettajan vallitseva näkökanta vaikuttaa myös hänen matematiikan opetukseensa. (Näveri ym., 2012, 83.) Joutsenlahden (2005, 103–104, 2006, 234; 2013, 108) mukaan matemaattinen ajattelu liittyy oppilaan metakognitioihin, jotka ohjaavat matemaattisen tiedon prosessointia. Prosessoinnissa tietoverkkoa järjestellään ja muokataan (Joutsenlahti, 2006, 234).

Matemaattinen tieto koostuu konseptuaalisesta tiedosta (käsitetiedosta), proseduraalisesta tiedosta (menetelmätiedosta) ja strategisesta tiedosta (Joutsenlahti, 2006, 234; Laitinen ym., 2015, 134–135). Konseptuaalinen tieto sisältää aihealueen käsitteiden ja niiden yhteyksien ymmärtämisen, soveltamisen sekä tietorakenteiden kehittymisen. Proseduraaliseen tietoon kuuluvat toimintojen ja taitojen käyttäminen, esimerkiksi laskurutiini. Nämä tiedon muodot eivät saisi jäädä opetuksessa irrallisiksi. (Laitinen ym., 2015, 134–135.) Joutsenlahden (2005, 89) mukaan strategiat ovat henkisiä operaatioita, joilla kognitiivisia prosesseja voidaan hallita. Matematiikan osalta esimerkiksi Pólyan (1971) ongelman ratkaisemisen vaiheita (ongelman ymmärtäminen, suunnitelman laatiminen ja toteutus sekä tulosten ja perustelujen tarkistaminen) voidaan pitää strategiana (Joutsenlahti, 2005, 89–91). Hyvän strategisen tiedon hallitsevalla oppilaalla esimerkiksi prosessina ongelman käsittely, toimintasuunnitelmat ja ongelmanratkaisu ovat kehittyneet kokonaisuudessaan.

Matemaattisen ja algoritmisen ajattelun välillä on nähty olevan yhteys. Sneiderin, Stephensonin, Schaferin ja Flickin (2014, 10–11) mukaan oppilaat hyödyntävät matemaattista ajattelua lähestyessään uutta tilannetta käytettävissä olevilla matemaattisilla taidoillaan. Vastaavasti oppilaat hyödyntävät algoritmista ajattelua lähestyessään uutta tilannetta, sillä he ovat tietoisia tavoista, joilla tietokoneet voivat auttaa visualisoimaan järjestelmiä ja ratkaisemaan ongelmia. Yhteiset kyvyt ovat heidän mukaansa ongelmanratkaisu, mallintaminen, datan analysoiminen ja johtopäätösten tekeminen sekä tilastotiede ja todennäköisyyslaskenta. (Sneider ym., 2014, 10–11.) Kyvyt perustuvat osittain matema-

tiikan aihealueisiin ja algoritmisen ajattelun harjoittamisen välineisiin ja tietojenkäsittelytieteisiin. Toisaalta merkittävänä yhdistävänä tekijänä voidaan nähdä algoritmien käyttö ongelmien ratkaisussa (vrt. myös Joutsenlahti, 2005, 62–76).

Myös Thiruvathukalin (2013, 4) mukaan algoritminen ajattelu sisältää osittain matemaattista ajattelua. Tarvitaan matemaattinen kypsyys ja selkeä matemaattinen ajattelu, joiden päälle voidaan rakentaa tietojenkäsittelylle erityistä matematiikkaa (Association for Computing Machinery (ACM) & IEEE Computer Society, 2013, 50). Algoritminen ajattelu ja matemaattinen ajattelu ovat kytköksissä toisiinsa, mutta ne eivät ole identtiset. Kumpaankin ajattelutapaan liitetään abstraktio, päättely ja tunnistettavat mallit. Eroiksi on puolestaan esitetty esimerkiksi algoritmisessa ajattelussa huomioon otettavat fyysiset rajoitteet (muun muassa tietokoneiden tehorajoitteet) (Wing, 2006, 35) ja toisaalta taas algoritmisessa ajattelussa ilmenevien monimutkaisten prosessien parempi esitysmuoto suhteessa matematiikan esitysmuotoihin. (Committee for the Workshops on Computational Thinking & National Research Council, 2010, 33–34.)

4.2 Matematiikka, kieli ja ajattelu

Koulussa oppilaat opiskelevat monia erityyppisiä kieliä. Silfverbergin, Portaankorva-Koiviston ja Yrjänäisen (2005, 152) mukaan matematiikan kieltä opitaan muiden kielien tapaan spontaanissa vuorovaikutuksessa – passiivisesti (lukeminen ja kuunteleminen) ja aktiivisesti (kirjoittaminen ja puhuminen). Joutsenlahden (2003, 188–189) mukaan kieleen kuuluvat puhutun ja kirjoitetun kielen lisäksi myös eleet, ilmeet, kuvat ja symbolit, ja kieli on mukana kognitiivisessa kehitysprosesseissa. Oppilaiden äidinkieli ja vieraat kielet ovat luonnollisia kieliä ja matematiikka ja ohjelmointikielet puolestaan formaaleja kieliä (Joutsenlahti & Kulju, 2015, 58). Silfverberg, Portaankorva-Koivisto ja Yrjänäinen (2005, 151) esittävät, ettei matematiikan ja luonnollisten kielten välillä ole selkeää rajaa, vaan matematiikan sanastoa esiintyy luonnollisessa kielessä ja luonnollista kieltä tarvitaan osana matematiikkaa.

Matematiikalla ja kielellä on havaittavissa monia yhtäläisyyksiä. Wakefield esittää (1999, 3–4, 2000, 272–273) yhteyksiä matematiikan ja kielten välillä, esimerkiksi: Kommunikoinnissa käytetään abstraktioita, ja ilmaisut ovat lineaarisia ja sarjamuotoisia. Symbolit ja säännöt ovat yhdenmukaisia, ja onnistuminen

vaatii näiden muistamista. Lisäksi symbolien järjestyksellä on vaikutusta merkitykseen. Tarvitaan koodikielelle muuntamisen ja koodin purkamisen taitoa. Harjoittelu ja aiempi käyttö luovat perustan tulevalle kehitykselle. Intuitio ja ”puhe ilman ajattelua” orastavat sujuvuutta. (Wakefield, 1999, 3–4, 2000, 272–273)

Matematiikan ja luonnollisten kielten välillä on havaittavissa silti joitakin selkeitä eroja. Joutsenlahden ja Kuljun (2015, 58–59) mukaan luonnolliset kielet eroavat ilmaisukyvykkyydeltään matematiikan symbolikielestä ja ohjelmointikielistä. Esimerkiksi tunteita ja asenteita voidaan ilmaista helpommin luonnollisen kielen avulla, sillä luonnollinen kieli on mukautuvampi kuin täsmällinen matematiikka (Joutsenlahti & Kulju, 2015, 58–59). Silfverberg, Portaankorva-Koivisto ja Yrjänäinen (2005, 152–153) kuvaavat matematiikan ominaisuuksiin persoonattomuuden ja ajattomuuden. Matemaattisten asioiden käsittelyn ja ilmaisun lisäämiseksi saatetaan tarvita apuvälineitä, kuten verbaalisuutta sekä symbolista ja graafista esitystä (Silfverberg ym., 2005, 151–152), mikä voi auttaa ymmärtämään matematiikkaa, joka on usein kirjoitettu hyvin kompaktiin symboliseen muotoon. Nämä piirteet voidaan tulkita poikkeaviksi suhteessa luonnolliseen kieleen.

Matematiikan kielenkaltaisuus heijastuu myös opettamiseen ja luokkahuoneessa tapahtuvaan diskurssiin. Silfverberg, Portaankorva-Koivisto ja Yrjänäinen (2005, 150) esittävät, että matematiikan opettaja on sekä matematiikan kielen opettaja että äidinkielen opettaja. Monikulttuurisuuden myötä kouluissa käytetään yhä useampia kieliä. Joutsenlahti (2006, 235–236) esittää kielen olevan opettajan ja oppijoiden välisessä kommunikaatiossa avainasemassa. Matematiikan opetuksessa on otettava huomioon esimerkiksi ratkaisujen perustelut ja täsmällisyys hyvän yhteisymmärrystä tavoittelevan sosiaalisen vuorovaikutuksen sijaan (Silfverberg ym., 2005, 153). Näin ollen matemaattisen yhteisön tapaa ratkoa ongelmia voitaisiin jopa pitää jossain määrin erilaisena kuin arkielämän tapaa.

Matematiikan opetuksessa diskurssi saattaa olla opettajajohtoista, jolloin vain harvat oppilaat voivat puhua matematiikan ongelmista ja kehittää matematiikan kieltä. Sen sijaan passiivista kielen oppimista on tarjolla runsaasti, jos opettaja käyttää paljon matematiikan kieltä (mallioppiminen) (Laitinen ym., 2015, 150; Silfverberg ym., 2005, 153–154). Passiivinen matematiikan kielen oppiminen ei riitä aktiivisen kielitaidon hankkimiseen. Oppilaiden keskinäisessä

diskurssissa (esimerkiksi yhteistoiminnallisia työtapoja käytettäessä) oppilaat puhuvat matematiikan kieltä, mutta kielen korrektius ei ole välttämättä tarpeeksi korkeatasoista, ja matematiikan kielellinen malli saattaa puuttua. Näiden diskurssimuotojen väliin jää tapa, jossa oppilaat sekä kommentoivat että arvioivat ideoita ja perusteluja, ja näin he saavat mallin sekä korrektista kielenkäytöstä että ovat aktiivisia kielen käyttäjiä. Lisäksi diskurssia esiintyy oppimateriaalin ja teknisten apuvälineiden kanssa työskenneltäessä. (Silfverberg ym., 2005, 153–154.)

Silfverbergin, Portaankorva-Koiviston ja Yrjänäisen (2005) tekemässä tutkimuksessa matemaattisten aineiden opettajaopiskelijat hahmottivat matematiikan kieleksi. Matematiikan koettiin ylittävän kielirajat. Matematiikka koettiin ajattelun kieleksi ja ensisijaisesti kirjoitetuksi kieleksi, jolla voi keskustella. Matematiikan opettamista ja oppimista ei kuitenkaan koettu samanlaiseksi kuin vieraan kielen. Opetuskielen täsmällisyyttä halutaan lisätä ja opetustapahtuman vuorovaikutusta vähentää siirryttäessä peruskoulusta lukion matematiikkaan. Matematiikan ymmärrykseen keskittyvää opetusta pidettiin tärkeänä. Sukupuolella ja matematiikan opintojen määrällä ei havaittu olevan yhteyttä näihin matematiikkakäsityksiin. (Silfverberg ym., 2005, 156–162.)

Matematiikalle ominainen kieli voidaan jakaa osiin. Joutsenlahti, Kulju ja Tuomi (2013, 110–111) esittävät matematiikan kielen koostuvan matematiikan luonnollisesta kielestä, matematiikan symbolikielestä ja matematiikan kuviokielestä. Matematiikan luonnollisella kielellä kuvataan esimerkiksi käsitteet ja annetaan merkityksiä symboleille äidinkielellä. Matematiikan symbolikieleen kuuluvat sovitut merkinnät, kaavat ja olioiden suhteet (esimerkiksi matematiikan symbolein esitetyt lausekkeet). Matematiikan kuviokieleen sisältyvät puolestaan kuvat, kuviot ja graafiset esitykset. (Joutsenlahti, 2006, 235–236; Joutsenlahti ym., 2013, 110–111.) Matematiikan kielten lisäksi tarvitaan yleistä luonnollista kieltä, jonka avulla opittava aines ja käsitteet liitetään oppilaan tietorakenteisiin ja konteksteihin (Joutsenlahti, 2006, 235–236).

Oppilas voi vaihtaa matematiikan kieltä tyypistä toiseen, ja yhdeksi oppimisen tavoitteeksi voidaankin ajatella sujuva kielimuotojen käyttö ja ymmärtäminen. Joutsenlahden, Kuljun ja Tuomen (2015, 64; 2013, 111) mukaan kielen variaatiota eli liikkumista kielten välillä kutsutaan koodinvaihdoksi. Matematiikan oppimisessa koodinvaihtoa tapahtuu siis matematiikan kielen, luonnollisen kie-

len ja kuviokielen sekä yleisen luonnollisen kielen (esimerkiksi äidinkieli) välillä. Ongelmanratkaisussa, kielentämisessä ja ratkaisua muille selvitettäessä koodinvaihto voi lisätä ymmärrettävyyttä. (Joutsenlahti & Kulju, 2015, 64; Joutsenlahti ym., 2013, 111.)

Matematiikan oppimisessa käsitteiden oppiminen on tärkeää (ks. luku 4.1 konseptuaalinen tieto), varsinkin, kun otetaan huomioon matematiikan abstrakti luonne. Sfard (2008, 111) määrittelee termin käsite seuraavasti: *”Concept is a symbol together with its uses.”* Sanaa ”symbol” on käytetty määritelmässä, koska sitä pidetään ”word”-sanaa kattavampana. (Sfard, 2008, 111.) Joutsenlahti (2003, 191–192) puolestaan esittää käsitteen muodostuvan käsitteen sisällöstä ja ilmaisusta. Opettajan tulee tuntea oppiaineensa sisältö ja ohjata oppilasta käsitejärjestelmien muodostamisessa (Laitinen ym., 2015, 133). Opetuksessa oppilas luo mentaalimallia käsitteestä aiempien tietojen, taitojen, kokemusten ja uskomusten perusteella. Oppilaan kielentäessä käsitettä hän myös prosessoi omaa käsitystään käsitteestä. Kielentämistä sisältävässä vuorovaikutuksessa opettaja ja oppilaat saavat tietoa ja erilaisia näkökulmia käsitteestä, ja voivat sen perusteella muokata omaa käsitesisältöään. Tähän liittyy myös olennaisesti kriittisyys. (Joutsenlahti, 2003, 191–193.)

Kieli ja ajattelu ovat yhteydessä toisiinsa. Vygotskin (1982, 98) mukaan ajattelua ja kieltä voidaan kuvata ympyröillä, jotka leikkaavat osittain toisiaan (ks. ajattelun ja kielen kehityksestä ja yhteydestä tarkemmin Vygotski, 1982). Sfardin (2008, 80–81) mukaan ajattelu toteutetaan yksilön toimesta ja yksin, eikä toisen ajattelua voi suoraan saavuttaa. Ajattelu onkin yksilöity versio kommunikoinnista (Sfard, 2008, 80–81). Joutsenlahden (2006, 234) mukaan oppilaan matemaattinen ajattelu voidaan havaita ainoastaan kommunikaation (esimerkiksi suullinen ja kirjallinen kieli sekä kuvat) välityksellä. Kielentäminen toimiikin apuvälineenä oppilaan ajattelun jäsentämisessä ja ajattelun välittämisessä ulkopuolisille. Kielentämisellä voidaan perustella ja reflektoida, ja lisäksi kielentämisessä ilmenevät oppilaan asenteet ja uskomukset. (Joutsenlahti, 2003, 189–192.) Samalla luodaan toimintatapaa, jossa yritetään etsiä erilaisia ratkaisuja yhden oikean ratkaisun sijaan (Laitinen ym., 2015, 143). Näin ollen esimerkiksi kielentämisen avulla virheiden tekemisen pelkoa voitaisiin mahdollisesti lieventää.

Silfverberg, Portaankorva-Koivisto ja Yrjänäinen (2005) pohtivat sopivan kielentämisen määrää. Ääripäiden (kaiken kielentäminen ja pelkkä symboliesitys) väliltä on löydettävä sopivat opettamisen esitystavat ja materiaalit, jotta matematiikan ymmärrettävyys, täsmällisyys, tiiveys ja kielestä riippumaton yksikäsitteisyys saavutetaan. Liiallisella ja laaduttomalla kielentämällä voidaan siis saada aikaan symboliesitystä hankalammin ymmärrettävä versio, tai matematiikan symbolikielen rakenteita ei opita riittävällä tasolla. (Silfverberg ym., 2005, 155–156.) Kielentämistä voidaan käyttää esimerkiksi opetuksessa, opetuksen suunnittelussa, oppimisessa ja oppilaan ajattelun ja käsitteiden oppimisen arvioinnissa (Joutsenlahti, 2003, 194, 2009, 75; Joutsenlahti & Kulju, 2015, 64). Oppilaat voivat kokea oman ajattelun kielentämisen vaikeaksi, mutta harjoittelulla kielentämistaitoja voi kehittää jo varhaisina kouluvuosina (Joutsenlahti, 2003, 188).

Matematiikan ratkaisut paperille kielentämällä passiivisemmatkin oppilaat voisivat aktivoitua matemaattisessa ajattelussa. Samalla oppilaat ja esimerkiksi opettaja voivat havaita ajattelun vahvuuksia, heikkouksia ja ongelmakohtia. Matematiikka ja ohjelmointikielet ovat molemmat formaaleja kieliä, joten voidaan olettaa, että matematiikan kielentämisen kaltaisella koodin tulkinnalla voidaan tarkastella, miten oppilaat ajattelevat ja ymmärtävät ohjelmointikoodin logiikan. Lye ja Koh (2014, 58) suosittelevat ohjelmointiprosessin ymmärtämiseksi käyttämään ääneen ajattelua (engl. think-aloud) ohjelmoidessaan.

Algoritmiselle ajattelulle on ehdotettu oman kielen (engl. computational thinking language, CTL) muodostamista perus- ja toisen asteen opetukseen. Kielessä kiteytyisivät käsitteet ja menetelmät, mutta se ei olisi ohjelmointikieli. (Lu & Fletcher, 2009, 261–262.) Groverin (2011) tutkimuksessa tehtiin viiden päivän robotiikka- ja tekniikkatyöpaja yläkoulu- ja lukioikäisiä vastaavilla oppilailla. Tutkimuksessa tarkasteltiin algoritmisen ajattelun ja CTL:n kehittymistä työpajan aikana. Tutkimuksessa aineisto oli pieni (n = 10), mutta havaittiin käsitteiden käytön runsas lisääntyminen kaikilla osa-alueilla ja algoritmisen ajattelun prosessien ja periaatteiden parempi hallinta. (Shuchi Grover, 2011.)

4.3 Matematiikan oppimiseen liittyviä tekijöitä

Matemaattiseen ajatteluun vaikuttavat useat tekijät. Joutsenlahden (2006, 233–234) mukaan oppilaan, opettajan ja oppilastoverien muodostama diskurssi, oppimateriaali, yhteiskunnalliset tekijät (esimerkiksi arvot, ympäröivä kulttuuri ja politiikka) ja uskomukset (esimerkiksi identiteetti ja matematiikkakuva) voivat muovata oppilaan matemaattista ajattelua. Joutsenlahti (2005, 51) esittää, että matemaattiseen ajatteluun vaikuttavat myös oppilaan matemaattiset kyvyt, opiskelijan ongelmanratkaisutaidot ja informaation prosessointi. Oppimisprosessiin kokonaisuudessaan taas vaikuttavat esimerkiksi oppilaan uskomukset, asenteet, tunteet, motivaatio, itsesäätely, opiskelijan käsitykset, näkemykset, itseluottamus, itsetunto, matematiikkapelko ja kunnianhimo (Joutsenlahti, 2005, 52).

Oppilaan omaa käsitystä itsestä oppijana on tärkeää tukea myönteiseen suuntaan. Laitisen, Rantamäen ja Joutsenlahden (2015, 140, 152) mukaan oppilaan käsitys matematiikkakuvasta muodostuu jo varhain. Matematiikasta keskusteleminen ja yhdessä pohtiminen kehittää matemaattisten taitojen ja ajattelun lisäksi myös vuorovaikutustaitoja ja itsetuntoa (Laitinen ym., 2015, 140, 152). Yrjönsuuren (1998, 89) mukaan itsearvostus on itsearviointia, uskooko oppilas olevansa kyvykäs ja menestyvänsä alueilla, joita hän pitää tärkeinä. Opettajien toiminta vaikuttaa oppilaan matematiikkakokemuksiin ja kokemukset taas vaikuttavat oppilaan käyttäytymiseen. Ongelmanratkaisutehtäviä ajatellen tärkeää on myös ongelmanratkaisusitkeyden kehittäminen, mikä vaikuttaa myös matematiikan oppimiseen. (Näveri ym., 2012, 84.)

Joutsenlahden pitkän matematiikan opiskelijoiden tutkimuksessa 1990-luvun loppupuolella vahvat stereotyyppiset käsitykset poikien paremmuudesta ja kiinnostuksesta eivät saa yleistä tukea. Poikia ei pidetty matematiikassa tyttöjä lahjakkaampina tai erityisen paljon matematiikasta kiinnostuneempina. Toisaalta sukupuolen mukaan tarkasteltuna tytöt eivät kokeneet haastavia matematiikan tehtävinä mieluisina, kuten pojat, ja tyttöjen minäkuva matematiikan osaajana oli heikompi kuin pojilla. (Joutsenlahti, 2002, 81–82.)

Oppilaan muodostaman käsityksen lisäksi on olemassa muitakin oppimisen haasteita. Sfard (2008, 16–27) käsittelee esimerkiksi väärinymmärryksiä (engl. misconceptions) ja oppimisvaikeuksia (engl. learning disabilities) mate-

matiikassa. Lisäksi matematiikan vaikeuksia on käsitellyt Taipale väitöskirjassaan (2010). Kognitiivisen psykologian olettamuksen mukaan sisäiset ja ulkoiset esitysmuodot ovat yhteydessä toisiinsa, joten voidaan olettaa ulkoisten matemaattisten esitysten vaikuttavan sisäisiin matemaattisiin esityksiin. Visuaaliset esitykset matematiikassa lisäävät opetusmateriaalien ja ongelmien ratkaisuiden ymmärrettävyyttä, ja visuaalisia esityksiä tulisi käyttää alempien asteiden lisäksi myös korkeammilla asteilla. (Ambrus, 1998, 75.)

4.4 Kielentäminen opetussuunnitelmissa

Sekä vuosien 2004 ja 2014 perusopetuksen opetussuunnitelmissa että vuoden 2015 lukion opetussuunnitelmassa kielentäminen on huomioitu useilla tavoilla. Opetussuunnitelmat kehottavat vuorovaikutuksen ja ilmaisun kehittämiseen esimerkiksi itsensä ilmaisemiseen ja esiintymiseen (Opetushallitus, 2004, 39–40, 2015, 16–17, 2016, 21). Opetuksen avulla pyritään tukemaan monipuolista kielenkäyttöä sekä luonnollisilla kielillä että matematiikan eri kielillä (Opetushallitus, 2016, 21).

Uusissa perusopetuksen ja lukion opetussuunnitelmissa eräs huomionarvoinen laaja-alaisen osaamisen osa-alue on monilukutaito. Monilukutaito on tiedon ja tekstien laaja-alaisen tulkinnan ja tuottamisen taito. Taidon avulla oppilas ymmärtää monimuotoisia tekstejä ja viestinnän muotoja, jotka on tuotettu eri tavoin. Monilukutaidon avulla kehitetään myös tiedonhankkimisen ja kriittisen ajattelun taitoja. Opetustilanteissa monilukutaidon harjoittamista voidaan tehdä sekä yksin että yhteistyönä. (Opetushallitus, 2015, 38–39, 2016, 22, 23, 283.) Opetussuunnitelma korostaa myös yksittäisten aineiden ja opettajan roolia kielen suhteen.

> ”Jokaisella oppiaineella on oma kielensä, tekstikäytäntönsä ja käsitteistönsä. – – Opetuksessa edetään arkikielestä käsitteellisen ajattelun kieleen. Kielitietoisessa koulussa jokainen aikuinen on kielellinen malli ja myös opettamansa oppiaineen kielen opettaja.” (Opetushallitus, 2016, 28.)

Molemmissa opetussuunnitelmissa kehotetaan antamaan tilaisuuksia matemaattisen ajattelun, käsitteiden ja ratkaisumenetelmien oppimiseen systemaattisuutta korostaen. Matemaattisiin perusvalmiuksiin nostetaan täsmällisen ilmaisun harjoitteleminen sekä ajatusten ja perustelujen ilmaiseminen yksiselit-

teisesti. Myös piirrosten ja luovuuden käyttö mainitaan ajattelua tukevina menetelminä. (Opetushallitus, 2004, 158, 163, 164, 2016, 374–375.)

Lukion opetussuunnitelman mukaan matematiikassa *”tuetaan opiskelijan taitoa siirtyä toisesta matemaattisen tiedon esitysmuodosta toiseen”* (Opetushallitus, 2015, 129). Lisäksi mainitaan ajattelusta ja kielestä:

> ”Matematiikan opetuksen tehtävänä on tutustuttaa opiskelija matemaattisen ajattelun malleihin sekä – – opettaa käyttämään puhuttua ja kirjoitettua matematiikan kieltä sekä kehittää laskemisen, ilmiöiden mallintamisen ja ongelmien ratkaisemisen taitoja.” (Opetushallitus, 2015, 129.)

Monipuolisella arvioinnilla ja palautteella yritetään edistää opiskelijan matemaattista ajattelua, motivaatiota ja itseluottamusta sekä pitkäjänteisyyttä. Pitkän matematiikan osalta on mainittu myös matemaattisen tiedon luonteen ymmärtämisen mahdollisuus. (Opetushallitus, 2015, 129, 131.)

4.5 Erilaiset kielentämistehtävät

Kielentämistä voivat hyödyntää sekä opettajat tekemissään tehtävänannoissa että oppilaat ratkaisuissaan. Sarikka (2014, 21–28) jakaa kielentämistehtävät kielentämistehtävätyyppeihin ja kielentämisstrategioihin, joista kielentämistehtävätyypit ovat kielentämistehtävien mallityyppejä esimerkiksi matematiikan opettajalle, kielentämisstrategiat sen sijaan oppilaiden tuottamissa ratkaisuissa esiintyviä kielentämistyyppejä (ks. myös sanallisten tehtävien ratkaisumallit Joutsenlahti, 2009, 78–82). Tämän tutkielman opetuskokeilutehtävissä hyödynnetään ensin kielentämistehtävätyyppejä tehtävänannoissa ja myöhemmin analysoidaan, ovatko tutkittavat käyttäneet tiettyjä kielentämisstrategioita.

Kielentämistehtäville on esitetty seitsemän mallityyppiä: koodaus, täydennys, virheen etsintä, ratkaisusta tehtävä, ratkaisun argumentointi, tiedonseulonta ja omin sanoin selvitys (Sarikka, 2014, 26–28). Tämän tutkielman opetuskokeilutehtävissä ei käytetä suoraan yhtä tiettyä kielentämistehtävämallia, vaan malleja on voitu käyttää samassa tehtävässä useampia, koska tutkimuksen aihealue (algoritminen ajattelu, ohjelmointi, geometria) poikkeaa tavanomaisista matematiikan tehtävistä.

Koodauksella tarkoitetaan kääntämistä matematiikan eri kieliosa-alueiden välillä eli luonnollisen kielen kääntämistä matematiikan kielelle tai päinvastoin.

Ratkaisun argumentoinnissa matematiikan eri kielien avulla perustellaan ratkaisua, joka on joko valmiiksi annettu tai itse tehty. Ratkaisun argumentointi sisältää esimerkiksi käsitteisiin liittyvät teoreettiset ja soveltavat sanalliset tehtävät. Selvityksessä omin sanoin ilmaistaan kirjallisesti tai suullisesti pyydetty asia käyttämättä matematiikan symbolikieltä. Tiedonseulonnassa puolestaan etsitään tehtävänannosta ratkaisun kannalta olennaiset asiat, koska tehtävänannossa on ylimääräisiä tietoja. Täydentämistehtävissä annettuun ratkaisuun täydennetään olennaisimmat osat, kuten puuttuvat välivaiheet ja perustelut. Virheenetsinnässä valmiista ratkaisusta haetaan virheitä, jotka korjataan, ja lisäksi oikeat ja korjatut vaiheet perustellaan. Ymmärrystä ja soveltamistaitoa vaativassa tehtävätyypissä ratkaisusta tehtävä päätellään annetun täydellisen ratkaisun perusteella kysytty asia eli tehtävänanto, johon ratkaisu sopii. (Sarikka, 2014, 26–28.)

5 TUTKIMUSONGELMA JA -KYSYMYKSET

Algoritminen ajattelu on taito, jota jokainen nykyisessä yhteiskunnassa elävä tarvitsee sekä arjessa että työelämässä ainakin jollakin tasolla. Useissa maissa algoritmista ajattelua on korostettu koulutuksessa, ja ajattelutaitoa on yritetty opettaa usein ohjelmoinnin avulla. Ongelmana on kuitenkin, että Suomessa ajattelutapaa ei ole vielä selkeästi nostettu esille, eikä oppilaiden algoritmista ajattelua ole juurikaan tutkittu. Tutkimuksen tavoitteena on selvittää algoritmista ajattelua tutkittavilla yläkoulun oppilailla ja lukion lyhyen ja pitkän matematiikan opiskelijoilla. Samalla selvitetään, onko tutkittavien taustatekijöillä yhteyksiä algoritmisen ajattelun näkyvyyteen. Lisäksi halutaan tutkia, miten tutkittavat kokevat tutkimuksen tehtävät. Tarkoituksena on myös selvittää teorian ja tämän tutkimuksen tulosten perusteella, millaisista tehtävistä voisi olla hyötyä algoritmisen ajattelun kehittämisessä. Tämän tutkimuksen kysymykset muodostavat osittain kehitystutkimuksen ensimmäisen syklin.

Tutkielman pääkysymykset ovat:

1. Mikä on algoritminen ajattelu tutkittavilla?
2. Miten kielentäminen tuo ohjelmoinnissa algoritmisen ajattelun esille?
3. Miten tutkittavat kokivat tehtävät?
4. Millaisista tehtävistä voisi olla hyötyä algoritmisen ajattelun kehittämisessä?

Alakysymyksinä ovat:

- Tuoko kielentäminen/dokumentointi ohjelmointiin palautetta?
- Millaisia ovat tutkittavien taustatekijöiden yhteydet algoritmisen ajattelun näkyvyydessä? Taustatekijöinä ovat sukupuoli, koulutusaste, luokka-aste, lyhyt/pitkä matematiikka, ohjelmointikokemus ja matematiikan arvosana. Taustatietojen avulla tarkastellaan yhtäläisyyksiä ja eroja eri taustaryhmien tutkimustehtävien ratkaisuissa ja kyselylomakkeen vastauksissa.

Tutkimukseen esitetään muutamia hypoteeseja. Aiemmin ohjelmoinneilla tutkittavilla ohjelmointiosuudet saattavat sujuvat helpommin, erityisesti, jos ohjelmointikieli on tuttu. Tutkimuksessa oletetaan kuitenkin, että myös tutkittavat, jotka eivät ole ohjelmoineet aiemmin, saavat tutkimuksen tehtävistä jotakin irti. Lukiolaisilla oletetaan olevan matemaattisesti paremmat valmiudet tehtäviin kuin yläkoululaisilla, ja toisaalta yläkoululaisilla voi olla luokka-asteen mukaisia eroja. Lisäksi hypoteesina on, että matematiikan korkeammalla arvosanalla saattaa olla positiivista merkitystä tehtävien tuloksiin ja niihin perustuvaan kyselyyn. Algoritmisen ajattelun oletetaan näkyvän loogisena päättelynä ja ongelmien hajottamisena pienempiin osiin.

6 TUTKIMUSMENETELMÄT

Tämä tutkielma perustuu metodologialtaan osittain kehittämistutkimukseen (engl. design based research, design research). Ensimmäisiä kehittämistutkimuksia edustaa Brownin työ 1990-luvun alusta (A. L. Brown, 1992). Brown pyrkii siihen, että tutkimus jo itsessään loisi käytännön työvälineitä opetukseen. Toisaalta tutkimuksen pitää perustua tieteelliseen tietoon, olla luotettavasti tehty ja toistettavissa. Eri toimijoita ja kokonaisuuksia ei voi opetuksessa tutkia erillään, vaan kaikki osatekijät vaikuttavat kokonaisuuteen. (A. L. Brown, 1992, 143.) Kiinnostus kehittämistutkimusta kohtaan on 2000-luvun aikana kasvanut selvästi (Anderson & Shattuck, 2012). Kehittämistutkimusten määrä on huomattava myös esimerkiksi Suomessa opetusalan opinnäytetöissä.

Opetuksessa ilmenevät ongelmat ja opetuksen kehittäminen ovat kehittämistutkimuksen ytimessä. Pernaa (2013, 11) määrittelee kehittämistutkimuksen lähtökohdaksi halun opetuksen kehittämiseen tutkimuspohjaisesti. McKenney ja Reeves (2014, 11) pitävät opetuksellisen kehittämistutkimuksen erona muihin tutkimustyyppeihin sitä, että kehitystutkimuksessa pyrkimyksenä on sekä ongelmien ratkaisu että ratkaisujen hyödyntäminen. Tutkimusaiheet nousevat käytännönläheisesti todellisista opetustilanteiden tarpeista (McKenney & Reeves, 2014, 3). Yksiselitteistä määritelmää kehittämistutkimukselle ei kuitenkaan ole (Pernaa, 2013; The Design-Based Research Collective, 2003), mutta tavoitteena on kuitenkin tehdä teoriaan pohjautuvaa kehitystyötä, jossa arvioinnin ja iteroinnin avulla päästään parempaan lopputulokseen (Pernaa, 2013, 17–18). Kehitystutkimus ei varsinaisesti ole oma metodologiansa, vaan se käyttää kvantitatiivista, kvalitatiivista ja yleisimmin monimenetelmäistä lähestymistapaa ongelmien ratkaisuun. (Kananen, 2012, 19; McKenney & Reeves, 2014, 8; Pernaa, 2013, 21.)

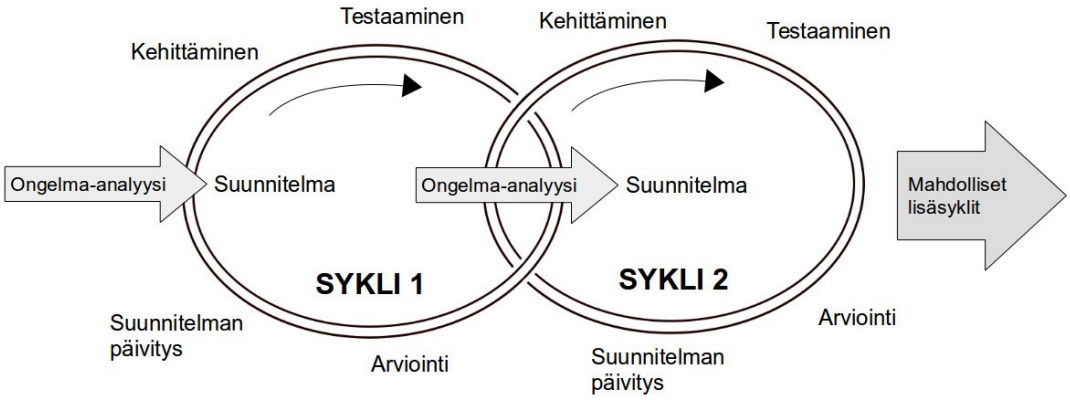
Kehittämistutkimus voidaan ajatella vaiheittaiseksi prosessiksi. Pernaan (2013, 17) mukaan kehittämistutkimuksessa suoritetaan aluksi ongelma-analyy-

si, jonka avulla havaitaan kehittämistarpeet sekä kehittämisen mahdollisuudet ja haasteet. Kun ongelman analysoinnin jälkeen tiedetään kehittämistavoitteet, voidaan luoda kehittämissuunnitelma. Kehittämistutkimuksen ominaispiirteitä ovat kehityssyklit, jotka koostuvat kehittämis-, arviointi- ja raportointivaiheista. (Pernaa, 2013, 17.) Kehittämistutkimuksen prosesseissa on kuitenkin runsaasti variaatiota (McKenney & Reeves, 2014, 12).

Kehittämistutkimuksen teossa olennaista on huomioida luotettavuus, vaikka syklien vaiheita voidaan kuvata joustavasti eri tavoilla. Pernaan tavoin opetuksellisen kehittämistutkimuksen periaatteita on määritellyt Design-Based Research Collective (2003). Oppimisympäristöjen kehityksen ja oppimisteorioiden kehityksen tulee nivoutua toisiinsa. Kehityksen ja tutkimuksen tulee edetä sykleissä, jotka koostuvat suunnittelusta, toteuttamisesta, analysoinnista ja uudelleensuunnittelusta. Tutkimuksen tulee johtaa teorioihin, joita voidaan soveltaa käytännön opetukseen ja sen suunnitteluun. Lisäksi tutkimuksen tulee selvittää, miten suunnitelma toimii autenttisessa ympäristössä. Tutkimuksen vaiheet ja syklit tulee dokumentoida tarkasti. (Pernaa, 2013, 20; The Design-Based Research Collective, 2003, 5.) Myös Edelson (2002, 116–117) määrittelee kehittämistutkimuksen tavoitteet melko samalla tavalla: tutkimuksen pitää olla tutkimuksiin pohjautuva, dokumentoitu systemaattisesti, arvioitu kehityksellisesti (muovataan analysoinnin, ratkaisujen ja suunnitelmien mahdollisia puutteita) ja yleistettävissä. Amielin ja Reevesin (2008) mukaan kehityksessä hyödynnetään olemassa olevia periaatteita ja teknologisia innovaatioita. Tärkeää on jalostaa ratkaisut käytäntöön ja edistää ratkaisujen käyttöönottoa (Amiel & Reeves, 2008). Kanasen (2012, 80–81) mukaan muutoksia ja tavoitteiden toteutumista tulee arvioida ja seurata.

Yhden syklin kehittämistutkimuksessa vaiheina ovat teoreettinen ongelma-analyysi, empiirinen ongelma-analyysi, kehittämisvaihe ja raportointi. Teoreettisessa ongelma-analyysissä tarkastellaan kirjallisuutta suhteessa tutkimusaiheeseen. Empiirisessä ongelma-analyysissä kirjallisuuden pohjalta nostetaan tutkimustarpeita, ja siihen voi sisältyä esimerkiksi kohderyhmälle teetetty kysely tai haastattelu. Kehittämisvaiheessa luodaan alustava kehittämistuotos ja raportointivaiheessa dokumentoidaan tulokset. Kahden syklin kehittämistutkimus vastaa yhden syklin tutkimusta, mutta empiirinen ongelma-analyysi ja kehittä-

misvaihe tehdään kahtena syklinä. (Pernaa, 2013, 185–186.) Kuviossa 2 on esitetty kaavio kehittämissykleistä.



KUVIO 2. Kehittämistutkimuksen syklit. Muokattu lähteestä (Pernaa, 2013, 19).

Kehittämistutkimus myös raportoidaan yleensä poikkeavasti muihin tutkimuksiin nähden. Yleisesti opinnäytetyössä raportin osat muodostuvat johdannosta, teoriaosasta, metodologiasta, tuloksista ja johtopäätöksistä. Kehittämistutkimuksessa raportin osat ovat johdanto, tutkimuskohteen kuvaus, tutkimusongelma, teoreettinen viitekehys, ratkaisun esittäminen, tutkimuksen luotettavuus, johtopäätökset ja kehittämisehdotukset. Teoria ohjaa kehittämisprosessia ja samalla on säilytettävä tutkimuksellinen ote. Kehitettävä tuote ja tutkimus ovat jatkuvasti yhteydessä toisiinsa. (Kananen, 2012, 15–20.)

Perusperiaatteiden pohjalta kehittämistutkimuksessa onkin yhtäläisyyksiä toimintatutkimukseen (Anderson & Shattuck, 2012, 17), esimerkiksi pyrkimys muutokseen ja samankaltaisten syklien käyttö prosessin aikana (Kananen, 2012, 41–42). Eroina kehittämistutkimuksen ja toimintatutkimuksen välillä ovat esimerkiksi seuraavat: toimintatutkimuksessa tutkija osallistuu kehittämiskohteensa toimintaan ja parannusta etsitään (ryhmän) toimintaan ei-sosiaalisten ilmiöiden (tuotteet, palvelut, prosessit, toiminnot, asiantilat) sijaan. Kehittämistutkimus poikkeaa myös puhtaasta laadullisesta tutkimuksesta, koska laadullisessa tutkimuksessa yritetään yleensä vain selittää ilmiö. Kehittämistutkimuksessa hyötyinä ovat havaitun ongelman poistaminen tai ainakin parantaminen, ja tunnusomaista kehittämistutkimukselle ovatkin uuden tuotteen tai esimerkiksi prosessin ominaisuuksien kehittäminen. Parempien vaihtoehtojen tunnistamisen li-

säksi vaihtoehtojen käyttökelpoisuutta on myös testattava. (Kananen, 2012, 20, 41–44.)

Kehittämistutkimuksen kritiikkinä ovat olleet pienet otoskoot ja se, että tutkimus toteutetaan kvalitatiivisesti (Edelson, 2002, 117–118; Pernaa, 2013, 20). Toisaalta kehittämistutkimus antaa mahdollisuuden myös lisätä kvantitatiivisia tutkimusmenetelmiä (Pernaa, 2013, 21). Kvalitatiivisten tutkimusmetodien kautta ei myöskään saada suureista tarkasteltua tilastollista merkitsevyyttä.

Tässä tutkielmassa on tehty laaja empiirinen ongelma-analyysi. Tutkimuksessa on ensin havaittu ongelma, joka on analysoitu tieteelliseen tietoon ja tutkimuksiin pohjautuen. Mahdollisiksi kehittämisen kohteiksi havaittiin algoritmisen ajattelun vähäinen käyttö opetuksessa, opetuksen epäyhtenäisyys ja puutteelliset keinot algoritmisen ajattelun opettamiseen ja osaamisen arviointiin. Myös opetussuunnitelmat ja erot eri maiden opetuskäytännöissä otettiin tutkimuksessa huomioon, koska Brownin (1992) mukaan opetussuunnitelmia ei voi erottaa opetuksen kokonaisuudesta. Tutkimuskohderyhmän kanssa toteutettiin opetuskokeilu ja kyselytutkimus (ks. lomake liite 2). Tämä tutkielma muodostaa kehittämistutkimuksen ensimmäisestä syklistä vain osan laajan ongelman ja sen kartoittamistarpeen vuoksi. Tarkoituksena on toteuttaa syklit myöhemmin kokonaisuudessaan tuottamalla kehittämismateriaaleja saatujen tulosten perusteella ja jatkamalla tietoisesti opetuksen kehittämistä.

Tässä tutkielmassa kyselytutkimusosio (survey-tutkimus) on vakioitu eli kaikille tutkittaville esitetään saman asiasisällön kysymykset (Vilkka, 2015). Kyselytutkimuksessa väitekysymysten vastaamiseen käytettiin neliportaista Likertin asteikkoa kvantitatiivisen analyysin helpottamiseksi. Neliportaisella asteikolla haluttiin saada kokemukset joko negatiivisiksi tai positiivisiksi jättämällä ”en osaa sanoa” -tyyppinen vastausvaihtoehto pois. Kyselyosuudessa on myös avoimia kysymyksiä, jotka analysoidaan laadullisesti.

Laadullinen tutkimus keskittyy joko käsityksiin tai kokemuksiin (Vilkka, 2015). Laadullisessa tutkimuksessa on tärkeää, että tutkija pyrkii objektiivisuuteen, eikä sekoita omia uskomuksiaan ja asenteitaan tutkimuskohteeseen (Eskola & Suoranta, 1998, 17). Kyselyosiossa avoimet kysymykset on pyritty tekemään objektiiviseksi sekä johdattelematta. Lisäksi on annettu yhtäläinen mahdollisuus sekä positiiviseen että negatiiviseen vastaamiseen ja perusteluihin. Kyselyssä on joustavasti käytetty sekä avoimia että suljettuja kysymyksiä, kos-

ka suljettuun kysymykseen saa vastattua lyhyestikin halutessaan. Avoimien kysymyksien vastauksia tyypitetään ja eritellään analysointia varten (Eskola & Suoranta, 1998, 181–188), mutta ne analysoidaan kvalitatiivisesti. Kvalitatiivisen tutkimuksen luotettavuuteen vaikuttavat tutkijan ennakko-oletukset ja asenteet (Eskola & Suoranta, 1998, 208). Tutkittavien vastatessa avoimiin kysymyksiin tutkija ei ole läsnä. Kvalitatiivista analyysia on suoritettu kvantitatiivista aineistoa ja analyysia tukevana eikä itsenäisenä kokonaisuutena. Avoimissa kysymyksissä ja väitekysymyksissä on tarkoituksellisesti myös päällekkäisyyttä.

Tutkimustehtävien analyysi tehdään kvalitatiivisesti kuvaillen, mutta analysointi perustuu myös vastausten kvantifioimiseen (Eskola & Suoranta, 1998, 164–165). Kvantifiointiin käytetään ennalta määrättyjä kriteerejä (esitetty liitteessä 3). Tutkimuksen luotettavuutta ja toistettavuutta pyritään lisäämään erityisesti tehtävien aihealueiden (algoritminen ajattelu, ohjelmointi ja kielentäminen) pisteytyksen jakamisella pienemmiksi kategorioiksi. Myös toistettavuuden kannalta arviointikriteerit ovat tärkeät. Johnsonin ja Onwuegbuzien (2004, 22) määrittelemää triangulaatiota käytetään osittain, koska algoritmisen ajattelun tehtävistä suoriutumista tutkitaan kvalitatiivisesti ja kvantitatiivisesti. Kehittämistutkimuksen kokonaisluotettavuutta lisäävät syklien määrä ja standardoidut mittarit (Pernaa, 2013, 21), joten tämän tutkimuksen kehittämistutkimuksen luotettavuuden osuus määräytyy vasta syklien muodostumisen perusteella.

Tutkimus on laadittu osittain poikittaistutkimuksena. Aikaulottuvuutta ja algoritmisen ajattelun osaamista on haettu vertailemalla eri vuosiluokkia. Tutkimus toteutetaan monimenetelmäisenä kehittämistutkimuksen osana (Johnson & Onwuegbuzie, 2004; Pernaa, 2013, 21).

7 TOTEUTUS

Tutkielmaan sisältyi opetuskokeilu, joka toteutettiin keväällä 2018 viidessä oppilaitoksessa. Kirjallisesti toteutetussa kielentämiskokeilussa oppilaat vastasivat itsenäisesti yhdestä neljään ohjelmointitehtävään, jotka käsittelivät perusgeometriaa ohjelmointikoodissa. Tutkittavien tarkoituksena oli kielentämällä ja koodia tulkitsemalla ratkaista tehtävät. Lisäksi tutkimukseen liittyi kyselylomake, joka sisälsi tehtäviin liittyviä suljettuja väitekysymyksiä ja avoimia kysymyksiä sekä kysymyksiä taustatiedoista. Opetuskokeiluun liittyvä tutkittavan lomake (sisältäen saatekirjeen, luvat, ohjeen, esimerkkitehtävät, tehtävät ja kyselylomakkeen) on esitetty liitteessä 2. Lomakkeiden vastaukset analysoitiin kvalitatiivisesti tai kvantitatiivisesti muuttujien luonteesta riippuen. Reliabiliteettia ja validiteettia arvioitiin lomakkeen, opetuskokeilun ja arvioinnin suhteen.

7.1 Opetuskokeilu

Algoritminen ajattelu soveltuisi useisiin matematiikan aihealueisiin, mutta kielentämiskokeilun tehtävät rajattiin aihealueeltaan geometriaan. Geometrian valinnan koettiin tukevan kirjallisen kielentämisen visuaalisen ilmaisemisen mahdollisuutta (kuvakieli). Geometria tarjoaa matemaattisille käsitteilleen selkeän mielikuvan, vaikka ohjelmointi voidaan kokea abstraktiksi. Lisäksi haluttiin painottaa, että ohjelmointia ja algoritmista ajattelua voidaan integroida matematiikan sisältöihin, tässä geometriaan.

Kaikki tutkittavat saivat samanlaiset tehtävät, joten tehtävien tasoa ei muutettu kouluasteiden ja luokka-asteiden mukaan. Tällä ratkaisulla pyrittiin takaamaan tehtävien vertailukelpoisuus. Tehtävissä käsiteltiin pääasiassa peruskoulun 7. luokka-asteen opetussisältöihin kuuluvia geometrian sisältöjä, mutta tehtäviin sisällytettiin myös korkeampien luokka-asteiden sisältöjä. Korkeampien

luokka-asteiden sisältöjä yritettiin käyttää kuitenkin niukasti ja niin, ettei niiden osaaminen ole kynnys tehtävistä suoriutumiseen. Tällä haluttiin tutkia, pystyvätkö nuorimmat tutkittavat venymään tehtävissä ja toisaalta onnistuvatko lukiolaiset kaikissa tehtävissä, koska heille kaikkien matematiikan sisältöjen pitäisi olla tuttuja.

Opetuskokeilussa käytettiin välineenä kielentämistä, jonka avulla oli tarkoitus saada esille tutkittavien matemaattinen ja algoritminen ajattelu. On mahdollista, että jotkin algoritmisen ajattelun osa-alueet saatiin paremmin ja jotkin heikommin esiin kielentämällä. Lisäksi koodin tulkintaa käytettiin kielentämistä vastaavalla tavalla välineenä selvittämään tutkittavien ajattelua ja ohjelmointikoodista esille nostamia asioita (esimerkiksi miten koodi toimii, onko jokin koodin rakenne toista rakennetta hankalampi). Kun tutkittavat kirjoittivat paperille tehtävien ratkaisuja, saatiin tietoa myös oppilaiden kokemuksista. Tehtävien suunnittelussa käytettiin näkökulmallisesti hyväksi Sarikan (2014, 26–28) kielentämistehtävätyyppejä, mutta niitä jouduttiin mukailemaan. Erityisesti ohjelmoinnin käyttö vaati kielentämistehtäviin muutosta, ja lisäksi tehtävänannot jätettiin tehtävätyypeistä avoimiksi, koska haluttiin tutkia tutkittavien ajattelua monipuolisesti. Arvio kielentämistehtävätyypeistä tehtiin oletuksilla, mitä tutkittavien ratkaisut saattaisivat laajasti pohdittuna sisältää ja mitä kielentämistehtävätyyppejä tutkittavien ratkaisut mahdollisesti vastaisivat.

Opetuskokeilussa päädyttiin paperilliseen toteutukseen, jotta tekniset ja tietosuojaan liittyvät ongelmat saatiin minimoitua ja geometrian visualisoiminen olisi mahdollisimman helppoa. Tutkittavilla oli enemmän mahdollisuuksia havainnollistaa ajatteluaan piirtämällä kuin mitä heillä olisi tietokoneella tehdyssä toteutuksessa. Lisäksi haluttiin välttää ohjelmoinnille tyypilliset tilanteet, joissa ohjelma ei käänny virheiden vuoksi, koska oli huomioitava, että suuri osa tutkittavista oppilaista ei ollut ehkä aiemmin ohjelmoinut. Paperilla ohjelma ”toimii” virheistä huolimatta, eikä koodieditori tai ohjelmointiympäristö muokkaa oppilaan ajattelua. Toisaalta paperiversion toteutuksessa menetettiin autenttinen ohjelmointi.

Tehtävät painottuivat geometrian lisäksi ohjelmointiin, koska ohjelmointi liittyy algoritmiseen ajatteluun. Lisäksi ohjelmointi on uudessa opetussuunnitelmassa yhdistetty osaksi matematiikan oppiainetta. Ohjelmointikieleksi valittiin Python, koska Scratch on usein käytössä alemmilla luokka-asteilla, ja Scratch

olisi ollut liian yksinkertainen ja tylsä jo aiemmin ohjelmoinneille. Lisäksi Python soveltui paremmin geometrian toteuttamiseen ja paperitoteutukseen. Pythonin yksinkertaisilla kommenttiriveillä yritettiin auttaa ja selittää kaikkein haastavimpia osia ohjelmointikoodista. Mikäli tutkimuskohderyhmänä olisivat olleet ainoastaan yläkoululaiset, olisi Scratch saattanut olla toimiva vaihtoehto, koska ohjelmointi on vasta tulossa opetukseen. JavaScriptissä puolestaan syntaksi on hieman Pythonia haastavampi.

Tämän tutkielman opetuskokeilussa ohjelmointi perustui lähinnä valmiiseen ohjelmakoodiin itse tuotettavan ohjelmointikoodin sijaan, koska ohjelmointikieltä ei voitu opettaa oppilaille ja opiskelijoille lyhyen ajan vuoksi. Tutkimuksen tehtävien taso yritettiin asettaa sopivaksi ottaen huomioon oppilaiden hyvin erilaiset taustat, varsinkin aiempi ohjelmointikokemus ja matematiikan taso. Haasteena oli, että tämän tutkimuksen puitteissa algoritmisesta ajattelusta voitiin tutkia vain pieniä osia esimerkiksi tutkimuksen lyhyyden vuoksi.

Tutkimukseen pyydettiin luvat rehtoreilta, tutkimukseen osallistuvilta opettajilta, tutkittavilta ja alaikäisten tutkittavien huoltajilta. Tarvittaessa lupaa pyydettiin myös korkeammilta virkamiestahoilta. Malli saatekirjeestä ja luvasta on esitetty liitteessä 2, mutta sanamuodot poikkeavat toisistaan kohdejoukon mukaan.

7.2 Ohje ja esimerkkitehtävät

Tutkittaville jaetussa lomakkeessa saatekirjeiden ja lupien lisäksi ohjeistettiin lomakkeen täyttöön (ks. liite 2). Ohjeessa painotettiin yleisen ohjeistuksen lisäksi kielentämisen tärkeyttä, koska tutkittavien ajattelun ilmaiseminen oli tutkimuksessa arvokkainta, tehtävien ratkaisujen oikeellisuus oli toisarvoista. Kielentäminen käsitteenä saattoi tuntua tutkittavista vieraalta, joten kielentämisen ohjeistukseen lisättiin toivomus perustelemisesta ja arvauksista. Tutkittavia kehotettiin myös havainnollistavien kuvien piirtämiseen, koska niistä voisi olla hyötyä sekä tutkittavalle tehtäviä ratkaistaessa että tutkittavan ratkaisun analysoinnissa.

Esimerkkitehtävässä A (ks. liite 2) esitettiin matematiikan avulla, mitä kielentämisellä tarkoitetaan, koska tutkittavat eivät ole välttämättä tehneet kielentämistehtäviä aikaisemmin. Ohjelmointikoodia ei vielä käytetty, jotta se ei sekoittaisi kielentämisen tarkoitusta. Esimerkkitehtävän ratkaisussa yritettiin hajottaa

tehtävä yksityiskohtaisesti osiin ja jäsentää ratkaisu, käyttää mahdollisimman paljon matemaattisia käsitteitä sekä kirjoittaa auki geometriset kaavat.

Esimerkkitehtävässä B esitettiin lyhyen ohjelmointikoodin avulla, mitä koodin tulkitsemisella tarkoitetaan. Tässä tehtävässä keskityttiin koodin kommenttiin, muuttujaan, muuttujan arvon sijoittamiseen, ehtolauseeseen ja tulostamiseen. Tehtävässä käytettiin matematiikkaa mahdollisimman vähän, jotta koodin tulkinta olisi helpompi tuoda tarkoituksenmukaisesti esille. Ratkaisussa yritettiin esittää koodin rakenteet yksityiskohtaisesti niin, että tutkittavat, joilla ei ole aiempaa ohjelmointikokemusta, voisivat saada idean sekä ohjelmoinnista että ohjelmointikoodin tulkitsemisesta. Ratkaisussa yritettiin käyttää mahdollisimman paljon ohjelmoinnin käsitteitä, jotta aiemmin ohjelmoineet käyttäisivät niitä omissa ratkaisuissaan. Tutkimuksessa sallittiin, että tutkittavat käyttivät esimerkkitehtäviä apuna varsinaisia tutkimustehtäviä ratkaistessaan.

Esimerkkitehtävässä C esitettiin yksityiskohtainen yhdistelmäratkaisu kielentämisestä ja koodin tulkinnasta. Esimerkki sisälsi vastaavankaltaista geometriaa kuin varsinaisissa tehtävissä. Lisäksi tehtävään sisällytettiin suuri osa varsinaisten tehtävien ohjelmointikoodien rakenteesta, jotta tutkittavat, jotka eivät olleet aiemmin ohjelmoineet, saisivat vihjeitä koodin toiminnasta ajattelun tueksi. Ratkaisun luettavuutta yritettiin helpottaa osittamalla ratkaisu ja viittaamalla merkinnöillä (nuolet, sulkeet) tekstejä vastaaviin koodin osiin.

Algoritmiseen ajatteluun liittyviä yleistystä ja abstraktiota ei tässä tutkimuksessa painotettu, koska niiden tuottaminen lyhyisiin ja paperiversiolla toteutettaviin tehtäviin oli haastavaa. Yleistystä ilmeni ensimmäisen tehtävän koodipaloissa, koska samoja funktiokutsuja käytettiin toistuvasti eli jokaiselle kulmalle tai sivun mittaamiselle ei tehty joka kerta kokonaan uusia funktioita tai pidempää koodia, ongelma ratkaistiin jo kerran hyväksi todetulla funktiolla. Tehtävässä näytettiin kuitenkin vain funktioiden kutsut, koska funktioiden kirjoittaminen koodiin olisi pidentänyt koodia merkittävästi.

7.3 Tehtävä 1

Ensimmäisessä tehtävässä mukailtiin Papertin (1985) kuvailemaa LOGO-ympäristön kilpikonnaa, jossa kilpikonna liikkuu ja piirtää kulmia ja viivoja ohjelmoijan käskyjen mukaisesti. Tässä tehtävässä ei pyydetty tutkittavia

luomaan ohjelmointikoodia, koska aiempaa ohjelmointikokemusta ei välttämättä ole. Näin ollen koodipalat annettiin valmiiksi, ja tehtävänä oli järjestää palat oikeaan järjestykseen kuvavihjeen perusteella (ks. liite 2). Tehtävä sisälsi erilaisia kulman käsitteitä ja asteita, puolisuunnikkaan piirin ja pinta-alan. Ensimmäiseen tehtävään annettiin yksityiskohtainen malli valmiiksi helpottamaan kielentämistä ja koodin tulkintaa (koodipalasta G). Tutkittavan toivottiin ilmaisevan kielentämisellä ja koodin tulkinnalla perusteluita, mitä hän tehtävästä matematiikan ja ohjelmoinnin osalta ajatteli ja miksi hän muodosti valitsemansa järjestyksen. Tehtävää helpotettiin kuvavihjeellä, koodipalojen tasamäärällä (ei ylimääräisiä) ja valmiilla G-palan ratkaisulla.

Tehtävässä oikean ratkaisun muodostamiseksi tutkittavan oli laskettava sivujen pituuksia ja pohdittava ratkaisua kokonaisuudessaan, jotta käännökset (kulmien suuruudet, koodipalat A ja D) suoritettaisiin oikeassa järjestyksessä. Tähän vaadittiin ymmärrystä koodipalojen sisäisistä laskuista ja ymmärrystä siitä, että eri koodipalojen laskut liittyivät toisiinsa. Tehtävän tavoitteena oli, että vaikka järjestystä ei olisi saatu muodostettua, olisi kielentämällä esitetty yksittäisten koodipalojen sisältöä, ja miten tutkittava ajatteli ratkaisun toimivan esimerkiksi pelkässä matematiikassa. Lisäksi tehtävän koodipalojen järjestyksen mielekkyys kokonaisuudessaan oli tärkeää, vaikka yksittäisissä koodipaloissa olisi järjestyksen osalta epäloogisuutta.

Algoritmisen ajattelun osalta ongelma oli jo valmiiksi hajotettu pienempiin osiin, joten tutkittavan oli muodostettava yksittäisistä pienistä ongelmista kokonainen ratkaisu erityisesti loogisella päättelyllä. Lisäksi tehtävällä yritettiin kartoittaa koodipalojen hajottamista osiin ja vaiheittaista käsittelyä, kaavojen ja mallien käyttöä sekä oman ratkaisun analysointia ja arviointia algoritmisen ajattelun näkökulmasta.

Tehtävässä oli piirteitä Sarikan (2014, 26–28) kielentämistehtävien tyypeistä koodaus, ratkaisun argumentointi ja omin sanoin selvitys. Koodauksen osalta ei kuitenkaan käytetty suoraan matematiikan kieltä tai luonnollista kieltä, vaan ohjelmointikieltä, jossa matematiikka esiintyi. Tämä voidaan kuitenkin kääntää matematiikan kielelle tai luonnolliselle kielelle. Ratkaisun argumentointia esiintyi mukaillen, koska ei ollut välttämätöntä, että tutkittava esittäisi teoreettisia käsitteitä tehtävästä. Lisäksi kokonaisen valmiin tehtävän sijaan annettiin yksittäisiä valmiita koodipaloja, jotka on perusteltavissa. Omin sanoin selvitystä

oli sovellettavissa esimerkiksi puolisuunnikkaan pinta-alan kirjallisessa kuvailussa, joka oli mahdollista tehdä ilman matematiikan symbolikieltä. Toisaalta tutkittava saattoi kääntää ohjelmointikielen myös matematiikan symbolikieleksi.

7.4 Tehtävä 2

Toisessa tehtävässä annettiin valmiiksi kokonaisen tehtävän ratkaisu ohjelmointikoodina, mutta koodissa esiintyi matemaattisia virheitä (ks. liite 2). Virheet olivat matemaattisia, koska ohjelmointikoodiin perustuvat virheet olisivat olleet todennäköisesti liian haastavia tutkittaville, jotka eivät olleet aiemmin ohjelmoineet. Tehtävä sisälsi neliöiden, suorakulmioiden ja kolmion pinta-alojen laskemista. Lisäksi tutkittavan piti piirtää geometrisia kuvioita ja päätellä, mihin muuttujaan tietyt pinta-alat sisältyivät. Tehtävää helpotettiin pitämällä virheet siedettävinä, pyytämällä kuvan piirtämistä ja antamalla tasamäärä lähtötietoja (ei ylimääräisiä).

Tutkittavan tehtävän ymmärtämisen helpottamiseksi ja ajattelun tutkimisen tukemiseksi tutkittavaa pyydettiin piirtämään sanallisen tehtävänannon mukainen periaatekuva. Tämän jälkeen tutkittava etsi ja korjasi ratkaisukoodin virheet, nyt aiemmin piirretty kuva toimi ajattelun tukena. Lisäksi tutkittavan toivottiin ilmaisevan kielentämisellä ja koodin tulkinnalla perusteluita, mitä hän ajatteli tehtävästä matematiikan ja ohjelmoinnin osalta. Tehtävänannon sanallinen osuus oli laaja, joten tehtävän ratkaiseminen vaati tarkkuutta ja matemaattiset käsitteet oli hallittava. Virheiden korjaamiseksi tutkittavan tuli ymmärtää puutteet laskukaavoissa ja osata perustella ne.

Algoritmisen ajattelun osalta virheiden etsiminen ja korjaaminen sekä looginen päättely olivat pääosassa. Looginen päättely perustui tekstikielentämisen lisäksi myös visuaaliseen havainnollistamiseen. Lisäksi tehtävää tuli myös analysoida ja arvioida, hajottaa osiin ja ratkaisua perustella vaiheittain. Algoritmiseen ajatteluun liittyivät myös kaavojen ja mallien käyttö.

Tehtävässä käytettiin Sarikan (2014, 26–28) kielentämistehtävien tyypeistä suoraan virheen etsintää sekä mukaillen koodausta, ratkaisun argumentointia, omin sanoin selvitystä ja tiedon seulontaa. Tiedon seulontaa käytettiin apuna tehtävänannon muodostamisessa, sillä ratkaisun kannalta olennaisia lähtötieto-

ja oli tehtävänannossa paljon, ja ne piti ymmärtää. Tehtävänannossa ei annettu kuitenkaan ylimääräisiä tietoja.

7.5 Tehtävä 3

Kolmannessa tehtävässä annettiin tutkittavalle tilaisuus luovuuteen, koska tutkittavalla oli mahdollisuus ohjelmoida tehtävässä itsekin (ks. liite 2). Tutkittavat, jotka eivät olleet aiemmin ohjelmoineet, huomioitiin sillä, että tehtävään annettiin valmiiksi muuttujia sijoituksineen ja lisäksi muuttujia käyttäviä lauseita, joten tutkittavan ei ollut pakko lisätä kommenttien lisäksi muuta koodia, jos hän ei halunnut. Tämän tehtävän avulla voitiin havaita, mikäli tutkittavat käyttävät mallioppimista esimerkkitehtävien, varsinaisten tehtävien ja lisälähteiden avulla. Lisäksi tutkittavia pyydettiin muodostamaan ohjelmointikoodia vastaava geometriaan liittyvä tehtävänanto. Tehtävän ratkaisemiseksi koodin lauseet tarvitsi laskea oikein muuttujien oikeilla arvoilla. Lisäksi piti ymmärtää, millaisen tehtävänannon valmiista tai muokatusta koodista voi muodostaa. Ratkaisu oli myös kielennettävä ja tulkittava.

Algoritmisen ajattelun osalta ongelma käsitteli loogista päättelyä, analysointia ja arviointia, hajottamista osiin, ratkaisun vaiheistusta sekä kaavoja ja malleja. Lisäksi muista tehtävistä poiketen tehtävässä ilmeni myös suunnittelua.

Tehtävässä käytettiin Sarikan (2014, 26–28) kielentämistehtävien tyypeistä mukaillen koodausta, ratkaisun argumentointia, omin sanoin selvitystä, täydentämistehtävää ja ratkaisusta tehtävää. Tehtävässä annettiin osittainen tai valmis ratkaisu, joka salli täydentämisen välivaiheilla. Lisäksi koodiin piti kirjoittaa täydentäviä kommentteja, ja valmis ratkaisu perustella kielentämällä ja koodin tulkinnalla. Tehtävä mukaili myös ratkaisusta tehtävää, koska ratkaisuun suunniteltiin tehtävänanto. Ratkaisu ei kuitenkaan ollut alkuperäisenä täydellinen, koska siitä puuttuivat ainakin kommentit.

7.6 Tehtävä 4

Neljännessä tehtävässä annettiin valmiiksi ohjelmointikoodi, jonka sisältö piti kielentää ja koodin toiminta tulkita (ks. liite 2). Tehtävä oli samankaltainen kuin esimerkkitehtävät B ja C. Tehtävän matemaattinen sisältö koostui neliön ja

ympyrän käsitteistä ja laskutoimituksista, ja oli näin ollen tehtävistä matemaattisesti haastavin yläkoululaisille. Tehtävää helpotettiin niin, että ratkaisusta tehtiin helpommin lähestyttävä jakamalla se jo valmiiksi pienempiin osiin koodin rakenteen ja sisällön mukaan. Lisäksi koodin kommenteissa esitettiin piin ja neliöjuuren käsitteiden vastaavuus ohjelmointikielen syntaksissa. Ratkaisussa ohjelman toiminnan kuvaaminen perusteellisesti vaati ymmärrystä matematiikasta ja koodin toiminnasta, mutta tehtävä yritettiin laatia niin, että ratkaisusta voisi saada irti pieniä yksittäisiä asioita tai kokonaiskuvan pitkäjänteisellä yrittämisellä.

Algoritmisen ajattelun osalta ongelma käsitteli loogista päättelyä, analysointia ja arviointia, hajottamista osiin, ratkaisun vaiheistusta sekä kaavoja ja malleja. Tehtävässä 4 käytettiin Sarikan (2014, 26–28) kielentämistehtävien tyypeistä mukaillen koodausta, ratkaisun argumentointia ja omin sanoin selvitystä.

7.7 Kyselylomake

Kyselylomakkeessa oli kolme osiota: taustatiedot, suljetut väitekysymykset ja avoimet kysymykset (ks. liite 2). Taustatiedoissa kysyttiin sukupuolta, koulutustasoa, matematiikan arvosanaa sekä ohjelmointikokemusta. Sukupuolen osalta päädyttiin antamaan vain vaihtoehdot nainen ja mies, koska sukupuoli muu olisi jäänyt aineistossa pieneksi, jolloin ryhmää ei tutkittavien anonymiteetin turvaamiseksi olisi voinut käyttää. Lisäksi vaihtoehto muu olisi voinut tuottaa väärin käytettynä ylimääräistä virhettä. Koulutustasossa selvitettiin koulutusastetta, luokka-astetta ja lukion osalta matematiikan opintojen laajuutta (lyhyt tai pitkä matematiikka). Ohjelmointikokemus jaettiin kuuteen osakysymykseen, jotta tiedettiin, minkä tasoinen ja laajuinen tutkittavien kokemus oli.

Väitekysymyksissä esitettiin väittämiä ohjelmointiin, kielentämiseen ja algoritmiseen ajatteluun liittyvistä osa-alueista. Vastausvaihtoehtoja oli neljä (1 = Täysin eri mieltä, 2 = Jokseenkin eri mieltä, 3 = Jokseenkin samaa mieltä, 4 = Täysin samaa mieltä). Avoimissa kysymyksissä kysyttiin tutkittavien kokemuksia ohjelmointi- ja kielentämistehtävistä, niiden tasosta, hyödyllisyydestä ja toteutuksesta.

7.8 Analysointi

Ennen analysointia tutkimuksesta poistettiin puutteelliset ja lupaa vailla olevat palautetut lomakkeet. Kyselylomakkeen taustatiedoissa sukupuolen osalta tyhjäksi jätetty vastaus tulkittiin vaihtoehdoksi muu tai ei halua vastata, eikä kyseisiä tutkittavia otettu mukaan sukupuolta koskeviin analysointeihin. Ohjelmointikokemuksen osalta kategorisoitiin lomakkeiden vastausten perusteella kolmiportaisesti: ei lainkaan, vähän ja paljon ohjelmoineet. Tutkittavien, jotka eivät olleet koskaan ohjelmoineet, ohjelmointikokemus oli ”ei ole”. Jos tutkittava oli ohjelmoinut vain kerran koulussa tai muuten kokeiluna, niin hän oli ”vähän” ohjelmoinut. Mikäli tutkittava oli ohjelmoinut enemmän kuin kerran, niin hän oli ”paljon” ohjelmoinut. Lisäksi muodostettiin muuttuja runsaasti ohjelmoineista. Tähän katsottiin vastausten perusteella kuuluvan tutkittavien, joilla ohjelmointikokemus oli erityisen runsas.

Tutkittavien ratkaisemat tehtävät kvantifioitiin. Kvantitatiivisen aineiston muodostamiseksi ratkaisut pisteytettiin kategorisoiden aihealueiden algoritminen ajattelu, ohjelmointi ja kielentäminen suhteen. Aihealueet sisälsivät tarkempia kategorioita, jolloin aihealueiden arvostelu oli strukturoidumpaa. Yksittäinen pieni kategoria arvosteltiin pistein 0-1, 0-2 tai 0-3. Kategorioiden ja pisteytyksen muodostamisessa huomioitiin tehtävien luonne, yksittäisen tehtävän osalta ratkaisujen laatu ja eri tutkittavien ratkaisujen laadussa havaittavat erot. Näiden perusteella pisteytysportaikkoa saatettiin lisätä tai laskea osittain myös tehtävän sisällön painotuksen mukaan. Tutkimuksen tehtäväkohtaisiin tuloksiin otettiin mukaan lomakkeet, joista kävi ilmi tutkittavan yritys suorittaa tehtävä. Pisteytysperusteet on esitetty yksityiskohtaisemmin liitteessä 3. Analysoinnissa aihealueen kokonaissuoriutuminen suhteutettiin kategorioista saatavaan yhteispistemäärään.

Väitekysymyksissä käytettiin neliportaista Likertin asteikkoa. Osa vastaajista arvioi kokemustaan jatkuvalla asteikolla portaikosta huolimatta. Nämä yksittäiset väliportaat otettiin mukaan tutkimukseen. Neliportaisella asteikolla pyrittiin pakottamaan vastaajat valitsemaan kokemuksensa joko samaa tai eri mieltä olevaksi ilman ”en osaa sanoa” -vaihtoehtoa. Likertin asteikkoa käsiteltiin tästä huolimatta ordinaaliasteikkona. Avoimet kysymykset analysoitiin kvalitatiivisesti.

Avoimet kysymykset tarkensivat kyselylomakkeen väitekysymyksiä ja antoivat osallistujalle perustelumahdollisuuden kokemuksiinsa.

Aineistoa analysoitiin kvantitatiivisesti tunnusluvuin moodi, mediaani, keskiarvo ja keskihajonta muuttujan tyypistä riippuen. Lisäksi tarkasteltiin riippuvuussuhteita ja merkitsevyyksiä. Merkitsevyyksiä laskettiin kahden toisistaan riippumattoman numeerisen muuttujan tapauksessa Mann-Whitneyn testillä ja kolmen toisistaan riippumattoman ryhmän tapauksessa Kruskal-Wallisin testillä. Kahden tai useamman ryhmän binominaalisen muuttujan merkitsevyyden testauksessa käytettiin Chi-square-testiä. Tilastollisen merkitsevyyden rajana pidettiin p-arvoa alle 0,05. Aineiston analysointi suoritettiin IBM:n SPSS Statistics -ohjelmiston versiolla 25.

7.9 Validiteetti ja reliabiliteetti

Tutkimuksessa oli mukana 5 eri oppilaitosta, mikä paransi yleistettävyyttä, koska oppilaitoksissa oli myös alueellista hajontaa. Otoskoko oli kuitenkin pienehkö, mikä taas heikensi yleistettävyyttä ja heikompien tilastollisesti merkitsevien erojen löytymistä. Erityisesti alaryhmäanalyyseissä vertailtavien otosten koko jäi pieneksi. Myös tehtäväkohtaisesti aineistoa kavensi se, että tutkittavat eivät vastanneet kaikkiin tehtäviin. Tehtävien analysoinnissa jako aihealueiden sisällä kategorioihin lisäsi luotettavuutta tehtävien pisteytyksen yhdenmukaisuuteen. Tutkimuslomakkeen esittäminen ja pisteytysperusteet helpottivat tutkimuksen toistettavuutta. Tulkinnan yhdenmukaisuutta lisäsi myös se, että tehtävälomakkeet käytiin kahteen kertaan läpi kaikkien tutkimuslomakkeiden palautuksen jälkeen saaden ennen lopullista arvostelua kokonaiskuva tehtävistä suoriutumisesta. Tehtävät arvosteltiin yksi kerrallaan, jotta arvostelu pysyisi mahdollisimman yhtenäisenä. Kaikkien lomakkeiden tulkinta suoritettiin yhden henkilön toimesta.

Kouluilla oli erilaiset käytännöt tehtävien tekemiseen, mikä vaikutti tehtävistä suoriutumiseen. Kolmen viikon lomakkeiden täyttöajasta jouduttiin joustamaan ja luopumaan esimerkiksi koulun tarpeiden, loma-aikojen ja tutkimuksen lupakäsittelyjen vuoksi. Osa joutui tekemään tehtävät omalla ajallaan kun taas osalla tehtävien teko järjestettiin tuntityöskentelynä. Osassa tapauksista oppilaille tarjottiin myös positiivista vaikutusta arvosanaan.

Objektiivisuuteen pyrittiin sillä, että tutkittavat tekisivät tehtävät itsenäisesti ilman lomakkeen ulkopuolista opetusta tai opastusta. Tätä varten lomake sisälsi ohjeita ja esimerkkitehtäviä, jotta aloitus tehtävien suorittamiseen olisi yhdenvertaisempi ja jotta tutkittavat suoriutuisivat tehtävistä myös ilman aiempaa ohjelmointikokemusta. Opettajilla saattoi kuitenkin olla merkitystä tehtävien teon motivoimiseen ja ohjeistuksiin. Ei voitu myöskään taata, että tutkittavat tekivät suorituksensa täysin itsenäisesti.

Tällä tutkimuksella saatiin selvitettyä algoritmisesta ajattelusta vain pieniä osa-alueita. Tehtävien ja kyselyiden osalta ei voitu taata, että ne mittaisivat kattavasti (tai ylipäätään) tutkittavaa aihetta. Tutkittavien välinen yhteistyö yritettiin poistaa, jotta yhden tutkittavan ajattelu ei näkyisi useamman tutkittavan ratkaisuissa, vaikka yhteistyö ja ryhmässä tekeminen voisivatkin kehittää algoritmista ajattelua.

8 TULOKSET

Tutkimuksen tuloksista esitetään yleisesti tutkittavien taustatiedot, tehtäväkohtaiset tulokset ja niihin liittyvät lisähuomautukset, kyselylomakkeen väitekysymysten ja avointen kysymysten tulokset sekä vertailut näiden kyselytulosten ja tehtävien tulosten välillä. Tuloksissa myös vertaillaan eri taustatietoryhmien välistä suoriutumista ja kokemuksia tehtävistä. Tulosten käsittely kootaan yhteen ja kokonaisuuden perusteella vastataan tutkimuskysymyksiin. Tulokset on esitetty liitteissä 5, 6, 7 ja 8.

8.1 Taustatiedot

Tutkimukseen osallistui viisi oppilaitosta. Tutkimuslomakkeiden alkuperäistä määrää ei tiedetä, koska alunperäisestä arvioidusta lomakkeiden määrästä kaikkia lomakkeita ei ole laitettu tutkittaville jakoon. Palautetuista 109 lomakkeesta 69 otettiin mukaan tutkimukseen ja 40 hylättiin (36 lomaketta hylättiin puuttuvien tai kielteisten lupa-asioiden vuoksi ja neljä lomaketta karsittiin puutteellisina).

Kyselylomakkeista koottu yhteenveto tutkimuksen osallistujien taustatiedoista on esitetty liitteen 4 taulukossa. Tutkittavista (31) 44,9 % oli naisia, (34) 49,3 % miehiä ja muita (5) 5,8 %. Sukupuoleen muu kuuluvat osallistujat, jotka olivat jättäneet vastaamatta kyselylomakkeen osalta vain sukupuolen kohtaan sekä osallistujat, jotka eivät olleet ehtineet täyttää taustatietoja. Tutkimuksen aineisto painottuu yläkoulun oppilaisiin. Tutkittavista seitsemäsluokkalaisia oli (15) 21,7 %, kahdeksasluokkalaisia (20) 29,0 % ja yhdeksäsluokkalaisia (27) 39,1 %. Lukiolaisten määrän jäädessä pieneksi (7) 10,1 % lukion lyhyen ja pitkän matematiikan ryhmät ja eri vuosikurssit yhdistettiin (abiturientteja ei alunperinkään otettu tutkimukseen mukaan).

Matematiikan arvosanat sijoittuvat osallistujilla välille 5-10, ja arvosanat kahdeksan (17 vastaajaa) ja yhdeksän (19 vastaajaa) ovat aineistossa yleisimmät. Tulokset käsiteltiin matematiikan arvosanan osalta kuuden eri arvosanan (5-10) mukaan ryhmittelemättä arvosanoja kahden tai useamman arvosanan kategorioihin. Suurin osa tutkittavista ei ollut osallistunut tietotekniikkakurssille tai -kerhoon (58,5 % vastanneista), mutta tähän saattoivat vaikuttaa tulkinnat eri kouluissa ja eri vastaajien välillä (esim. itsenäinen osallistuminen yhden päivän lyhyelle kurssille, osallistuminen koulun TVT-tunneille, osallistuminen erilaisiin teknisiin valinnaisaineisiin).

Aiemman ohjelmointikokemuksen osalta jouduttiin käyttämään erityisesti tulkinnanvaraisuutta. Aineisto ryhmiteltiin kolmeen kategoriaan ”ei ohjelmoinut” (55,4 % vastanneista), ”ohjelmoinut vähän” (20,0 % vastanneista) ja ”ohjelmoinut paljon” (24,6 % vastanneista) kyselylomakkeen ohjelmointia käsittelevien kysymysten 4a, 4c-4f perusteella. Jos tutkittavat muistivat ohjelmoineensa, mutta eivät esimerkiksi muistaneet ohjelmointikieliä tai eivät arvioineet ohjelmointiin käyttämiään aikoja, niin tutkittava on tulkittu kuuluvaksi kategoriaan ”ohjelmoinut vähän”. Aineiston perusteella vaikuttaisi siltä, että noin puolella osallistujista on kuitenkin jokin käsitys ohjelmoinnista ja sen kokeilemisesta jollakin tasolla.

Osa tutkittavista saattaa sekoittaa yleisesti tieto- ja viestintätekniikan kurssin ohjelmointikokemukseen. Osa osallistujista ei muista käytettyä ohjelmointikieltä, ja osa ei tiedä, mitkä kielet ovat ohjelmointia, sillä vastauksina olivat esimerkiksi ”suomi” tai ”Excel”. Tähän tutkimukseen hyväksyttäviä ohjelmointikieliä vastauksissa olivat Python (6 kpl), JavaScript (4 kpl), Java (3 kpl), Lua (2 kpl), DrRacket (1 kpl), Frozen-harjoitus (1 kpl), GML (1 kpl) ja Scratch (1 kpl). Maksimimäärä ohjelmointikieliä, joita yksittäiset osallistujat raportoivat, oli kolme. Ohjelmointiin liittyvien kysymysten perusteella vaikuttaisi siltä, että kahdeksalta tutkittavalta löytyy runsaammin ohjelmointikokemusta (tunnistavat ohjelmointikielet, heillä on yleensä useampia ohjelmointivuosia ja/tai käytettävä kerta-aika on melko pitkä, ohjelmoinnin taajuus voi vaihdella).

Ohjelmointikokemus on aineistossa vahvasti sukupuolittunut. Naisista ”ei ohjelmoineiden” kategoriaan kuului (21) 67,7 %, miehistä (15) 44,1 %, naisista ”vähän ohjelmoineisiin” (9) 29,0 % ja miehistä (4) 11,8 % ja naisista ”paljon ohjelmoineisiin” (1) 3,2 % ja miehistä (15) 44,1 %. Lisäksi runsaasti ohjelmoineiden joukkoon arvioidut osallistujat olivat kaikki miespuolisia. Tietotekniikkakurs-

sin ja -harrastuksen osalta sukupuolittuminen ei ollut näin suurta, vaikka miespuolisten harrastuneisuus oli suurempaa kuin naisten. Tietotekniikan osalta osuuksiin voivat jälleen vaikuttaa tulkinnallisuus lomakkeen kysymyksen 4b suhteen sekä alueelliset erot harrastusmahdollisuuksissa ja koulujen erilainen tietotekniikkaopetuksen tarjonta.

8.2 Tehtävät

Kaikkien tutkimustehtävien arviointi on suoritettu algoritmisen ajattelun, ohjelmoinnin ja kielentämisen aihealueiden avulla. Näiden aihealueiden sisäisistä kategorioista osa voi olla sisällöltään melko lähellä toisiaan, mutta pisteytys niiden välillä voi silti vaihdella, koska näkökulma on niissä erilainen. Yksittäisen tehtäväratkaisun pisteytys on tulkinnanvaraisinta, mikäli ratkaisutaso vaikuttaa jäävän pisteytysportaikossa pisteiden puoliväliin, mutta pisteytysportaikon kasvattaminen ei ole järkevää. Tehtävien tulokset on käsitelty ensin tehtäväkohtaisesti ja tehtävien tulokset on koottu yhteen aihealueiden ja kategorioiden mukaan liitteen 6 taulukkoon ja taustamuuttujien mukaan liitteen 7 taulukoihin. Alaluvussa 8.4 käsitellään tehtävien kokonaispisteytykset ja otetaan huomioon myös kyselylomakkeen tulokset, ja loppuluvuissa vastataan tulosten perusteella tutkimuskysymyksiin.

8.2.1 Tehtävä 1

Tehtävään vastasi 62 osallistujaa. Tehtävään tuotettiin paljon ratkaisuja, joissa on täytetty vain taulukko tai osa taulukosta (26 vastaajaa, 41,9 %). Tämä vaikuttaa selkeästi pisteytykseen, koska ilman muunlaista kirjallista osuutta ajattelun ilmaiseminen on hankalaa, ja taulukon täyttämisen perusteella on voitu tulkita vain loogisen päättelyn osuutta. Taulukon täyttämällä pisteitä saa enintään kaksi algoritmiseen ajatteluun ja koko tehtävään. Vain taulukkoa täytetyissä ratkaisuissa algoritmisen ajattelun pisteosuus (13,5 %) jäi selkeästi alhaisemmaksi kuin niillä, jotka ilmaisivat ajatteluaan myös tekstin tai kuvien avulla (50,7 %).

Liitteen 6 taulukossa tuloksissa on huomioitu kaikki vastaajat, joten pelkän taulukon täyttäneet heikentävät pisteissä algoritmisen ajattelun ja koko tehtävän

kokonaissuoriutumista. Koko tehtävästä pisteosuuden keskiarvo vain taulukon tehneillä oli 4,0 % ja muilla 45,9 %. Tutkittavilla, jotka tekivät muutakin kuin taulukon täydentämisen, pisteosuuden keskiarvo kielentämisessä oli 44,4 % ja ohjelmoinnissa 43,4 %.

Algoritmisessa ajattelussa tarkasteltiin loogista päättelyä, analysointia, hajottamista ja kaavoja. Loogisen päättelyssä täydet pisteet sai 34 vastaajaa (54,8 % vastanneista). Arvostelussa huomattiin, että osa oli käyttänyt kahdesti A-koodipalaa (6 vastaajaa, 9,7 %), ja ratkaisu toimii myös näin. Palojen D ja A keskinäisestä väärästä järjestyksestä ei vähennetty pisteitä (väärä AD-vastaus 13 vastaajalla, 21,0 % ja oikea DA-järjestys 18 vastaajalla, 29,0 %). Muutamalla kulmat tai kulmien tyypit olivat väärässä järjestyksessä, vastaus oli muuten epäjohdonmukainen tai sivun 4 jälkeen suoritettiin vielä 90 asteen kääntyminen tai suorakulman tekstin tulostaminen. Liitteessä 9 on esitetty esimerkkiratkaisu, jossa on kattavasti kielennetty koodin eri vaiheita ja toimintoja, ja ymmärretty ohjelman toiminta.

Analysoinnissa ja arvioinnissa arvioitiin ratkaisun perustelua kokonaisuudessaan. Tässä oli suurta vaihtelua: osa arvioi ratkaisuaan ohjelmoinnin ja matematiikan osalta yksityiskohtaisesti ja osa ei. Muutamalla ratkaisua oli arvioitu yleisellä tasolla, joten analysointi ei tuottanut erityistä hyötyä, eikä ilmaissut selkeästi algoritmista ajattelua. Yksittäisissä ratkaisuissa esiintyi algoritmisen ajattelun yleistämisen käyttöä, koska samojen funktioiden toimintojen vastaavuus eri syötteillä oli huomattu.

Hajottamisen ja algoritmien osalta koko ratkaisu oli saatettu hajottaa osiin kielennettäessä ja selittää koodin vaiheita, mutta koodipalojen sisältöä ei välttämättä oltu hajotettu pienempiin osiin. Koodin hajottaminen osiin (erityisesti ehtorakennetta käsittelevä koodipala A sekä piiriä, pinta-alaa ja tulostusta käsittelevä H) oli usein melko puutteellista. Kolme vastaajista oli hajottanut taulukon koodipalajärjestyksen piirtämällä niistä kuvituksen sarjakuvamaisesti. Piirin ja pinta-alan sekä kuljetun matkan laskuja oli laskettu melko harvoissa ratkaisuissa, ja kuljetun matkan laskeminen oli vahva edellytys koodipalojen oikean järjestyksen aikaansaamiseksi. Kaavojen ja mallien käyttö, esimerkiksi laskujen laskeminen ja ilmaiseminen, oli melko harvinaista, varsinkin laskuista ja kulmista esitetyt perustelut.

Ohjelmoinnissa ratkaisut olivat vaihtelevat yksilöittäin ja eri osa-alueittain. Keskimäärin pisteitä saatiin 25,2 % maksimista ja kaikissa osa-alueissa pisteet olivat keskimäärin alle yhden. Esimerkiksi funktioista oli saatettu ymmärtää kääntymiset ja sivujen mittaamiset, mutta toisaalta tulostuksesta ei oltu ilmaistu mitään. Kahden A-koodipalan käyttö taulukossa oikein osoitti hyvää ehtorakenteen osaamista. Summamuuttuja (*kuljettuMatka*) oli ymmärretty toisinaan mielenkiintoisesti (esimerkiksi sivu1+sivu2=6 eikä oikea ratkaisu 9), vaikka tehtävä oli saatettu silti ratkaista oikein. Ohjelmoinnin rakenteiden hallitsemisen arviointi oli suhteellisen hankalaa, mikäli kielentäminen oli vähäistä.

Kielentämisen osalta kuvan piirtämistä oli hyödyntänyt vain viisi vastaajista eli havainnollistavia kuvia oli piirretty tähän tehtävään hyvin vähän. Käsitteitä (erityisesti ohjelmoinnin) oli poimittu mallikohdasta, mutta käsitteitä ei välttämättä käytetty itsenäisesti. Muutamissa ratkaisuissa looginen päättely (koodipalojen järjestys) saattoi olla hyvää/oikeita, mutta kielentäminen oli tehty todella yleisellä tasolla, jolloin esimerkiksi ohjelmointiin ja matematiikan löytämiseen sekä ymmärtämiseen ohjelmointikoodista oli vaikea saada riittävää osaamisnäyttöä. Toisaalta runsaalla kielentämisellä ratkaisuista huomattiin esimerkiksi muuttujan ja funktion käsitteen sekavaa käyttöä, laskujen laskemattomuutta, print-funktion väärinymmärryksiä (ratkaisuista ilmeni, että print-funktiolla esimerkiksi piirretään kuvioon kulma, eikä tuoteta print-funktion tekstiä).

8.2.2 Tehtävä 2

Tehtävään 2 vastasi 55 osallistujaa. Algoritmisessa ajattelussa osa-alueet olivat virheiden etsiminen ja korjaaminen, looginen päättely, analysointi, hajottaminen ja kaavat. Virheiden löytämisessä *rakennustenPintaAla* korjattu usein arvoon 164 (piharakennukset ja asuinrakennus) tai 14 (vain piharakennukset) jo sijoitusvaiheessa ennen for-silmukkarakennetta. Pellon pinta-alan korjaus lausekkeen perään (*:2*) on koodillisesti Python-kielelle oikein. Myöhemmin tämä voi aiheuttaa ongelmia, jos sulkujen käyttö unohtuu. Kolmio oli saatettu ratkaisuissa piirtää ja korjata koodissa oikein, mutta kielentämällä laskiessa kolmion jakolasku oli usein unohtunut. Kysytyn pinta-alan muuttujaa oli korjattu usein puutteellisesti, esimerkiksi osa summamerkinnöistä oli jäänyt korjaamatta, tai lauseeseen oli lisätty virheellisesti muuttujia. Liitteessä 9 on esitetty esi-

merkkiratkaisu, jossa on käytetty kuvitusta mittoineen ja selitetty koodista löytyneet virheet korjauksineen.

Loogisessa päättelyssä tarkasteltiin erikseen kuvan ja tekstikielentämisen osuuksia. Yhdeksän vastaajista (16,4 %) piirsi tehtävään ainoastaan kuvan, mikä heikensi kokonaispisteitä huomattavasti. Kuvan piirtäminen onnistui vastaajilta hyvin ja kuvat vastasivat tehtävänantoa. Keskimäärin pisteitä kuvasta saatiin 1,36 (arvostelu 0-2 p). Toisaalta kuvien puutteet liittyivät keksittyihin tai vääriin arvoihin, pellon väärään kokoon, rakennuksien puuttumiseen tai rakennusten vääriin muotoihin. Tehtävänantoa ei ollut aina ymmärretty, välillä kuvat olivat mittasuhteiltaan täysin pielessä (neliönmuotoisista suorakaiteita, kuviot eivät sisäkkäin), eikä hahmotettu käytettävää tehtävän termistöä. Osalla oli myös ongelmia suuruuslukujen kanssa kertolaskussa (esim. asunnon pinta-alan arvo pihan pinta-alaa suurempi).

Analysoinnissa ja arvioinnissa oli tehtävän 1 tavoin suurta vaihtelua ja osa kuvaili ratkaisuaan vain pintapuolisesti. Pienessä määrässä ratkaisuista ohjelmointikoodia ei oltu käsitelty vaan tehtävä oli ratkaistu pelkän tehtävänannon perusteella. Hajottaminen näkyi osittain jo tehtävänannon ongelmaa purkavan kuvan piirtämisessä sekä koodin virheiden käsittelyssä, ja toisaalta moni oli osannut tehtävän käsittelyn vaiheittain, mutta esimerkiksi laskujen ja kaavojen käsittely oli osin puutteellista.

Ohjelmoinnin osalta tehtävänantoa ja muuttujien nimiä ei täysin ole tehtävässä ymmärretty, eikä ole tiedetty, mitä koodissa lasketaan. Silmukkarakenteen ymmärrystä osoitettiin harvoin. Silmukkarakenteesta ei hahmotettu mitä rakennuksia siinä lasketaan yhteen ja miten. Jos tehtävä ratkaistiin pelkällä tehtävänannolla koodia lukematta, ohjelmointiosaamisen osoitus kielentäen jäi heikoksi. Kommentit liittyvät osittain silmukkaan, minkä vuoksi kommentteihin liittyvät pisteet jäivät todennäköisesti alhaiseksi (0,25 p; 0-2 pisteestä).

Kielentämisen osoitus jäi puutteelliseksi, jos vastauksessa oli pelkkä kuva. Vastaajilla oli kuitenkin useita eri tapoja ratkaisuun: osalla vain kuvaa, toisilla vain laskuja ja osalla paljon sanallista selitystä. Lisäksi löytyi edellisten yhdistelmiä. Matematiikan käsitteet perustuivat suurelta osin pinta-alaan (esiintyi tehtävän muuttujissa), mutta osa oli myös käyttänyt ansiokkaasti muitakin geometrian käsitteitä. Kielentämällä oli usein saatu tehtyä looginen ratkaisu (0,95 p; 0-2 pisteestä). Matemaattisia käsitteitä ilmaistaan enemmän kuin ensimmäisessä

tehtävässä tehtävätyypinkin vuoksi. Ohjelmoinnin käsitteitä esiintyi todella vähän ja termejä oli lähinnä vain silmukkaa käsitelleissä ratkaisuissa.

8.2.3 Tehtävä 3

Tehtävään 3 vastasi 44 osallistujaa. Algoritmisen ajattelun aihealueesta tarkasteltiin loogista päättelyä, suunnittelua, analysointia, hajottamista ja kaavoja. Loogisen päättelyn osalta esiintyi vaihtelua siinä, oliko tehtävä yritetty ratkaista kokonaisuudessaan ja miten hyvin koodin yhteys matematiikkaan oli ymmärretty. Ratkaisuista havaittiin, että suuri osa oli ymmärtänyt koodissa esiintyvän yhteenlaskun ja tulon oikeilla luvuilla, mutta laskut oli silti saatettu jättää laskematta tai yhdistäminen geometrian kaavoihin tai termeihin oli jäänyt tekemättä.

Analysoinnissa ja arvioinnissa oli suurta vaihtelua ansiokkaasti perustelluista ratkaisuista pelkkiin koodin lukujen merkintöihin. Joissakin ratkaisuissa geometriaa oli pohdittu kielennettäessä hyvin tai omaa tehtävänantoa ja ratkaisua oli arvioitu perusteellisesti siitäkin huolimatta, että päättely saattoi johtaa aritmetiikkaan tai tehtävään sopimattomaan geometriseen kuvioon. Hajottamisen osalta suuri osa ansiokkaasti kielennetyistä ratkaisuista eteni koodin mukaan vaiheittain järkevästi ja sekä tehtävänanto että sen ratkaisu oli hajotettu sopiviin osiin. Osa oli hajottanut tai havainnollistanut ongelmaa myös kuvalla. Lisäksi aineistossa oli ratkaisuja, joissa koodi oli hajotettu järkevästi esimerkiksi kolmeen osaan ja kommentit olivat hyödyllisiä, vaikka kielentäminen ja matematiikan yhdistäminen geometriaan oli ollut haasteellista. Hajottamisesta saatiin tässä tehtävässä keskimäärin yli puolet maksimipisteistä (1,09 p; 0-2 pisteestä).

Osallistujille annettiin tehtävässä vapaus suunnitella tehtävänanto geometrian ja lyhyen valmiin koodin reunaehdoilla. Lisäksi annettiin mahdollisuus koodin lisäämiseen. Ratkaisuissa käytettiin tästä huolimatta suhteellisen vähän geometriaa (17 vastaajaa, 38,6 % vastaajista) ja tehtävänantoja (19 vastaajaa, 43,2 % vastaajista). Laaditut tehtävänannot olivat rutiininomaisia ja suoraviivaisia laskutehtäviä ilman taustatarinaa. Tehtävänannon oheen oli usein piirretty havainnollistava kuva. Tehtävänannoista tyypiltään selkeästi vain aritmetiikkaan liittyviä oli 7 vastaajalla (15,9 % vastaajista) ja geometriaan liittyviä 11 vastaajalla (25,0 % vastaajista). Liitteessä 9 esitetään esimerkkiratkaisu suorakulmaisel-

le särmiölle. Ratkaisussa oli myös tehty oikeaoppinen kommentointi koodiin (ei näy kuvassa).

Osa tutkittavista ei ole muodostanut omaa tehtävänantoa, vaikka tehtävään on tehty kielentämällä ratkaisu ja laskuja. Lisäksi aineistossa oli ratkaisuja, joissa oli myös koodiin liittymättömiä tehtävänantoja (esimerkiksi olisi tarvinnut lisätä tai muokata koodia, jotta se vastaisi muodostettua tehtävänantoa, tai tehtävänanto ei liittynyt yhtään koodiin). Vaihtelu oli kuitenkin jälleen suurta, osa tehtävänannoista ja ratkaisuista oli luovia ja erinomaisia. Tehtävässä 3 esiintyy hieman algoritmiseen ajatteluun liittyvää rinnakkaisuuden ominaisuutta. Tehtävää ratkaistaessa osallistuja joutuu pohtimaan samaa asiaa samanaikaisesti kahdesta eri näkökulmasta – geometrisen tehtävänannon muodostamisesta ja koodin tulkitsemisesta.

Ohjelmoinnissa havaittiin, että ratkaisuissa on ymmärretty hyvin muuttujat ja muuttujien arvojen vastaavuus lauseissa ja yritetty tuottaa koodiin sopivia ja hyödyllisiä kommentteja. Tämä myös näkyy kyseisten kategorioiden keskimääräisissä pisteissä (muuttujat 0,93 p; kommentit 0,82 p; 0-2 pisteestä). Omaa koodia oli lisätty vain harvoin (5 vastaajaa, 11,4 % vastaajista). Omissa koodeissa ei käytetty ehto-, silmukka- tai metodirakenteita (funktiorakenteita), vaan lisäykset perustuivat uusiin muuttujiin ja laskutoimituksiin.

Kielentämisessä havaittiin, että perustelujen pisteet jäivät tehtävässä melko alhaisiksi (1,07 p; 0-3 pisteestä). Toisaalta matematiikka oli löydetty koodista melko hyvin, ja muuttujien arvojen yhteys lausekkeissa oli ymmärretty, vaikka yhteyttä geometriaan ei olisi muodostettu. Tehtävänantoon tai ratkaisuun oli piirretty usein kuva (15 vastaajaa, 34,1 % vastaajista), ja kuvat olivat suorakulmioita, suunnikkaita, suorakulmaisia särmiöitä, kolmioita tai ympyröitä. Lisäksi tässä tehtävässä matemaattisia käsitteitä käytettiin melko paljon, monipuolisesti ja itsenäisesti (tehtävässä oli annettuna vain termit luku, summa ja tulo). Ohjelmoinnin käsitteitä esiintyi ratkaisuissa vain vähän (liittyen muuttujiin ja kommentteihin), mikä on ymmärrettävää lyhyen koodin vuoksi.

8.2.4 Tehtävä 4

Tehtävään 4 vastasi 44 osallistujaa. Algoritmiseen ajatteluun liittyivät kategoriat looginen päättely, analysointi, hajottaminen ja kaavat. Looginen päättely oli

pääpiirteissään hyvää, mikäli oli yritetty koko tehtävää, jolloin ratkaisun loogisuus oli säilynyt usein. Lisäksi koodista oli löydetty ja ymmärretty usein keskeiset matemaattiset osuudet. Loogisen päättelyn keskimääräiset pisteet nousivatkin yli puoleen maksimipisteistä (1,07 p; 0-2 pisteestä).

Kaavojen osalta mielenkiintoista oli se, että koodin kaavoja oli tässä tehtävässä yritetty avata enemmän, ja ajattelua kaavoista ja laskuista oli kielennetty hyvin. Ympyrän kaava oli kielentämisen perusteella oikein 11 vastaajalla (25,0 % vastaajista). Toisinaan ympyrän kaava oli väärin niin, että toinen säteen osuus oli jäänyt kaavasta pois. Tämä saattaa johtua osittain koodista, jossa ympyrän alan laskemisen kohdalla on rivinvaihtoja. Ehtorakenteessa olevaa erotusta ja neliöjuurta käsiteltiin kielentäessä jonkin verran, mutta osassa vastauksista melko suppeasti. Liitteessä 9 on melko kattavasti kuvattu, hajotettu ja laskettu koodin osia.

Analysointi ja hajottaminen noudattelivat keskimäärin melko hyvää tasoa. Tässä tehtävässä ratkaisulle oli annettu valmiiksi koodia paloihin hajottavat neljä laatikkoa. Laatikoista täydennettiin keskimäärin noin 3 laatikkoa (1 laatikko tai yritys 9 kpl; 2 laatikkoa 5 kpl; 3 laatikkoa 9 kpl; 4 laatikkoa 21 kpl). Analysointi oli yleisesti hyvää niillä, jotka olivat yrittäneet käydä koodia kokonaisuudessaan läpi ja kielentää ajattelunsa myös epävarmoissa kohdissa. Koodista oli tällöin usein pystytty poimimaan matematiikkaa ja tehtävän idea, jolloin analysointi ja arviointi riitti ainakin puoliin kategorian pisteistä. Hajotuksen osalta puutteita esiintyi vähän pinta-aloja käsittelevässä osassa ja erityisesti if-osan hajottamisessa. If-osasta oli käsitelty vain osa laskuista (neliöjuurta ja erotussivua ei oltu kielennetty), ja tulostus oli jätetty usein kokonaan mainitsematta.

Tehtävä sisälsi ohjelmoinnin osalta silmukkaa lukuun ottamatta paljon erilaisia rakenteita. Ohjelmoinnin rakenteita oli ymmärretty tässä tehtävässä melko hyvin. Kielentämisen perusteella 50,0 % vastaajista vaikutti ymmärtävän ohjelman toiminnan ja siihen liittyvän termistön oikein, kun taas ratkaisuista seitsemässä (15,9 % vastaajista) esiintyi selkeää puutteellisuutta. Ohjelma ei esimerkiksi ”piirrä” ympyröitä tai neliöitä. Muuttujat ja lauseet oli ymmärretty pääosin hyvin, keskimäärin ratkaisuista annettiin pisteitä yli puolet (1,11 p; 0-2 pisteestä). Tulostuksia käsiteltiin ratkaisuissa harvoin.

Ehtorakenteessa näyttäisi olevan ongelmia, koska if-osan ehtoa ei ollut ratkaisujen perusteella täysin ymmärretty. Toisinaan esimerkiksi koko if-raken-

teen sisältö käsitettiin ehtona ja toisinaan if-ehtoon oli yhdistetty vain seuraava erotusta käsittelevä lause, eikä koko if-lohkoa ennen else-rakennetta. Tällöin aiheutuu ongelmia myös else-osan ymmärtämisessä, jos ehtoa ei ole ymmärretty. Kielentämisen perusteella else-osan oli ymmärtänyt oikein 12 vastaajaa (27,3 % vastaajista) ja puutteellisesti 16 vastaajaa (36,4 % vastaajista). Muutama ratkaisu viittaisi siihen, että kommentteja oli jäänyt lukematta tai ymmärtämättä koodista, esimerkiksi tieto kuvioiden sisäkkäisyydestä.

Kielentämisen aihealueessa positiivisena havaintona oli termien käytön määrä, koska sekä matemaattisia että ohjelmoinnin termejä esiintyi ratkaisuissa. Toisaalta tehtäväpohja tukee tässä termien käyttöä, mutta termejä oli käytetty osin myös itsenäisesti. Kuvia oli piirretty melko usein ratkaisun oheen (19 vastaajaa, 43,2 %) ja niistä suuri osa oli piirretty tehtävän mukaisesti täysin oikein. Ratkaisu eteni usein johdonmukaisesti kielentämisen avulla, matematiikkaa oli löydetty koodista ja lisäksi kaavoja oli perusteltu suhteellisen hyvin.

8.3 Kyselylomake

Kyselylomakkeen osalta muutamat kohdat tulkittiin tyhjiksi yksittäisillä tutkittavilla, jos vastauksen tulkinnassa oli epäselvyyttä, erityisesti väitekysymyksissä. Kyselylomakkeen avoimissa kysymyksissä ei saatu riittävästi tutkittavia, jotta olisi voitu ryhmitellä ja vertailla vastauksia tilastollisesti.

8.3.1 Väitekysymykset

Tutkimuksen väitekysymyksiin vastanneiden tutkittavien määrä vaihteli välillä 57-63 henkilöä (82,6-91,3 %). Koko aineistossa väitekysymyksissä mediaani oli kaikissa kysymyksissä ”jokseenkin eri mieltä” (arvo 2) tai ”jokseenkin samaa mieltä” (arvo 3). Kysymyskohtaiset vastausosuudet ja tunnusluvut on esitetty liitteen 5 taulukossa. Ohjelmoinnin osalta kaikissa kysymyksissä mediaani oli 2. Kielentämisen osalta väittämässä kielentämisen hyödyllisyydestä oppimisessa mediaaniksi saatiin 3 samoin kuin kuvien piirtämisen helppoudessa. Algoritmisessa ajattelussa mediaaniksi 3 saatiin loogisen ajattelun käytössä, monimutkaisten ratkaisujen yksinkertaistamisessa, aiemmin opitun matematiikan hyödyntämisessä ja ratkaisujen arvioinnin helppoudessa.

Sukupuolen mukaan verrattaessa merkitsevät erot miesten ja naisten välillä tulivat kysymyksissä 11, 13 ja 23 (p-arvot 0,006; 0,046 ja 0,031 tässä järjestyksessä), joissa miehet olivat enemmän samaa mieltä väittämien kanssa. Lukio- ja yläkouluopiskelijoita verrattaessa lukiolaiset antoivat merkitsevästi positiivisempia vastauksia ja olivat enemmän samaa mieltä väittämissä 5 (p-arvo 0,036), 6 (p-arvo 0,008), 7 (p-arvo 0,011), 14 (p-arvo 0,040), 24 (p-arvo 0,025), 27 (p-arvo 0,049), 32 (p-arvo 0,003), 34 (p-arvo 0,037), 35 (p-arvo 0,012) ja 37 (p-arvo 0,004). Merkitsevät erot luokka-asteryhmillä verraten tulivat kysymyksiin 5, 6, 7, 14, 19, 22, 24, 32 ja 37 ja kaikissa kysymyksissä pääsuuntauksena yksittäisiä luokka-asteiden poikkeamia lukuun ottamatta oli, että korkeamman luokka-asteen opiskelijat olivat enemmän samaa mieltä väittämien kanssa.

Matematiikan arvosanoilla verrattaessa merkitsevät erot eri arvosanaryhmien välillä saatiin kysymyksiin 7 (p-arvo 0,015), 19 (p-arvo 0,048) ja 24 (p-arvo 0,048). Pääsuuntauksena kysymyksissä on, että parempia arvosanoja saaneet vastasivat väitteisiin positiivisemmin, mutta trendissä on yksittäisiä poikkeamia eri arvosanoissa. Tutkittavien määrät eri alaluokissa jäävät melko pieniksi, mikä voi tehdä suurempaa prosentuaalista vaihtelua tuloksiin. Aiemman ohjelmointikokemuksen suhteen merkitsevät erot ryhmien välillä kysymyksiin 11 (p-arvo 0,011), 13 (p-arvo 0,039), 23 (p-arvo 0,013) ja 25 (p-arvo 0,026). Kysymyksissä selkeästi suuntaus kaikkien kysymyksien osalta, että mitä enemmän tutkittavat olivat aikaisemmin ohjelmoineet, sitä enemmän tutkittavat olivat samaa mieltä väitteiden kanssa. Väitteet ohjelmointikoodin kirjoittamisen helppoudesta (kysymys 11), ohjelmointikoodin kirjoittamisesta pitämisestä (kysymys 13) ja ohjelmointikoodin jakamisesta vaiheisiin ja sääntöihin (kysymys 25) liittyvätkin vahvasti aikaisempaan ohjelmointikokemukseen, jolloin kysymysten merkitsevä ero oli melko odotettua.

8.3.2 Avoimet kysymykset

Ohjelmointitehtävien tason osalta osallistujat ilmaisivat, että tehtävät olivat vaikeita, mutta osa piti siitä, että tehtävissä oli haastetta (kysymys 38). Syiksi arvioitiin, ettei ohjelmointia ollut vielä opetettu, aiempi ohjelmointiharjoittelu oli ollut helpompaa tai ohjelmointikieli oli ollut ymmärrettävämpää kuin tutkimustehtävien Python-kieli. Lisäksi haastavuuden ilmaistiin vaikuttaneen tehtyjen tehtä-

vien lukumäärään. Joidenkin tutkittavien mielestä kaikki tehtävät olivat yhtä vaikeita, osa taas ilmaisi tehtävien alkujen olevan helppoja ja sen jälkeen koodien vaikeutuvan, osan mielestä eri tehtävien taso vaihteli ja osa taas ilmaisi, että kun yritti jonkin aikaa ymmärtää tehtäviä, niiden ymmärtäminen helpottui. Yksi tutkittavista ilmaisi, että ehtorakenne oli vaikea ymmärtää. Muutama osallistuja koki, että koodit olivat yleisesti hankalia.

Toisten mielestä tehtävät olivat tasoltaan sopivia tai melko helppoja. Osa tutkittavista ilmaisi, että ohjeiden ymmärtäminen oli haastavaa, mutta osa taas ilmaisi esimerkkitehtävien helpottaneen ratkaisujen tekemistä. Osalle ymmärtämiseen ja tekemiseen oli kulunut kohtuuttomasti aikaa. Moni koki kaksi ensimmäistä tehtävää helpoiksi. Yksi tutkittavista perusteli, että tehtävät ovat vaikeita sellaiselle, joka ei osaa matematiikkaa. Lisäksi muutama ilmaisi, että välillä tehtäviä ymmärsi, mutta ajatukset saattoivat mennä sekaisin. Moni tutkittavista ajatteli joko niin, että tehtävät olivat vaikeita eivätkä he osanneet ratkaista niitä tai että he ajattelivat osanneensa, mutta epäilivät, että ymmärsivät varmaan aivan väärin. Tutkittavat olivat siis kovin epävarmoja.

> T30: ”Kaikki ovat vaikeita koska emme ole opiskelleet ohjelmointia ennen”
>
> T44: ”Teht. 1 oli selkein. Vaikka tuntui että osasin jotenkin, meni varmaankin ihan väärin. Tein varmaan liian yksinkertaisesti.”
>
> T68: ”Melko helppoja, jos ymmärsin tehtävänannon oikein. Pitää vain keskittyä kunnolla. Ohjelmointikoodeissa on määrätynlaista toistoa, mikä sekoittaa välillä.”

Monet kokivat ohjelmointitehtävät vaikeiksi tai haastaviksi (kysymys 40). Osa ei ymmärtänyt tehtäviä. Muutama vastaaja myös kommentoi, ettei ohjelmointia ole opetettu, ja ilmaisi sen vaikuttavan negatiiviseen kokemukseen. Joissakin vastauksissa tehtäviä kuvattiin myös tyhmiksi tai muilla epämääräisillä termeillä (”hämärä”, ”omituinen”, ”epäselvä”). Toisaalta tehtäviä pidettiin usein kivoina, hauskoina, hyvinä ja yksinkertaisina mutta yksityiskohtaisina. Vastaukset vaikuttivatkin puolittuvan negatiivisiin ja positiivisiin kokemuksiin. Paperitoteutus jakoi mielipiteet myös vahvasti kahtia. Osa koki, että tietokoneella ohjelmointikoodin olisi voinut ymmärtää ja ratkaista paremmin. Paperitoteutuksen koettiin vievän myös kauan aikaa. Toisaalta paperitoteutusta pidettiin mukavana vaihteluna ja se sai myös kannatusta.

> T29: ”[--] Koneella olisi luultavasti ollut nopeampaa, mutta paperilta ne jäävät todennäköisemmin mieleen paremmin.”
>
> T54: ”Liian vaikeita, ei ymmärtänyt…”
>
> T58: ”Ohjelmointitehtävät olivat ihan mukavia ja hyviä, olisin kaivannut enemmän haastetta. Toteutus oli mielestäni hyvä, mutta olisin tehnyt tutkimuksen mielummin digitaalisena.”
>
> T68: ”Ihan mielenkiintoista vaihtelua matemaattisiin pulmiin. Paperille on helpompi havainnollistaa piirtämällä kuvia, joiden tekeminen koneella olisi hidasta.”

Moni koki myös kielentämistehtävät tasoltaan vaikeiksi, hankaliksi tai melko vaikeiksi (kysymys 39). Osa ilmaisi, ettei ollut koskaan aiemmin kielentänyt. Osa koki, ettei ymmärtänyt kielentämistä tai joitakin yksittäisiä kohtia. Toisaalta monet kokivat kielentämistehtävät myös helpoiksi, kivoiksi ja hyviksi. Moni vertaili keskenään kielentämistä ja ohjelmointia, ja kielentäminen koettiin selkeästi ohjelmointia helpommaksi. Vaikeaksi koettiin juuri koodin ymmärtäminen ja sen kielentäminen. Osa koki kielentämisen tason erilaiseksi eri tehtävissä tai tehtävien eri osissa, osa taas koki tason olevan samanlainen kaikissa tehtävissä. Yksi osallistuja koki kielentämisen vievän liikaa aikaa yksinkertaisissa tehtävissä, mikä turhautti.

> T2: ”Kielentämistehtävät olivat vaikeita, koska niissä joutui miettimään paljon.”
>
> T46: ”Kielentämistehtävät olivat helppoja, kun ymmärsi koodin ja osasi matemaattisia kaavoja.”
>
> T57: ”Mielestäni kielentämistehtävät olivat helpompia kuin ohjelmointi-tehtävät. Luulen ymmärtäneeni edes jotakin kielentämisestä.”

Kysyttäessä mielipidettä kielentämisestä osallistujat ilmaisivat jälleen kielentämisen vaikeaksi tai jotenkin kummalliseksi, ja sen, ettei kielentämistä ollut tehty aiemmin (kysymys 41). Osa koki tehtävien välillä olleen tasoeroja. Muutama osallistuja ilmaisi, että aluksi kielentäminen oli hankalaa, mutta ajan kanssa kielentäminen muuttui helpommaksi. Osa ei pitänyt kielentämisestä. Toisaalta taas noin puolet osallistujista piti kielentämisestä, ja tehtäviä kuvattiin kivoiksi, helpoksi ja hyviksi. Osa ilmaisi kielentämisen olevan myös hyödyllistä.

> T33: ”En oikein tiennyt mitä kaikkea siihen piti kirjoittaa.”

> T46: ”Kielentämistehtävät olivat mukavia, kun sai kertoa, miten tehtävät on ratkaissut.”
>
> T58: ”En pitänyt niistä, helppojen ja itsestään selvien asioiden selittäminen oli ärsyttävää ja hyöty kyseenalaista.”

Kysymyksen 42 osalta selkeä enemmistö piti kielentämistä hyödyllisenä ja ajatteli siitä olevan hyötyä matematiikan oppimisessa jatkossa (27 selkeästi positiivista kokemusta, 18 selkeästi negatiivista kokemusta). Negatiivisissa kokemuksissa kuvattiin usein, että kielentämistä ei oltu ymmärretty. Moni ilmaisi, että kielentäminen on vielä uutta ja siihen tarvittaisiin opetusta. Kielentämisen koettiin usein selventävän pitkiä ja hankalia laskuja, ja sen koettiin auttavan myös matematiikan ymmärtämisessä. Ratkaisujen kirjoittaminen koettiin hyödylliseksi, kielentämisen koettiin lisäävän ymmärrystä ja helpottavan selittämistä ja ratkaisemista. Osa mainitsi myös, että kielentämisessä pitää todella ajatella. Yksi osallistuja ilmaisi, että hän oppi kielentämään paremmin ja toinen koki oppineensa uuden näkökulman.

> T44: ”Siinä käy laskun ehkä tarkemmin läpi vaihe vaiheelta kuin normaalisti, joten sillä voisi säästyä turhilta virheiltä kokeissa.”
>
> T58: ”Itselleni niistä ei ollut yhtään tai merkityksettömän vähän hyötyä. Jos on juuri aloittanut ohjelmoinnin ja/tai tehtävä on huomattavasti monimutkaisempi, hyötyä saattaa olla. Kielentäminen on turhan yksityiskohtaista ja vie liikaa aikaa. Vaikeissa tehtävissä yksinkertaiset kommentit monimutkaisista kohdista olisivat hyödyllisempiä.”
>
> T68: ”Tajusi koodikielien toiminnan periaatteita. Kielentäminen helpottaa laskujen toimintaperiaatteen hahmottamista, kun pitää ajatella mitä oikeasti tapahtuu.”

Yhteenvetona avoimien kysymysten vastauksista voidaan todeta, että useille osallistujille ohjelmointitehtävät olivat vaikeita ja kielentäminen mahdollisista haasteista huolimatta koettiin hyödylliseksi. Sekä ohjelmointiin että kielentämiseen tarvittaisiin opetusta. Osallistujat olivat hyvin eri tasoilla ohjelmoinnissa, joten eriyttäminen olisi tärkeää.

8.4 Tehtävien vertailu ja yhteenveto

Tehtävistä saaduissa pisteissä oli jonkin verran eroja eri tehtävien välillä (ks. liite 6). Keskimäärin ensimmäisestä tehtävästä pisteitä saatiin 28,3 %, toisesta

37,7 %, kolmannesta 33,9 % ja neljännestä 44,2 % maksimipisteistä. Hajontaa oli runsaasti tehtävien sisällä eri aihealueissa ja kokonaispisteiden keskihajonta muodostui erityisen suureksi. Vaihteluväli kaikkien tehtävien pisteissä oli 0-100 % pois lukien tehtävän 3 ohjelmointiosuus, jossa enimmillään pisteitä saatiin 57 % maksimista. Keskihajonnan ollessa suuri sekä kokonaispisteissä että aihealueiden pisteissä voidaan todeta oppilaiden olevan hyvin eritasoisia algoritmisen ajattelun, ohjelmoinnin ja kielentämisen osalta.

Hajontaa tarkasteltiin myös eri taustaryhmissä (ks. liite 7). Eri sukupuolilla hajontaa oli runsaasti kaikissa tehtävissä ja kaikissa tehtävien aihealueissa ja kategorioissa, eikä suoriutumisessa tullut tutkimuksen otannalla edes trendinomaisia eroja sukupuolten välille. Eri sukupuoliryhmien sisällä tasoero oppilaiden välillä on siis huomattava. Hajontaa oli runsaasti myös eri koulutustasojen ja vuosiluokkien oppilaiden ryhmissä. Tasoeroja oli siis oppilaiden välillä kaikilla vuosiluokilla, mutta lähes kaikissa tehtävissä ja osa-alueilla havaitaan trendinomaisia muutoksia tai merkitsevästi parempaa suoriutumista mitä korkeammalle vuosiluokalle tai koulutusasteelle edetään (ks. liitteet 7 ja 8). Kaikkien tehtävien pisteet yhdistäen saadaan merkitsevä muutos algoritmisen ajattelun kehittymisessä vuosiluokilla ja koulutusasteella edetessä. Tämä tukee tutkimusta edeltävästi tehtyä hypoteesia.

Tutkimushypoteesina oli, että aiempi ohjelmointikokemus ennustaisi parempaa tehtävissä menestymistä, mutta tämä hypoteesi ei saanut tutkimuksessa vahvistusta. Hajonta eri tehtävien pisteissä jää suureksi. Kaikki tehtävät yhdistäen ei havaittu tutkimuksen otannalla eroa eri aihealueilla suoriutumisessa. Toisaalta saatiin vahvistusta, että myös tutkittavat, jotka eivät olleet ohjelmoineet aiemmin, saivat tutkimuksen tehtävistä jotain irti. Matematiikan arvosana sen sijaan oli vahvasti tehtävissä menestymistä ennustava tekijä, mikä tuki hypoteesia (ks. liitteet 7 ja 8). Merkitseviä eroja saatiin tehtävän 2 ohjelmoinnin pisteitä lukuun ottamatta kaikkien tehtävien kaikissa aihealueissa. Korkeampi matematiikan arvosana siis ennusti parempaa suoriutumista algoritmisessa ajattelussa, ohjelmointitaidoissa ja kielentämisessä.

Ohjelmoinnin kokonaispisteissä verrattaessa eri matematiikan arvosanaryhmillä ei havaittu tilastollisesti merkitsevää eroa. Tämä johtuu todennäköisesti pisteosuuksien suurista keskihajonnoista, vaikka suuntaus näyttäisi siltä, että korkeammalla matematiikan arvosanalla myös suoriutuminen tehtävässä oli

keskimäärin parempaa. Jokaisessa tehtävässä suhteellisen moni oli käyttänyt matematiikan (ja mahdollisesti ohjelmoinnin) käsitteitä melko itsenäisesti, vaikka osa oli käyttänyt vain samoja käsitteitä kuin tehtävässä valmiiksi ilmeni. Erityisenä algoritmisen ajattelun ja ohjelmoinnin välisenä havaintona oli, että yleistystä oli havaittavissa muutamissa ratkaisuissa, esimerkiksi funktiota oli käsitelty ensin kerran, ja sitten todettu, että käytetään funktiota vastaavasti uudelleen esimerkiksi eri asteilla, joten funktioiden merkitys yleistämisessä oli ymmärretty.

Algoritmisesta ajattelusta saatiin pisteitä keskimäärin 35,1 % maksimimäärästä ensimmäisessä tehtävässä. Tähän vaikutti algoritmisen ajattelun pisteiden jääminen alhaiseksi, mikäli tehtävässä oli täytetty ainoastaan taulukkoa. Toisessa tehtävässä osaaminen oli keskimäärin 45,9 %, kolmannessa tehtävässä 41,1 % ja neljännessä 47,5 % (ks. liitteet 6 ja 8). Vaihtelu on osaamisessa oppilaiden välillä kuitenkin suurta. Yksittäisistä kategorioista suoriutumista on melko vaikea arvioida, koska tehtävät painottivat ja mittasivat eri kategorioita. Lisäksi ensimmäisessä tehtävässä kategorioiden pisteet ovat alhaiset johtuen ratkaisuista, joissa oli täydennetty vain taulukko. Kolmannessa tehtävässä hajottamista osiin oli osattu paremmin kuin muissa tehtävissä. Tehtävissä 2 ja 4 oli analysoitu ja käytetty kaavoja hieman paremmin kuin muissa tehtävissä. Loogisesta päättelystä oli keskimäärin saatu yli puolet pisteistä tehtävissä 1, 3 ja 4. Väitekysymysten perusteella osallistujat kokivat käyttäneensä loogista ajattelua, yksinkertaistusta ja matemaattisia kaavoja sekä hieman yleistämistä. Kokemukset vastasivat melko hyvin tehtävien kirjallisista ratkaisuista saatua osaamiskäsitystä ja pistemääriä. Avoimissa vastauksissa ei otettu kantaa algoritmiseen ajatteluun.

Ohjelmoinnin osaaminen ensimmäisessä tehtävässä oli keskimäärin 25,2 %, toisessa tehtävässä 15,8 %, kolmannessa tehtävässä 25,0 % ja neljännessä tehtävässä 40,6 % (ks. liitteet 6 ja 8). Toisen tehtävän pisteet jäivät alhaisiksi, mikä saattoi johtua silmukkarakenteesta, jonka havaittiin olevan haastava. Yleisesti silmukka-, funktio- ja ehtorakenteet olivat monelle hankalia, ja toisinaan muuttujien nimiä, alustusta ja arvojen tallentamista ei ymmärretty (esimerkiksi silmukan indeksimuuttuja ja summamuuttuja). Geometria ja kielentäminen paljastivat osalla ohjelmoinnissa virhekäsityksen – ohjelma ei piirrä kuvioita esimerkiksi laskiessaan piirejä ja pinta-aloja, ohjelma ei ymmärrä millaisesta kuviosta on kyse, se vaan laskee ihmisen tuottaman koodin mukaisesti. Kommen-

tointi osattiin keskimäärin melko hyvin. Mielenkiintoista on, että muuttujien ja lausekkeiden ymmärtäminen parani selkeästi ensimmäisestä tehtävästä neljänteen tehtävään. Tämä saattaa johtua muuttujien ja lausekkeiden ymmärtämisestä vähitellen, mikäli tehtävät on suoritettu numerojärjestyksessä. Hajonta oli kaikissa kategorioissa suurta. Ohjelmoinnin haastavuus heijastuu myös selkeästi väitekysymysten ja avointen kysymysten vastauksista. Väitekysymyksissä mediaani oli 2, joten ohjelmoinnin helppoudesta, hyödyllisyydestä ja mukavuudesta oltiin jokseenkin eri mieltä.

Kielentämisen osaaminen ensimmäisessä tehtävässä oli keskimäärin 25,9 %, toisessa tehtävässä 38,3 %, kolmannessa tehtävässä 35,8 % ja neljännessä tehtävässä 44,2 %. Tehtävissä 2 ja 4 suoriuduttiinkin muita tehtäviä paremmin kategorioissa perustelut/kielentämisen käyttö, matemaattisten kaavojen käyttö ja kielentämällä toteutettu jäsennelty ratkaisu. Selkeästi heikoin kategoria oli ohjelmointikäsitteiden käyttö. Matemaattisten käsitteiden käyttö lisääntyi tehtävänumeroiden mukaisessa järjestyksessä. Osaaminen vaihtelee oppilaiden välillä huomattavasti. Oppilaiden ratkaisuista ei erottunut erityisiä kielentämisstrategioita. Väitekysymysten perusteella kielentämisen hyödyllisyydestä ja kuvien piirtämisen helppoudesta oltiin jokseenkin samaa mieltä. Hyödyllisyys tuotiin esille myös avointen kysymysten vastauksissa. Mielenkiintoista on, että matematiikan jäsentämisen ja käsitteiden käytön osalta kielentämisen avun koettiin olevan jokseenkin eri mieltä, vaikka avoimissa vastauksissa hyöty matematiikassa tuotiin esille. Kokonaisuudessaan mediaani oli hieman yli 2. Avoimissa vastauksissa kielentäminen koettiin ohjelmointia helpommaksi, mutta melko haastavaksi. Suurin osa piti kielentämisestä tai koki sen hyödylliseksi avointen vastausten perusteella. Kielentämiseen kaivattiin opetusta.

8.5 Mikä on algoritminen ajattelu tutkittavilla?

Algoritmisesta ajattelusta oppilaat saivat tehtävistä keskimäärin 39,2 % pisteistä. Algoritmisen ajattelun osaaminen oli vaihtelevaa, ja tasoerot havaittiin kaikissa tehtävissä. Algoritmisen ajattelun eri kategorioissa vaihtelu oli myös suurta. Matematiikan korkeampi arvosana ja korkeampi koulutustaso tai vuosiluokka ennustivat parempaa algoritmisessa ajattelussa suoriutumista. Sukupuolella

tai aiemmalla ohjelmointikokemuksella ei havaittu merkitsevyyttä suoriutumisessa.

Keskimäärin heikoimmin oppilaat suoriutuivat kaavoista ja malleista, virheiden korjaamisesta sekä suunnittelusta. Toisaalta kyselylomakkeen vastauksissa oppilaat keskimäärin kokivat hyödyntävänsä oppimiansa yleisiä matemaattisia kaavoja ja malleja. Lisäksi algoritmisen ajattelun osa-alueista oppilaat kokivat käyttävänsä loogista ajattelua ja yksinkertaistusta. Korkeimmat osa-aluepisteet havaittiinkin loogisessa päättelyssä. Hypoteesin mukaisesti algoritminen ajattelu näkyi tutkittavilla loogisena päättelynä, ja hajottamistakin oli käytetty osassa tehtävistä onnistuneesti.

8.6 Miten kielentäminen tuo ohjelmoinnissa algoritmisen ajattelun esille?

Kielentämisen avulla saatiin tietoa oppilaiden algoritmisesta ajattelusta. Pelkkä ohjelmointi tai matemaattinen ratkaisu eivät tuo kokonaisuudessaan näkyväksi oppilaiden algoritmisen ajattelun prosessia. Algoritmisessa ajattelussa haasteiden syitä sekä vahvuuksia saatiin esille kielentämisen avulla. Esimerkiksi pelkkä ohjelmointikoodin virhe ei välttämättä paljasta perustana olevassa algoritmisessa ajattelussa tapahtuvaa virhettä. Jotkin algoritmisen ajattelun osa-alueet voisivat olla vaikeasti todennettavissa ilman kielentämistä, esimerkiksi analysointi-, arviointi- ja suunnitteluprosessit. Näistä osa-alueista saatiin kielentämisen avulla tietoa. Kielentämisen perusteella arvioitiin oppilaiden ohjelmointia, matemaattista ajattelua ja algoritmista ajattelua eri kategorioissa.

Kielentämisen ja dokumentoinnin havaittiin tuovan ohjelmointiin paljon palautetta. Kielentämällä saatiin lisätietoa ongelmista: millaisia virheet ovat, miksi ja miten niitä tehdään. Ohjelmoinnin haasteisiin saatiin näkyvyyttä kielentämisen avulla, muun muassa väärinymmärryksiin sekä rakenteiden ja syntaksin ymmärtämisen virheisiin. Esimerkiksi laadittu koodi toimi vahingossa, vaikka oppilaan ajattelu oli puutteellista. Muita havaittuja ohjelmoinnin ongelmia olivat esimerkiksi koodin logiikka, muuttujien nimet, lauseiden laskut, rakenteiden lohkot, ehdot ja muuttujien arvojen muuttuminen. Paljon ohjelmointia harrastaneilla havaittiin liiallistakin oikaisemista ja huonoja ohjelmointikäytäntöjä, mitkä eivät ilman kielentämistä helposti tule esille. Näin opettaja voi puuttua ongelmiin ajois-

sa. Paljon ohjelmoineilla havaittiin, että ohjelmointitermien käyttö ei kuitenkaan onnistunut.

Kielentäminen paljastaa osa-alueita, joissa eritasoiset oppilaat ovat lahjakkaita tai heikkoja. Kielentämisen avulla opettaja havaitseekin hankalia aihealueita ja eriyttämisen tarpeita, joita hän ei muuten ehkä tiedosta. Lisäksi kielentäminen antoi myös oppilaille viitteitä siitä, mitkä osiot olivat hankalia. Oppilaat joutuivat kielentäessään miettimään ja arvioimaan tiettyjä ohjelmointikohtia eri tavalla. Toisaalta kielentäminen paljasti myös sen, että oppilas, joka ei saanut koodia toimimaan, ymmärsi teoriaa ohjelmoinnista ja osasi käyttää esimerkiksi ohjelmoinnin termejä oikein. Oppimista saadaan näkyväksi kielentämisen avulla pelkän ohjelmoinnin sijaan. Lisäksi palautetta saatiin oppilailta ohjelman toiminnan oivaltamisesta ja hyvästä ajatteluprosessista.

8.7 Miten tutkittavat kokivat tehtävät?

Algoritmiseen ajatteluun liittyvistä kysymyksistä ilmeni, että oppilaat kokivat käyttäneensä tehtävissä loogista ajattelua, yksinkertaistusta ja aiemmin opittuja matemaattisia kaavoja. Lisäksi he kokivat käyttäneensä hieman yleistämistä. Näistä erityisesti looginen ajattelu ilmenikin tehtävien ratkaisuissa. Muista osa-alueista oppilaat olivat jokseenkin eri mieltä, vaikka vaihtelu oli suurta. Kyselyn tuloksia oli hankala arvioida eri taustatekijöiden suhteen, koska vastaukset jäivät usein yksittäisiksi mielipiteiksi. Oppilaat olivat ratkaisuistaan epävarmoja, sillä omien ratkaisujen oikeellisuuteen ja asioiden ymmärtämiseen ei luotettu, vaikka näiden osallistujien ratkaisut olivat usein esimerkillisiä.

Osa oppilaista koki ohjelmoinnin vaikeaksi ja tyhmäksi, erityisesti seitsemäsluokkalaiset. Syyksi osa ilmaisi sen, ettei ohjelmointia ollut vielä opetettu. Kokemukset ohjelmoinnin tasosta vaihtelivat huomattavasti. Osa osallistujista koki ohjelmoinnin kuitenkin kivaksi ja helpoksi. Muutama piti haasteellisuudesta ja koki tehtävät positiivisiksi pulmatehtäviksi. Osa koki, että matematiikan ja kaavojen osaamisella on yhteys ohjelmoinnin osaamiseen, ja tämä havaittiinkin merkitseväksi tekijäksi. Mielipiteet jakautuivat myös paperitoteutuksen suhteen. Digitaalisuutta kannatettiin mahdollisen koodin ymmärtämisen ja ajan säästymisen vuoksi. Paperiversiosta puolestaan pidettiin esimerkiksi vaihtelun, mieleen jäämisen ja helpon piirtämisen vuoksi. Ohjelmoinnin helppoudesta, hyödyllisyy-

destä ja mukavuudesta oltiin yleensä jokseenkin eri mieltä, vaikka vaihtelu kokemuksissa oli suurta.

Oppilaat kokivat kielentämisessä kuvien piirtämisen vastaamisen perustelemiseksi olevan helppoa ja tämä näkyi myös algoritmisen ajattelun kuvalla havainnollistavissa kategoriapisteissä suoriutumisena. Kielentämisen koettiin selventävän ratkaisuja ja matematiikan ymmärtämistä sekä lisäävän ajattelua ratkaisua laadittaessa. Suuri osa koki kielentämisen olevan hyödyllinen ja kiva menetelmä, mutta kielentämiseen toivottiin opetusta. Toisaalta osa ei pitänyt kielentämisestä eikä ymmärtänyt sen tarkoitusta. Lisäksi kielentämisen koettiin vievän aikaa, eikä kielentämisestä koettu olevan hyötyä yksinkertaisten asioiden kirjoittamisessa. Osa ilmaisi kielentämisen auttavan matematiikan ymmärtämisessä, mutta tämän osalta esitettiin myös negatiivisia kokemuksia. Kielentämisen selkeydestä, avusta, helppoudesta ja mukavuudesta oltiin jokseenkin eri mieltä, mutta vaihtelu oli osallistujilla suurta.

8.8 Millaisista tehtävistä voisi olla hyötyä algoritmisen ajattelun kehittämisessä?

Kirjallisuuden perusteella algoritmiseen ajatteluun kuuluu paljon eri sisältöjä, joita voidaan opetella monilla tavoilla. Algoritmiseen ajatteluun liitetään kehittymistä kuvaavia taipumuksia, kuten sinnikkyys, yhteistyötaidot ja virheiden sekä monimutkaisuuden sietäminen. Tämän tutkimuksen tehtävien ratkaisuista havaittiin, että osa oli luovuttanut tehtävän ratkaisemisen nopeasti, eikä kunnon yritystä oltu saatu aikaan.

Menetelmiksi on ehdotettu muun muassa CTL:n käyttämistä ja prosessiluonteisuutta (esimerkiksi ohjelmointia kokonaisuutena pelkän koodauksen sijaan). Algoritminen ajattelu olisi myös liitettävä oppiaineisiin irrallisuuden sijaan ja näytettävä laskennan osuus eri aloilla. Visuaalisuuden merkitystä korostetaan oppimisessa. Menetelmissä ja materiaaleissa on otettava huomioon oppilaiden taustat, kuten sukupuoli ja sosiokulttuurisuus. Oppilaiden on hyvä saada roolimalleja eri taustan ammattilaisista, joiden kanssa opetuksessa tehdään yhteistyötä myös opettajien tietojen ja taitojen kehittämiseksi. Tämän opetuskokeilun ratkaisuista havaittiin, että oppilaat olivat hyvin eri tasoisia algoritmisessa ajattelussa, ohjelmoinnissa ja kielentämisessä. Näin ollen eriyttämistä tarvitaan. Tä-

hän voisi auttaa eri ohjelmointikielten käyttäminen, visuaalisuus ja erilaisten tehtävätasojen laatiminen.

Algoritmisen ajattelun opetuksen välineistä mainitaan esimerkiksi ohjelmointi, pelisuunnittelu, robotit, simulaatiot, ongelmanratkaisu ja paperilla (tai sanallisesti) toteutettavat algoritmitehtävät. Ohjelmointiin tukeuduttaessa käytetään usein pidempiaikaista projektia yksittäisten tehtävien sijaan. Luovuutta ja tuotteiden luomista suositaan, sillä oppilaiden halutaan kehittävän menetelmiä ja välineitä pelkän käytön sijaan. Tarjolla on myös valmiita tehtäviä erilaisissa ohjelmointikerhoissa ja -sivustoilla. Välineinä voidaan käyttää monia tietoteknisiä laitteita (esimerkiksi tietokone, kännykät, e-tekstiilit) tai suorittaa tehtäviä ilman laitteita. Tärkeää on, että tehtävät tukevat monipuolisesti algoritmisen ajattelun osa-alueiden oppimista ja tehtävien kautta saavutetaan myös algoritmisen ajattelun taipumuksia. Opetuksen kannalta olisi hyvä käyttää tehtäviä, jotka ovat arvioitavissa, mutta algoritmisen ajattelun arviointi keskittyy usein yksittäiseen ohjelmointikieleen tai ympäristöön.

Tämän tutkielman opetuskokeilu osoittaa, että ohjelmointia voitaisiin ehkä hyödyntää muissakin matematiikan aihealueissa kuin aritmetiikassa. Algoritmista ajattelua voitaisiin todennäköisesti kehittää myös yksittäisillä ohjelmointi- ja kielentämistehtävillä eri oppiaineiden ja erityisesti matematiikan aihealueissa, kunhan perusohjelmointitaito olisi hankittu. Kielentäminen voisi sopia algoritmisen ajattelun kehittämiseen ja ohjelmoinnin ja algoritmisen ajattelun arvioimiseen.

9 POHDINTA

Tässä tutkielmassa haluttiin selvittää perusopetuksen opetussuunnitelmassa 2014 esiintyvän algoritmisen ajattelun tilaa Suomessa, koska tätä ei ole aiemmin juuri tutkittu. Tutkielmassa selvitettiin algoritmisen ajattelun sisältöä ja opetuksen tilaa kansainvälisesti, sekä tutkittiin algoritmista ajattelua peruskoululaisilla ja lukiolaisilla. Tutkimus toteutettiin kirjallisen kielentämisen avulla geometriaa sisältävillä ohjelmointitehtävillä ja kyselylomakkeella. Tarkoituksena on kehittää kirjallisuuden ja tutkimuksen tulosten perusteella jatkossa materiaalia opettajille ja oppilaille algoritmisen ajattelun kehittämiseksi.

Algoritmisen ajattelun osaaminen oppilailla oli keskimäärin 39,2 %, mutta tasoerot oppilaiden välillä olivat suuria. Matematiikan korkeammalla arvosanalla sekä korkeammalla koulutustasolla ja vuosiluokalla havaittiin olevan merkitsevä yhteys parempaan suoriutumiseen algoritmisessa ajattelussa. Sukupuolella tai aiemmalla ohjelmointikokemuksella ei havaittu olevan merkitsevyyttä suoriutumisessa. Tutkituista algoritmisen ajattelun kategorioista keskimäärin heikoimmin oppilaat suoriutuivat kaavoista ja malleista, virheiden korjaamisesta sekä suunnittelusta. Korkeimmat kategoriapisteet havaittiin loogisessa päättelyssä.

Kielentäminen vaikutti tuovan ohjelmoinnissa algoritmisen ajattelun esille suhteellisen hyvin. Kirjallinen kielentäminen teki oppilaiden algoritmisen ajattelun prosessia näkyväksi, myös hankalasti todennettavissa osa-alueissa. Lisäksi saatiin tietoa algoritmisen ajattelun haasteista ja vahvuuksista syineen. Kielentämisen ja dokumentoinnin havaittiin tuovan ohjelmointiin paljon vastaavaa palautetta. Kielentämällä saatiin lisätietoa ohjelmoinnin oivaltamisesta sekä ongelmista, esimerkiksi väärinymmärryksistä, huonoista ohjelmointikäytännöistä, termien käytöstä, ohjelmointikoodin rakenteiden ja syntaksin virheistä, mitkä muuten olisivat saattaneet jäädä huomaamatta. Kielentäminen näyttäisi tuovan oppimisen näkyväksi ohjelmoinnissa.

Oppilaiden kokemukset ohjelmoinnista ja kielentämisestä jakautuivat vahvasti. Ohjelmoinnin helppoudesta, hyödyllisyydestä ja mukavuudesta oltiin yleensä jokseenkin eri mieltä, mutta erot kokemuksissa olivat suuria. Osa oppilaista koki ohjelmoinnin myös kivaksi ja helpoksi. Syyksi ohjelmoinnin kielteisiin kokemuksiin osa ilmaisi sen, ettei ohjelmointia ollut vielä opetettu. Osa koki, että matematiikan ja kaavojen osaamisella on yhteys ohjelmoinnin osaamiseen. Mielipiteet jakautuivat kahtia myös paperitoteutuksen suhteen. Kielentämisen selkeydestä, avusta, helppoudesta ja mukavuudesta oltiin jokseenkin eri mieltä, mutta vaihtelu oli osallistujilla suurta. Kielentämisen koettiin selventävän ratkaisuja ja matematiikan ymmärtämistä sekä lisäävän ajattelua ratkaisua laadittaessa. Suuri osa koki kielentämisen olevan hyödyllinen ja kiva menetelmä, mutta toisaalta kielentämisen koettiin vievän aikaa, ja siihen toivottiin opetusta. Oppilaat kokivat käyttäneensä tehtävissä loogista ajattelua, yksinkertaistusta ja aiemmin opittuja matemaattisia kaavoja, ja lisäksi kuvien piirtäminen perusteluna koettiin helpoksi. Oppilaat olivat tekemistään ratkaisuista ja ymmärtämisestään usein epävarmoja.

Algoritmisen ajattelun kehittämisessä hyötyä näyttäisi kirjallisuuden perusteella olevan tehtävistä, jotka suosivat algoritmisen ajattelun taipumusten ja sisältöjen kehittymistä, tukevat käsitteiden käyttöä sekä CT:n opetuksen luokkahuonekulttuurin ominaisuuksia. Algoritminen ajattelu ja laskenta tulisi integroida oppiaineisiin tai aloihin luontevasti. CT:n opetuksen välineistä mainitaan esimerkiksi ohjelmointi, pelisuunnittelu, robotit, simulaatiot, ongelmanratkaisu ja ilman tietokonetta toteutettavat algoritmitehtävät. Visuaalisuus voi tukea oppimista. Luovuutta, tuotteiden luomista ja pidempiaikaisia monipuolisia projekteja suositaan. Menetelmissä ja materiaaleissa on otettava huomioon oppilaiden taustat ja kannustettava kaikkia sukupuoleen ja sosiokulttuurisuuteen katsomatta. Tehtävillä tulisi olla myös algoritmisen ajattelun sisältöjen osaamisen arviointikriteerit. Oppilaat olivat keskenään hyvin eri tasoisia algoritmisessa ajattelussa, ohjelmoinnissa ja kielentämisessä, joten tehtävissä saatetaan tarvita myös eriyttämistä.

9.1 Tulosten tarkastelu

Osallistujien aiemmin käyttämien ohjelmointikielien osalta vaikuttaisi siltä, että Pythonin käyttö tutkimuksessa oli sopiva valinta, koska Python oli käytetyin kieli, ja perusopetuksen opetussuunnitelma (Opetushallitus, 2016) kehottaa oikean ohjelmointikielen käyttöön yläkouluikäisillä. Lisäksi Python ja Pythoniin perustuva VPython on mainittu helposti omaksuttaviksi ilman aiempaa ohjelmointikokemusta (Ahamed ym., 2010; Flórez ym., 2017). Toisaalta CT:n tutkimuksia on tehty pääasiassa Scratchillä ongelmanratkaisumenetelmällä (Hsu, Chang, & Hung, 2018). Tutkimuksen tehtävät erottelivat osaamista, koska hajonta koko aineistossa ja eri vuosiluokilla oli suurta. Oppilaiden kouluvuosilla havaittiin olevan merkitsevyys algoritmisessa ajattelussa suoriutumiseen (vastaavasti Román-González, Pérez-González, & Moreno-León, 2018).

Vastaavasti kuin Papastergioun (2008, 601–602) ja Kaarakaisen ym. (2013) tutkimuksessa, myös tämän tutkimuksen aineistosta nousi esiin, että tytöt harrastavat ohjelmointia ja tietotekniikkaa poikia vähemmän. Toisaalta tehtävien suoriutumisessa ei havaittu eroja sukupuolten välillä (tämä vastaa Lau & Yuen, 2009) tai aiemman ohjelmointikokemuksen välillä ainakaan tutkimuksessa käytetyllä aineistokoolla. Aiemmin ohjelmoineiden määrä oli aineistossa melko pieni. Tämä saattaa tosin johtua siitä, että kouluissa ei ole aikaa ohjelmoinnille, koska opetussuunnitelman matematiikan sisältö on määrällisesti pysynyt muuttumattomana. Myös Niemelä (2018) esittää väitöstutkimuksessaan, että samalla kun jotain lisätään opetussuunnitelmaan, olisi sieltä myös karsittava jotain pois.

Mielipiteet jakautuivat ohjelmoinnin ja kielentämisen suhteen. Moni oppilas koki ohjelmointitehtävät haastaviksi. Ohjelmointi onkin usein haastavaa aloittelijoille (esim Lau & Yuen, 2009, 697). Kielentäminen koettiin aikaa vieväksi, mutta siitä koettiin olevan hyötyä sekä matematiikassa että ratkaisun tarkemmassa läpikäynnissä (vastaavasti Sarikka, 2014, 51). Oppilaat kokivat myös epävarmuutta ymmärtämisestään ja tehtävien ratkaisuistaan. Oppilaan käsityksillä itsestä on havaittu olevan vaikutusta matematiikan oppimiseen (Joutsenlahti, 2005, 51–52) ja pystyvyys on mainittu myös muutamissa tietojenkäsittelytiedettä käsittelevässä julkaisussa (esim. Papastergiou, 2008, 600–604; Pioro, 2004, 14, 22). Tutkimuksessa oli vaikea verrata Selbyn (2014) kuvaamia Bloomin tak-

sonomian, CT:n ja ohjelmoinnin hierarkioiden haastavuusjärjestyksiä, koska tutkittavilta ei noussut esiin tiettyjä haastavia teemoja.

Näyttäisi siltä, että kehittämistutkimuksella toteutettavissa materiaaleissa pitäisi huomioida selkeä kompakti johdatus algoritmiseen ajatteluun ja ohjelmointiin. Materiaaleissa voisi käyttää toisiinsa yhteydessä olevia versioita opettajille ja oppilaille. Opettajien versiossa tulisi olla kuvaus ydinsisällöistä ja opetuskäytännöistä, koska opettajilla on todennäköisesti monenlaisia käsityksiä siitä, mitä ovat algoritminen ajattelu ja ohjelmointi, ja miten opetussuunnitelma edellyttää niitä opetettavan. Haasteiksi voivat muodostua myös useat eri ohjelmointikielet (tuotoksen eli materiaalin olisi oltava yleistettävissä), joten materiaalin tulee olla käytettävissä useammalle kuin yhdelle eri kielelle.

Materiaalin pitäisi ottaa huomioon myös eriyttäminen. Osalle oppilaista oli epäselvää, miten ratkaisuista voi kertoa luonnollisella kielellä, ja osa ei tiennyt, mitä ohjelmointi tarkoittaa. Osassa tutkimuksen ratkaisuista ei havaittu sinnikästä yrittämistä, mutta voi olla, että ohjelmointikoodi on uutena englanninkielisenä asiana toisille oppilaille hyvin haastavaa. Kolmiportainen ohjelmointitehtävien tasorakenne näyttäisi olevan tarpeellinen, koska osa oppilaista tarvitsisi lyhyitä johdatustehtäviä, osalle tehtävät olivat sopivan tasoisia ja osa oppilaista olisi tarvinnut haastetta enemmän. Johdattelussa ohjelmointiin apuna voidaan käyttää ohjelmointikielten rinnalla ohjelmointiympäristöjä, jotta ohjelmoinnin perusteet ymmärretään paremmin (esim. Dann ym., 2012). Yksittäisten tehtävien lisäksi algoritmista ajattelua ja ohjelmointia voidaan yrittää kehittää esimerkiksi ohjelmistotuotannon ketterien menetelmien (Fronza, Ioini, & Corral, 2017), projektien ja pelillisyyden avulla. Nämäkin voidaan integroida luontevasti eri oppiaineisiin.

Tämän tutkielman geometriaan perustuvien tehtävien mukaan vaikuttaisi siltä, että ohjelmointia voitaisiin ehkä hyödyntää muissakin matematiikan aihealueissa kuin aritmetiikassa. Geometria toi lisäarvoa tuloksiin, koska ohjelmoinnista paljastui virhekäsityksiä. Algoritmista ajattelua voitaisiin todennäköisesti kehittää myös yksittäisillä ohjelmointi- ja kielentämistehtävillä eri oppiaineiden ja erityisesti matematiikan aihealueissa, kunhan perusohjelmointitaito olisi hankittu. Tällöin ohjelmointi jakautuisi tasaisemmin eri matematiikan kursseille ja ohjelmointitaitoja pystyttäisiin pitämään yllä. Toisaalta Niemelä (2018) ehdottaa

väitöstutkimuksessaan opetussuunnitelmissa lisäämään ohjelmointitaitojen kehittämistä varten joukko-oppia, logiikkaa ja algoritmimatematiikkaa.

Algoritmisen ajattelun opettamisen ja arvioinnin menetelmät muuttuvat määritelmäsuuntausten mukana, mikä aiheuttaa opetukseen ongelmia (Denning, 2017b). Tulosten perusteella näyttäisi siltä, että kielentäminen voisi sopia algoritmisen ajattelun kehittämiseen, koska se toi näkyviin CT:n osa-alueita ratkaisuissa. Lisäksi algoritmisen ajattelun esille tuomisen seurauksena on mahdollista, että kielentäminen voisi sopia ohjelmoinnin ja algoritmisen ajattelun (oikeudenmukaiseen) arvioimiseen. Kirjallinen kielentäminen auttaisi opettajaa havaitsemaan oppimista myös passiivisilla oppilailla (vrt. Laitinen ym., 2015; Silfverberg ym., 2005). Arviointikriteerit voitaisiin muodostaa CT:n sisältöjen perusteella, ja kirjallisten kielentämistuotosten perusteella voitaisiin arvioida CT:tä kriteerejä käyttäen. Kielentäminen arviointimenetelmänä voisi olla myös yleistettävissä käytettyjen tehtävätyyppien, välineiden ja ympäristöjen sekä ohjelmointikielten ylitse. Tutkimuksen tehtävien ratkaisuissa tyyli vaihteli tehtävittäin ja yksilöittäin, eikä yleistä kielentämisstrategiaa näin ollen havaittu (vrt. Sarikka, 2014).

9.2 Validiteetti ja reliabiliteetti

Algoritmiselle ajattelulle ei ole olemassa yksikäsitteistä määritelmää, ja sisältöjäkin on paljon. Tutkimuksessa jäi epäselväksi, onko käytetty CT:n tulkinta riittävän samankaltainen kuin valtaosalla vertailututkimuksista. Algoritmisen ajattelun laajuuden ja käytettävissä olevan ajan vuoksi siitä mitattiin vain osia. Ei ole varmaa, olivatko opetuskokeiluun laaditut tehtävät onnistuneet suhteessa CT:n osa-alueisiin, ja tehtävien rajallisen laajuuden vuoksi osaa kategorioista ei saatu kattavasti testattua. Standardoitua mittausmenetelmää CT:n tutkimiseen ei oltu vielä kehitetty, joten CT:n mittauksen sopivuudesta ei voitu olla varmoja. CT:lle on laadittu heikosti yleistettävissä olevia yksittäisiä arviointimenetelmiä, mutta ne eivät sopineet tähän tutkimukseen.

Tutkimukseen oli haastavaa laatia sopivan tasoisia tehtäviä, koska kirjallisuudessa on eroja sisällöissä ja niiden tasoissa, ja lisäksi koulujärjestelmät ovat kansainvälisesti erilaisia oppilaiden ikävuodet ja opetussuunnitelmat huomioiden. Lisäksi algoritmisen ajattelun käsitteistä ja sisällöistä oli hankala muodostaa kompakteja kysymyksiä. Algoritmisen ajattelun kyselyosuus voitaisiin jatkos-

sa tehdä validoiduin kysymyksin (Yagci, 2019). Tutkimuksessa haluttiin toteuttaa tehtävät paperilla. Burtonin (2010) kuvailemiin kynäpaperitehtäviin verrattuna tässä tutkimuksessa oikean lopputuloksen sijaan esille haluttiin oppilaiden ajatteluprosessi tehtäviä ratkaistaessa, joten avoimet vastaukset koettiin monivalintoja paremmiksi. Toisaalta voidaan pohtia, johtuiko ohjelmointikoodin ymmärtämisen vaikeus osittain paperisesta toteutuksesta.

Tehtävien arvostelukategoriat laadittiin yhdistäen eri CT:n määritelmiä. Epäselvää kuitenkin oli, olivatko kategoriat hyviä suhteessa tehtäviin ja oliko kategorioita osattu käyttää arvioinnissa oikein. Myös kategorioiden keskinäisten pistesuhteiden erilainen määrittely ja painotus kokonaispisteytyksessä saattoivat vaikuttaa tulkintaan. Oppilaat laativat tehtäviin avoimet ratkaisut, joten objektiivisuuden tavoitteesta huolimatta ratkaisujen analysoinnissa tehtiin tulkintaa. Yhden tulkitsijan tapauksessa arvioinnit olivat kuitenkin keskenään vertailukelpoisia ja objektiivisuutta haettiin taulukoitujen kriteerien avulla.

Kyselyosuuden ja tehtävien perusteella osa oppilaista ei ymmärtänyt tehtävänantoja, väitekysymyksiä tai kielentämistä. Voi myös olla, etteivät tutkittavat ymmärtäneet tehtäviä keskenään samalla tavalla. Toistettavuuden vuoksi tutkimuksen osana on esitetty tehtävät ja niiden arviointiperusteet taulukoituna. Tehtävissä oli ohjelmoinnin ja matematiikan kontekstia, joten näiden aiheiden tuntemus saattoi vaikuttaa algoritmisen ajattelun tuloksiin tässä tutkimusasetelmassa. Wernerin ym. (2012, 219) mukaan algoritmisen ajattelun arvioinnissa ei saa testata ohjelmoinnin syntaksia vaan CT:n osa-alueita ohjelmointitehtävien kautta. Matemaattisten teemojen käyttäminen johti myös siihen, että matematiikan osaaminen vaikutti kokonaispisteisiin ja eri aihealueiden pisteisiin, vaikka tehtäviä pyrittiin suunnittelemaan niin, että suuri osa tehtävistä onnistuisi myös ilman etevää matemaattista osaamista. Toisaalta Burtonin (2010, 4–6, 11) mukaan algoritmisen ajattelun tehtävien luominen on vaikeaa niin, etteivät tehtävät ole sidoksissa ohjelmointiosaamiseen ja tavanomaisiin matematiikan ongelmiin.

Osa oppilaista saattoi tehdä parityötä itsenäisen työskentelyn sijaan, jolloin kyseinen vastaus ei kuvaa yksilön osaamista ja vinouttaa tuloksia. Koulujen väliset erot saattoivat myös vaikuttaa tutkimuksen tuloksiin.

9.3 Johtopäätökset

Algoritmisen ajattelun määritelmien, käsitteiden ja sisältöjen moninaisuuden sekä yhteisen konsensuksen puuttumisen vuoksi opetuksen kannalta tärkeitä opetusmenetelmiä ja CT:n arviointimenetelmiä ollaan vasta tutkimassa ja kehittämässä. Lisäksi algoritmisen ajattelun ja ohjelmoinnin opetussuunnitelmasisällöissä on huomattavia eroja kansainvälisesti. Yhteisen kriteeristön puuttuminen aiheuttaa ongelmia algoritmisen ajattelun opettamiseen ja kehittämiseen. Suomen opetussuunnitelmissa tulisi keskittyä ohjelmoinnin lisäksi algoritmiseen ajatteluun, ja tarkentaa sen sisältöjä. Opetussuunnitelma on vasta jalkautunut yläkoulujen opetukseen, ja lukion opetussuunnitelmassa ohjelmointia ei ole edes mainittu. Suomen opetussuunnitelmat näyttävätkin yhteiskunnallisiin tavoitteisiin, kansainvälisiin tutkimuksiin ja opetussuunnitelmalinjauksiin verrattuna vähän vanhentuneilta. CT vaatii standardien luomista ja kehittämistä. Jatkossa tarvitaan laajoja tutkimuksia algoritmisen ajattelun opetuksesta Suomessa ja ulkomailla.

Algoritmisen ajattelun osaamisessa on paljon kehitettävää ja tasoerot ovat suuret. Olisi tärkeää selvittää, mitkä tekijät aiheuttavat oppilaiden osaamisen erot. Tutkimuksessa havaittiin usein oppilaiden epävarmuus, joten jatkossa tarvittaisiin tutkimusta liittyen oppilaiden minäpystyvyyden yhteyteen algoritmisessa ajattelussa, matemaattisessa ajattelussa (kielentämisessä) ja ohjelmoinnissa. Lisäksi tarvitaan jatkotutkimuksia selvittämään osaavatko oppilaat käyttää laskennallisia välineitä hyödyllisesti ja luoda itse erilaisia tuotoksia algoritmisen ajattelun ja ohjelmoinnin avulla. Pelkkä TVT:n opetus ja oppiminen eivät riitä 2000-luvun taitojen kartuttamiseen.

Algoritmisen ajattelun kenttä on laaja, esimerkiksi algoritmisen ajattelun arvioinnin ja kehittämisen tutkimusta pitäisi jatkaa. Tulevaisuudessa olisi tärkeää keskittyä myös pidempiin ja monipuolisempiin algoritmisen ajattelun projekteihin, joissa useammat algoritmisen ajattelun osa-alueet olisivat mitattavina. Tällöin saataisiin enemmän tietoa kielentämisen sopivuudesta algoritmisen ajattelun kehittämiseen ja arviointimenetelmäksi. Kielentämisen sopivuudesta ohjelmoinnin ja algoritmisen ajattelun opetukseen ja arviointiin tarvittaisiin lisätutkimusta.

Lisäksi opettajille tarvittaisiin koulutusta algoritmisen ajattelun käsitteen ymmärtämiseen, välineiden hyödyntämiseen ja algoritmisen ajattelun opettamiseen luontevasti sekä omissa oppiaineissa että oppiainerajat ylittäen. Opettajien käsityksiä ja opetustapoja tulisi myös tutkia laajemmin. Miten opettajat saavat esimerkiksi algoritmisen ajattelun ja ohjelmoinnin sisällytettyä oppiaineisiin nykyisin tuntiresurssein? Kiinnostavaa olisi myös tietää, opetetaanko ohjelmointia vain tietyssä vaiheessa lukukautta (esimerkiksi loppukeväällä, kun matematiikan sisältö on saatu käsiteltyä) vai opetetaanko ohjelmointia tasaisemmin lukuvuoden ajan. Jatkossa olisi hyvä tutkia, millaista algoritmisen ajattelun opetusta opettajat antavat ja mitä menetelmiä sekä välineitä he käyttävät. Koska opetussuunnitelmassa puhutaan käsitteistä TVT, algoritminen ajattelu ja ohjelmointi, olisi hyvä tutkia opettajien käsityksiä näiden sisällöistä ja opettajien käyttämistä opetusmenetelmistä. Lisäksi olisi tarpeellista tutkia, miten algoritminen ajattelu ja ohjelmointi on otettu opetukseen mukaan, ovatko ne matematiikasta ja muista oppiaineista irrallinen kokonaisuus vai onko tehty integrointia.

Vaikka algoritminen ajattelu on sisällytetty opetussuunnitelmaan, tarvittaisiin vielä tarkempaa ohjeistusta opettajille, jotta opetusta ja oppimista saataisiin laajennettua syvällisempiin ajattelun prosesseihin. Ohjelmointiprosessit ja erityisesti algoritminen ajattelu olisi nostettava esille pelkän koodauksen sijaan. Tutkimuksen tulosten perusteella algoritmisen ajattelun ja ohjelmoinnin osaaminen on keskimäärin heikkoa, kuten on myös aikaisemmista tutkimuksista saatu viitteitä. Näitä taitoja olisi tärkeä edistää, jotta 2000-luvun taidot kehittyisivät ja saataisiin riittävästi osaavaa työvoimaa, sillä teknologiaa tarvitaan jokaisella alalla ja teknologisen maailman ymmärtämistä vaaditaan päivittäin.

10 LÄHTEET

Agarwal, R. P., Agarwal, H., & Sen, S. K. (2013). Birth, growth and computation of pi to ten trillion digits. *Advances in Difference Equations*, *2013*(1), 1–59.

Ahamed, S. I., Brylow, D., Ge, R., Madiraju, P., Merrill, S. J., Struble, C. A., & Early, J. P. (2010). Computational Thinking for the Sciences. A Three Day Workshop for High School Science Teachers. *ACM. Proceedings of the 41st ACM technical symposium on Computer science education - SIGCSE '10*, 42–46.

Aho, A. V. (2011). Ubiquity symposium: Computation and Computational Thinking. *Ubiquity*, *2011*(January), 8.

Aiken, J. M., Caballero, M. D., Douglas, S. S., Burk, J. B., Scanlon, E. M., Thoms, B. D., & Schatz, M. F. (2013). Understanding Student Computational Thinking with Computational Modeling. *AIP Conference Proceedings*, 46–49.

Allan, V., Barr, V., Brylow, D., & Hambrusch, S. (2010). Computational thinking in high school courses. *ACM. Proceedings of the 41st ACM technical symposium on Computer science education - SIGCSE '10*, 390–391.

Ambrósio, A. P., Xavier, C., & Georges, F. (2014). *Digital Ink for Cognitive Assessment of Computational Thinking*. IEEE. 2014 IEEE Frontiers in Education Conference (FIE) Proceedings.

Ambrus, A. (1998). Use of visual representations in mathematics education. Teoksessa S. Kaartinen (Toim.), *Matemaattisten aineiden opetus ja oppiminen. Matematiikan ja luonnontieteiden opetuksen tutkimusseura*

ry:n päivillä Oulussa 18.-20.10.1996 pidetyt esitelmät ja alustukset. Oulun yliopiston kasvatustieteiden tiedekunnan opetusmonisteita ja selosteita 78/1998. Oulun yliopiston kasvatustieteiden tiedekunta, Oulu. Sivuja 212. (ss. 75–81). Oulu: Oulun yliopistopaino.

Amiel, T., & Reeves, T. C. (2008). Design-Based Research and Educational Technology: Rethinking Technology and the Research Agenda. *Educational Technology & Society*, *11*(4), 29–40.

Anderson, T., & Shattuck, J. (2012). Design-Based Research: A Decade of Progress in Education Research? *Educational Researcher*, *41*(1), 16–25.

Angeli, C., Voogt, J., Fluck, A., Webb, M., Cox, M., Malyn-Smith, J., & Zagami, J. (2016). A K-6 Computational Thinking Curriculum Framework: Implications for Teacher Knowledge. *Journal of Educational Technology & Society; Palmerston North*, *19*(3), 47–57.

Association for Computing Machinery (ACM), & IEEE Computer Society. (2013). *Computer Science Curricula 2013. Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. 514.

Balanskat, A., & Engelhardt, K. (2015). *Computing our future. Computer programming and coding Priorities, school curricula and initiatives across Europe* (s. 87). Belgium: European Schoolnet (EUN Partnership AIBSL).

Barr, D., Harrison, J., & Conery, L. (2011). Computational Thinking: A Digital Age Skill for Everyone. *Learning & Leading with Technology*, *38*(6), 20–23.

Barr, V., & Stephenson, C. (2011). Bringing computational thinking to K-12: What is involved and what is the role of the Computer Science Education Community? *ACM Inroads*, *2*(1), 48–54.

Basawapatna, A., Koh, K. H., & Repenning, A. (2010). Using scalable game design to teach computer science from middle school to graduate school. *Proceedings of the fifteenth annual conference on Innovation and technology in computer science education - ITiCSE '10*, 224–228.

Basawapatna, A., Koh, K. H., Repenning, A., Webb, D. C., & Marshall, K. S. (2011). Recognizing computational thinking patterns. *Proceedings of the 42nd ACM technical symposium on Computer science education - SIGCSE '11*, 245–250.

Basogain, X., Olabe, M. A., Olabe, J. C., Ramírez, R., Rosario, M. D., & Garcia, J. (2016). *PC-01: Introduction to computational thinking: Educational technology in primary and secondary education*. 2016 International Symposium on Computers in Education (SIIE).

Basu, S., Kinnebrew, J. S., & Biswas, G. (2014). Assessing Student Performance in a Computational- Thinking Based Science Learning Environment. Teoksessa S. Trausan-Matu, K. E. Boyer, M. Crosby, & K. Panourgia (Toim.), *Intelligent Tutoring Systems 12th International Conference, ITS 2014 Honolulu, HI, USA, June 5-9, 2014 Proceedings* (ss. 476–481). Cham Heidelberg New York Dordrecht London: Springer.

Bell, T., Alexander, J., Freeman, I., & Grimley, M. (2009). Computer Science Unplugged: school students doing real computing without computers. *The New Zealand Journal of Applied Computing and Information Technology*, *13*(1), 20–29.

Bell, T., Witten, I. H., & Fellows, M. (2015). *CS Unplugged. An enrichment and extension programme for primary-aged students*.

Berkeley, I. S. N. (2018). A Computational Conundrum: "What is a Computer?" A Historical Overview. *Minds and Machines*, *28*(3), 375–383.

Bers, M. U., Flannery, L., Kazakoff, E. R., & Sullivan, A. (2014). Computational thinking and tinkering: Exploration of an early childhood robotics curriculum. *Computers & Education*, *72*, 145–157.

Bloom, B., Engelhardt, M., Furst, E., Hill, W., & Krathwohl, D. (1956). *Taxonomy of Educational Objectives: The Classification of Educational Goals, Handbook 1 Cognitive Domain*. New York: McKay.

Bocconi, S., Chioccariello, A., Dettori, G., Ferrari, A., & Engelhardt, K. (2016). *Developing computational thinking in compulsory education – Implications for policy and practice*. 68.

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. *American Educational Research Association meeting, Vancouver, BC, Canada*, 25.

Brown, A. L. (1992). Design Experiments: Theoretical and Methodological Challenges in Creating Complex Interventions in Classroom Settings. *Journal of the Learning Sciences*, *2*(2), 141–178.

Brown, N. C. C., Kölling, M., Crick, T., Jones, S. P., Humphreys, S., & Sentance, S. (2013). Bringing computer science back into schools: Lessons from the UK. *ACM. Proceeding of the 44th ACM technical symposium on Computer science education - SIGCSE '13*, 269–274.

Burton, B. (2010). Encouraging algorithmic thinking without a computer. *Olympiads in Informatics, 2010. Institute of Mathematics and Informatics, Vilnius*, *4*, 3–14.

CEPIS (Council of European Professional Informatics Societies). (a2014). *e-Competence in Europe. Analysing Gaps and Mismatches for a Stronger ITC Profession. European report*. Noudettu osoitteesta http://www.cepis.org/media/DIGITALJOBS_Deliverable_4.3_EU1.pdf

CEPIS (Council of European Professional Informatics Societies). (b2014). *e-Competence in Europe. Analysing Gaps and Mismatches for a Stronger ITC Profession. Executive Summary* [Executive Summary]. Noudettu osoitteesta http://www.cepis.org/media/CEB_Summary_Brochure_2014_v1.01.pdf (Viitattu 2.4.2018)

CoderDojo Foundation. (ei pvm.). *CoderDojo*. Noudettu osoitteesta https://coderdojo.com/

Committee for the Workshops on Computational Thinking, & National Research Council. (2010). *Report of a Workshop on the Scope and Nature of Computational Thinking*. National Academy Press.

Computational. (2019). Teoksessa *MOT Pro Englanti*. Kielikone Oy & Gummerus Kustannus Oy.

Computer Science Education Research Group. (ei pvm.). *CS Unplugged*. Noudettu osoitteesta https://csunplugged.org/en/

Cooper, S., Pérez, L., & Rainey, D. (2010). K--12 Computational Learning. *Communications of the ACM, ACM*, *53*(11), 27–29.

Crossley, J. N., & Henry, A. S. (1990). Thus spake al-Khwārizmī: A translation of the text of Cambridge University Library Ms. Ii.vi.5. *Historia Mathematica*, *17*(2), 103–131.

Crow, D. (7.2.2014). Why every child should learn to code. Noudettu 17. toukokuuta 2018, osoitteesta The Guardian website: https://www.theguardian.com/technology/2014/feb/07/year-of-code-dan-crow-songkick (Viitattu 17.5.2018)

Csizmadia, A., Curzon, P., Dorling, M., Humphreys, S., Ng, T., Selby, C., & Woollard, J. (2015). *CAS computational thinking - A Guide for teachers*. Hodder Education, Digital Schoolhouse, Computing At School, 2015.

CSTA, & ISTE. (2011). *Computational Thinking in K–12 Education. Teacher resources.*

Curzon, P., McOwan, P. W., Cutts, Q. I., & Bell, T. (2009). Enthusing & Inspiring with Reusable Kinaesthetic Activities. *Proceedings of the 14th Annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education*, 94–98. Noudettu osoitteesta http://doi.acm.org/10.1145/1562877.1562911 (Viitattu 6.4.2019)

Curzon, P., Peckham, J., Taylor, H., Settle, A., & Roberts, E. (2009). Computational Thinking (CT): On Weaving It In. *ACM SIGCSE Bulletin - ITiCSE '09*, *41*(3), 201–202.

Dann, W., Cosgrove, D., Slater, D., Culyba, D., & Cooper, S. (2012). Mediated transfer: Alice 3 to Java. *Proceedings of the 43rd ACM technical symposium on Computer Science Education - SIGCSE '12*, 141–146.

Dede, C., Mishra, P., & Voogt, J. (2013). Working Group 6: Advancing computational thinking in 21st century learning. *International summit on ICT in education. EDU Summit 2013*, 1–6.

Denning, P. J. (2009). The profession of IT. Beyond computational thinking. *Communications of the ACM*, *52*(6), 28–30.

Denning, P. J. (a2017). Computational Thinking in Science. *American Scientist*, *105*(1), 13–17.

Denning, P. J. (b2017). Remaining Trouble Spots with Computational Thinking. *Communications of the ACM*, *60*(6), 33–39.

Dr. Scratch: drscratch.org. (2014). Noudettu osoitteesta http://www.drscratch.org/ (Viitattu 18.7.2018)

Edelson, D. C. (2002). Design Research: What We Learn When We Engage in Design. *Journal of the Learning Sciences*, *11*(1), 105–121.

Eskola, J., & Suoranta, J. (1998). *Johdatus laadulliseen tutkimukseen* (7. painos). Jyväskylä: Vastapaino.

Espino, E. E., & González, C. (2016). Gender and Computational Thinking: Review of the literature and applications. *ACM*, 1–2.

Espino, E. E., & González, C. S. (2015). Influence of Gender on Computational Thinking. *ACM*, 1–2.

Europe Code Week. (ei pvm.). *European CodeWeek*. Noudettu osoitteesta http://codeweek.eu/ (Viitattu 7.6.2018)

European Commission. (2014). *e-SKILLS: THE INTERNATIONAL DIMENSION AND THE IMPACT OF GLOBALISATION. Final Report*. Noudettu osoitteesta http://www.cepis.org/media/FINAL_INTERNATIONAL_e_Skills_report_Aug_141.pdf (Viitattu 29.3.2018)

Fant, K. (1993). *A Critical review of the Notion of the Algorithm in Computer Science*. Proceedings of the 21st Annual Computer Science Conference.

Flórez, F. B., Casallas, R., Hernández, M., Reyes, A., Restrepo, S., & Danies, G. (2017). Changing a Generation's Way of Thinking: Teaching Computational Thinking Through Programming. *Review of Educational Research*, *87*(4), 834–860.

Folkerts, M. (2001). Early Texts on Hindu-Arabic Calculation. *Science in Context*, *14*(1–2), 13–38.

Fronza, I., Ioini, N. E., & Corral, L. (2017). Teaching Computational Thinking Using Agile Software Engineering Methods. *ACM Transactions on Computing Education*, *17*(4), 1–28.

Gander, W., Petit, A., Berry, G., Demo, B., Vahrenhold, J., McGettrick, A., … Meyer, B. (2013). Informatics Education: Europe Cannot Afford to Miss the Boat. *Informatics Europe & ACM Europe Working Group on Informatics Education*, 1–21.

Ganley, C. M., & Vasilyeva, M. (2014). The role of anxiety and working memory in gender differences in mathematics. *Journal of Educational Psychology*, *106*(1), 105–120.

García-Peñalvo, F. J., & Mendes, A. J. (2017). Exploring the computational thinking effects in pre-university education. *Computers in Human Behavior 80 (2018)*, 407–411.

García-Peñalvo, F. J., Reimann, D., Tuul, M., Rees, A., & Jormanainen, I. (2016). An overview of the most relevant literature on coding and computational thinking with emphasis on the relevant issues for teachers. *Belgium. TACCLE 3 Consortium*, 72.

Ghione, F. (2016). The algebra section of the Museum on the History of Arabic Sciences in Muscat. *Lettera Matematica*, *4*(2), 87–100.

Gouws, L. A., Bradshaw, K., & Wentworth, P. (2013). Computational thinking in educational activities. An evaluation of the educational game Light-Bot.

Proceedings of the 18th ACM conference on Innovation and technology in computer science education - ITiCSE '13, 10–15.

Grover, S., & Pea, R. (2013). Computational Thinking in K-12: A Review of the State of the Field. *Educational Researcher*, *42*(1), 38–43.

Grover, Shuchi. (2011). Robotics and Engineering for Middle and High School Students to Develop Computational Thinking. *Paper presented at the Annual Meeting of the American Educational Research Association New Orleans, April 7–11, 2011*, 1–15.

Guzdial, M. (2008). Education. Paving the way for computational thinking. *Communications of the ACM*, *51*(8), 25–27.

Hambrusch, S., Hoffmann, C., Korb, J. T., Haugan, M., & Hosking, A. L. (2009). A multidisciplinary approach towards computational thinking for science majors. *ACM SIGCSE Bulletin. SIGCSE'09, March 3–7, 2009, Chattanooga, Tennessee, USA*, *41*(1), 183–187.

Heintz, F., Mannila, L., & Färnqvist, T. (2016). A Review of Models for Introducing Computational Thinking, Computer Science and Computing in K-12 Education. *Proceedings - Frontiers in Education Conference, FIE*, 22.

Hemmendinger, D. (2010). A plea for modesty. *ACM Inroads*, *1*(2), 4–7.

Hjelm, E. (2016). *Tytöt, pojat ja tietojenkäsittelyn opettaminen*. Helsingin yliopisto.

Holtgrewe, U. (2014). New new technologies: the future and the present of work in information and communication technology. *New Technology, Work and Employment*, *29*(1), 9–24.

Hour of Code. (ei pvm.). *Hour of Code*. Noudettu osoitteesta https://hourofcode.com/us (Viitattu 7.6.2018)

Hsu, T.-C., Chang, S.-C., & Hung, Y.-T. (2018). How to learn and how to teach Computational thinking: Suggestions based on a review of the literature. *Computers & Education*, *126*, 296–310.

Hu, C. (2011). Computational Thinking – What It Might Mean and What We Might Do About It. *ITiCSE'11, Proceedings of the 16th annual joint conference on Innovation and technology in computer science education. ACM*, 223–227.

Hyde, J. S. (2005). The gender similarities hypothesis. *American Psychologist*, *60*(6), 581–592.

Hyde, J. S., & Mertz, J. E. (2009). Gender, culture, and mathematics performance. *Proceedings of the National Academy of Sciences of the United States of America*, *106*(22), 8801–8807.

Informatics Europe, & ACM Europe Working Groupon Informatics Education. (2013). *Informatics education: Europe cannot afford to miss the boat*.

International Olympiad in Informatics. (ei pvm.). *International Olympiads in Informatics Computer Science*. Noudettu osoitteesta https://ioinformatics.org/

ISTE, & CSTA. (2011). *Operational Definition of Computational Thinking for K-12 Education* (s. 1). Noudettu osoitteesta https://www.iste.org/resources, http://www.csteachers.org/, http://www.iste.org/docs/ct-documents/computational-thinking-operational-definition-flyer.pdf?sfvrsn=2

Johnson, R. B., & Onwuegbuzie, A. J. (2004). Mixed Methods Research: A Research Paradigm Whose Time Has Come. *Educational Researcher*, *33*(7), 14–26.

Jones, S. P. (2011). *Computing at School. International comparisons*. 1–12.

Joutsenlahti, J. (2002). Pitkän matematiikan opiskelijoiden matematiikkauskomukset 1990-luvulla. Teoksessa H. Silfverberg & J. Joutsenlahti (Toim.), *Tutkimuksella parempaan opetukseen. Matematiikan ja luonnontieteiden tutkimusseura ry:n päivät Tampereella 28.-29.9.2001. Tampereen yliopiston opettajankoulutuslaitoksen*

julkaisuja. Sivuja 184. (ss. 79–85). Tampere: Juvenes-Print - Tampereen yliopistopaino Oy.

Joutsenlahti, J. (2003). Kielentäminen matematiikan opiskelussa. Teoksessa A. Virta & O. Marttila (Toim.), *Opettaja, asiantuntijuus ja yhteiskunta. Ainedidaktinen symposium 7.2.2003* (ss. 188 – 196). Turun yliopisto.

Joutsenlahti, J. (2005). *Lukiolaisen tehtäväorientoituneen matemaattisen ajattelun piirteitä 1990-luvun pitkän matematiikan opiskelijoiden matemaattisen osaamisen ja uskomusten ilmentämänä.* Tampereen yliopisto. Opettajankoulutuslaitos.

Joutsenlahti, J. (2006). Matematiikan oppimateriaali ja oppilaan matemaattinen ajattelu. Teoksessa T. Asunta & J. Viiri (Toim.), *Polkuja tutkimukselliseen opettamiseen ja oppimiseen matemaattisissa aineissa. Tutkimuksia 84* (ss. 231–242). Jyväskylän yliopistopaino, Jyväskylä: Jyväskylän yliopisto, opettajankoulutuslaitos.

Joutsenlahti, J. (2009). Matematiikan kielentäminen kirjallisessa työskentelyssä. Teoksessa R. Kaasila (Toim.), *Matematiikan ja luonnontieteiden opetuksen tutkimuspäivät Rovaniemellä 7.-8.11.2008. Lapin yliopiston kasvatustieteellisiä raportteja 9.* (ss. 71–86). Rovaniemi: Lapin yliopisto.

Joutsenlahti, J., & Kulju, P. (2015). Kielentäminen matematiikan ja äidinkielen opetuksen kehittämisessä. Teoksessa T. Kaartinen (Toim.), *Monilukutaito kaikki kaikessa* (ss. 57–76). Tampereen Yliopistopaino Oy – Juvenes Print, Tampere: Tampereen yliopiston normaalikoulu.

Joutsenlahti, J., Kulju, P., & Tuomi, M. (2013). Matemaattisen lausekkeen kontekstualisointi sanalliseksi tehtäväksi ja tarinaksi. Opetuskokeilu kirjoittamisen hyödyntämisestä matematiikan opiskelussa. Teoksessa L. Tainio, K. Juuti, & S. Routarinne (Toim.), *Ainedidaktinen tutkimus koulutuspoliittisen päätöksenteon perustana*. Suomen ainedidaktisen tutkimusseuran julkaisuja. Ainedidaktisia tutkimuksia 4.

Järvinen, E.-M. (1998). Lego/logo-oppimisympäristö teknologiakasvatuksessa. Teoksessa S. Kaartinen (Toim.), *Matemaattisten aineiden opetus ja oppiminen. Matematiikan ja luonnontieteiden opetuksen tutkimusseura ry:n päivillä Oulussa 18.-20.10.1996 pidetyt esitelmät ja alustukset. Oulun yliopiston kasvatustieteiden tiedekunnan opetusmonisteita ja selosteita 78/1998. Oulun yliopiston kasvatustieteiden tiedekunta, Oulu. Sivuja 212.* (ss. 196–212). Oulu: Oulun yliopistopaino.

Kaarakainen, M., Kivinen, O., & Vainio, T. (2018). Performance-based testing for ICT skills assessing: a case study of students and teachers' ICT skills in Finnish schools. *Universal Access in the Information Society*, *17*(2), 349–360.

Kaarakainen, M.-T., Kivinen, O., & Tervahartiala, K. (2013). Kouluikäisten tietoteknologian vapaa-ajan käyttö. *Nuorisotutkimus*, (2), 20–33.

Kalelioğlu, F., Gülbahar, Y., & Kukul, V. (2016). A Framework for Computational Thinking Based on a Systematic Research Review. *Baltic J. Modern Computing*, *4*(3), 583–596.

Kananen, J. (2012). *Kehittämistutkimus opinnäytetyönä. Kehittämistutkimuksen kirjoittamisen käytännön opas.* (M. Heikkinen, Toim.). Jyväskylän ammattikorkeakoulu.

Katz, V. J., & Barton, B. (2007). Stages in the history of algebra with implications for teaching. *Educational Studies in Mathematics*, *66*(2), 185–201.

Kekäläinen, O. (2015). Onko automatisointiajattelu paras suomennos käsitteestä ”computational thinking”? Teoksessa J. Viteli & A. Östman (Toim.), *Tuovi 13: Interaktiivinen tekniikka koulutuksessa 2015-konferenssin tutkijatapaamisen artikkelit* (ss. 27–29). Research Reports 15. Informaatiotieteiden yksikkö, Tampereen yliopisto.

Kelleher, C., & Pausch, R. (2005). Lowering the Barriers to Programming : A Taxonomy of Programming Environments and Languages for Novice Programmers. *ACM Computing Surveys*, *37*(2), 83–137.

Kiss, G. (2010). A Comparison of Programming Skills by Genders of Hungarian Grammar School Students. *2010 Symposia and Workshops on Ubiquitous, Autonomic and Trusted Computing*, 24–30.

Kivinen, O., & Kaarakainen, M.-T. (2012). Opetusta eriyttävä digitaalinen ReadIT- opetusohjelma lukemisstrategioiden harjaannuttamisessa. *Kasvatus*, *43*(4), 361–374.

Knuth, D. E. (1997). *The art of computer programming: fundamental algorithms, volume 1*. Addison-Wesley.

Koh, K. H., Basawapatna, A., Bennett, V., & Repenning, A. (2010). Towards the automatic recognition of computational thinking for adaptive visual language learning. *Proceedings - 2010 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2010*, 59–66.

Kolaitis, P. G. (2011). Technical Perspective: The Quest for a Logic for Polynomial-Time Computation. *Association for Computing Machinery, ACM. New York*, *54*(6), 103.

Koodiaapinen. (2019). *Koodiaapinen (MOOC)*. Noudettu osoitteesta http://koodiaapinen.fi/mooc/ (Viitattu 8.9.2017)

Laine, E. (2017). *Tietokoneavusteisen geometrian opetuksen vaikutuksia laskennallisen ajattelun kehittymiseen ja opiskelijoiden motivaatioon*. Tampereen yliopisto.

Laitinen, M., Rantamäki, H., & Joutsenlahti, J. (2015). Puhutko matematiikkaa? Teoksessa T. Kaartinen (Toim.), *Monilukutaito kaikki kaikessa*. Tampere: Tampereen yliopiston normaalikoulu. Tampereen Yliopistopaino Oy - Juvenes Print.

Lankiewicz, H., & Wąsikiewicz-Firlej, E. (2014). *Languaging Experiences: Learning and Teaching Revisited*. Cambridge Scholars Publishing.

Lau, W. W. F., & Yuen, A. H. K. (2009). Exploring the effects of gender and learning styles on computer programming performance: Implications for programming pedagogy. *British Journal of Educational Technology*, *40*(4), 696–712.

Lee, I., Martin, F., Denner, J., Coulter, B., Allan, W., Erickson, J., … Werner, L. (2011). Computational Thinking for Youth in Practice. *ACM Inroads*, *Vol. 2*(No. 1), 32–37.

Liikenne- ja viestintäministeriö, Tekes, & Teknologiateollisuus ja Verkkoteollisuus. (2017). *Digibarometri 2017*. Noudettu osoitteesta http://www.digibarometri.fi

Linn, M. C. (1985). The Cognitive Consequences of Programming Instruction in Classrooms. *Educational Researcher*, *14*(5), 14-16 + 25-29.

Lister, R., Seppälä, O., Simon, B., Thomas, L., Adams, E. S., Fitzgerald, S., … Sanders, K. (2004). A multi-national study of reading and tracing skills in novice programmers. *ACM SIGCSE Bulletin*, *36*(4), 119–150.

Liukas, L. (2015). Hello Ruby. Teoksessa K. Myllyrinne (Kääntäjä), *Alkuteos: Hello Ruby: Adventures in Coding.* (Neljäs painos, s. 112). Keuruu: Otavan kirjapaino Oy.

Loidl, S., Mühlbacher, J., & Schauer, H. (2005). Preparatory Knowledge: Propaedeutic in Informatics. Teoksessa R. T. Mittermeir (Toim.), *From Computer Literacy to Informatics Fundamentals. International Conference on Informatics in Secondary Schools – Evolution and Perspectives, LNCS 3422, ISSEP 2005 Klagenfurt, Austria, March 30 - April 1, 2005 Proceedings. Roland T. Mittermeir (Ed.)* (ss. 104 – 115). Germany: Springer-Verlag Berlin Heidelberg.

Lopez, M., Whalley, J., Robbins, P., & Lister, R. (2008). Relationships between reading, tracing and writing skills in introductory programming. *Proceeding of the fourth international workshop on Computing education research - ICER '08. Sydney, Australia.*, 101–112.

Lowrie, I. (2018). ALGORITHMS AND AUTOMATION: An Introduction. *Cultural Anthropology*, *33*(3), 349–359.

Lu, J. J., & Fletcher, G. H. L. (2009). Thinking about computational thinking. *ACM SIGCSE Bulletin*, *41*(1), 260–264.

Luoma-aho, E. (2010). Matematiikan peruskäsitteiden historia. 1. Algebra ja aritmetiikka. *Matematiikkalehti Solmu*, 80.

Lye, S. Y., & Koh, J. H. L. (2014). Review on teaching and learning of computational thinking through programming: What is next for K-12? *Computers in Human Behavior*, *41*, 51–61.

Mannila, L., Dagiene, V., Demo, B., Grgurina, N., Mirolo, C., Rolandsson, L., & Settle, A. (2014). Computational Thinking in K-9 Education. *Proceedings of the Working Group Reports of the 2014 on Innovation & Technology in Computer Science Education Conference - ITiCSE-WGR '14*, 29.

Marcelino, M. J., Pessoa, T., Vieira, C., Salvador, T., & Mendes, A. J. (2018). Learning Computational Thinking and scratch at distance. Teoksessa *Computers in Human Behavior* (Vsk. 80, ss. 470–477). Elsevier Ltd.

Margolis, J., Ryoo, J. J., Sandoval, C., Lee, C., Goode, J., & Chapman, G. (2012). Beyond access: Broadening participation in high school computer science. *ACM Inroads*, *3*(4), 72–78.

Markov, A. A. (1954). *Theory of Algorithms (Teoriya algorifmov). WORKS OF THE MATHEMATICAL INSTITUTE IM. V.A. STEKLOV. Vol. XLII* (J. J. Schorr-Kon & Israel Program for Scientific Translations, IPST Staff, Käänt.). Moskva-Leningrad, Jerusalem: Izdatel'stvo Akademii Nauk SSSR. Tulostus: Keter Press, sidonta: Wiener Bindery Ltd.

Marshall, K. S. (2011). *Was that CT? Assessing Computational Thinking Patterns through Video-Based Prompts*. Esitetty tilaisuudessa Proceedings of the Annual Meeting of the American Educational Research Association, New Orleans, LA.

Massachusetts Institute of Technology. (2018). MIT App Inventor. Noudettu 12. heinäkuuta 2018, osoitteesta http://appinventor.mit.edu/explore/front.html (Viitattu 12.7.2018)

McDowell, C., Werner, L., Bullock, H. E., & Fernald, J. (2003). The impact of pair programming on student performance, perception and persistence. *IEEE. Proceedings of the 25th International Conference on Software Engineering (ICSE'03)*, 602–607.

McKenney, S., & Reeves, T. C. (2014). Educational Design Research. Teoksessa J. M. Spector, M. D. Merrill, J. Elen, & M. J. Bishop (Toim.), *Handbook of Research on Educational Communications and Technology* (ss. 131–140). Noudettu osoitteesta http://link.springer.com/10.1007/978-1-4614-3185-5_11 (Viitattu 27.4.2019)

Milosz, M., & Milosz, E. (2018). Professional Work of Computer Science Students and their Academic Achievements – Imagination vs. Reality. *International Journal of Engineering Pedagogy (iJEP)*, *8*(1), 16–28.

Moreno-León, J., & Robles, G. (a2015). Analyze your Scratch projects with Dr. Scratch and assess your Computational Thinking skills. *Scratch Conference 2015*, 1–7.

Moreno-León, J., & Robles, G. (b2015). Dr. Scratch: a web tool to automatically evaluate Scratch projects. *ACM. Proceedings of the Workshop in Primary and Secondary Computing Education, WiPSCE '15*, 132–133.

Moreno-León, J., Robles, G., & Román-González, M. (2015). Dr. Scratch: Automatic Analysis of Scratch Projects to Assess and Foster Computational Thinking. *RED. Revista de Educación a Distancia*, (46), 1–23.

Moreno-Leon, J., Robles, G., & Roman-Gonzalez, M. (2016). Comparing computational thinking development assessment scores with software complexity metrics. *IEEE Global Engineering Education Conference, EDUCON*, 1040–1045.

Moschovakis, Y. N. (2001). *What is an algorithm? Teoksessa Mathematics Unlimited - 2001 and beyond* (Engquist, B & Schmid, W, Toim.). Noudettu osoitteesta http://www.math.ucla.edu/~ynm/papers.htm

Moskal, B., Lurie, D., & Cooper, S. (2004). Evaluating the effectiveness of a new instructional approach. *ACM SIGCSE Bulletin. Proceedings of the 35th SIGCSE technical symposium on Computer science education - SIGCSE '04*, 75–79.

Mäcklin, J., & Nikula, M. (2010). *Matemaattisen ajattelun kirjallinen kielentäminen matemaattisen ongelman ratkaisuvälineenä*. Opettajankoulutuslaitos, Hämeenlinna.

National Science Foundation. (ei pvm.). *The Value of Computational Thinking across Grade Levels (VCTAL)*. Noudettu osoitteesta https://www.comap.com/Free/VCTAL/index.html

Neittaanmäki, P., Lehto, M., & Kankaanranta, M. (2014). *KOHTI LASKENNALLISEN AJATTEUN OSAAMISTA Oppilaiden laaja-alaisen osaamisen edistäminen ja tieto- ja viestintätekniikan opetuskäytön vahvistaminen* (s. 47). Informaatioteknologian tiedekunta, Jyväskylän yliopisto.

Niemelä, P. (2018). *From Legos and Logos to Lambda: A Hypothetical Learning Trajectory for Computational Thinking*. Tampereen teknillinen yliopisto.

Nunes, T., Bryant, P., Evans, D., Bell, D., Gardner, S., Gardner, A., & Carraher, J. (2007). The contribution of logical reasoning to the learning of mathematics in primary school. *British Journal of Developmental Psychology*, *25*(1), 147–166.

Näveri, L., Ahtee, M., Laine, A., Pehkonen, E., & Hannula, M. S. (2012). Erilaisia tapoja johdatella ongelmanratkaisutehtävään - esimerkkinä aritmagon-tehtävän ratkaiseminen alakoulun kolmannella luokalla. Teoksessa H. Krzywacki, K. Juuti, & J. Lampiselkä (Toim.), *Suomen ainedidaktisen tutkimusseuran julkaisuja, Ainedidaktisia tutkimuksia 2. Matematiikan ja*

luonnontieteiden opetuksen ajankohtaista tutkimusta (ss. 81–98/203). Suomen ainedidaktinen tutkimusseura ry. Unigrafia Oy, Helsinki.

OECD. (2016). *PISA 2015 Results (Volume I): Excellence and Equity in Education, PISA*. Noudettu osoitteesta http://dx.doi.org/10.1787/9789264266490-en

Ojanen, S. (2016). *Kirjallinen kielentäminen yliopiston matematiikan opetuksessa*. Helsingin yliopisto.

Opetus- ja kulttuuriministeriö. (7.12.2016). *PISA 2015: Suomalaisnuoret edelleen huipulla, pudotuksesta huolimatta. OKM:n lehdistötiedote 6.12.2016*. Noudettu osoitteesta http://www.oph.fi/ajankohtaista/verkkouutiset/101/0/pisa_2015_suomalaisnuoret_edelleen_huipulla_pudotuksesta_huolimatta?language=fi (Viitattu 27.3.2018)

Opetus- ja kulttuuriministeriö. (21.7.2017). PISA 2015: Suomalaisnuoret parhaiden joukossa yhteistoiminnallisessa ongelmanratkaisussa [Tiedote]. Noudettu osoitteesta https://minedu.fi/artikkeli/-/asset_publisher/pisa-2015-suomalaisnuoret-parhaiden-joukossa-yhteistoiminnallisessa-ongelmanratkaisussa

Opetus- ja kulttuuriministeriön tiedote. (21.1.2014). *Kiuru: Ohjelmointi peruskoulun opetussuunnitelman perusteisiin*. Noudettu osoitteesta http://valtioneuvosto.fi/artikkeli/-/asset_publisher/1410845/kiuru-ohjelmointi-peruskoulun-opetussuunnitelman-perusteisiin (Viitattu 22.3.2018)

Opetushallitus. (2004). *Perusopetuksen opetussuunnitelman perusteet 2004*. Vammalan Kirjapaino Oy Vammala: Opetushallitus.

Opetushallitus. (2015). *Lukion opetussuunnitelman perusteet 2015*. Noudettu osoitteesta http://www.oph.fi/download/172124_lukion_opetussuunnitelman_perusteet_2015.pdf

Opetushallitus. (2016). *Perusopetuksen opetussuunnitelman perusteet 2014* (4. painos). Next Print Oy, Helsinki: Opetushallitus.

Opetushallitus. (2019). *Lukion opetussuunnitelman perusteet 2019. Luonnos 14.3.2019*.

Panko, R. R. (2008). IT employment prospects: beyond the dotcom bubble. *European Journal of Information Systems*, *17*(3), 182–197.

Papastergiou, M. (2008). Are Computer Science and Information Technology still masculine fields? High school students' perceptions and career choices. *Computers and Education*, *51*, 594–608.

Papert, S. (1980). *Mindstorms: Children, Computers, and Powerful Ideas*. New York: Basic Books, Inc.

Papert, S. (1985). *Lapset, tietokoneet, ajattelemisen taito. (Alkuperäisteoksesta Mindstorms: Children, Computers, and Powerful Ideas. All about LOGO - how it was invented and how it works, 1980)* (J. Lehtinen, Kääntäjä.). Gummerus Oy, Jyväskylä: Kirjayhtymä.

Park, Y. (13.6.2016). 8 digital skills we must teach our children. Noudettu 17. toukokuuta 2018, osoitteesta World Economic Forum website: https://www.weforum.org/agenda/2016/06/8-digital-skills-we-must-teach-our-children/ (Viitattu 17.5.2018)

Perković, L., Settle, A., Hwang, S., & Jones, J. (Bilkent, Ankara, Turkey.2010). A framework for computational thinking across the curriculum. *Proceedings of the fifteenth annual conference on Innovation and technology in computer science education - ITiCSE '10*, 123–127.

Pernaa, J. (Toim.). (2013). *Kehittämistutkimus opetusalalla*. PS-kustannus.

Pioro, B. T. (2004). Performance in an Introductory Computer Programming Course as a Predictor of Future Success for Engineering and Computer Science Majors. *International Conference on Engineering Education, Gainesville, Fl, USA*, 1–22.

Pólya, G. (1971). *How to Solve it: A New Aspect of Mathematical Method*. Yhdysvallat: Princeton University Press.

Queen Mary University of London. (ei pvm.). *CS4FN - Computer Science for Fun*. Noudettu osoitteesta http://www.cs4fn.org/

Raspberry Pi Foundation. (ei pvm.). *Code Club*. Noudettu osoitteesta codeclub.org

Repenning, A. (2006). Excuse me, I need better AI! Employing Collaborative Diffusion to make Game AI Child's Play. *Proceedings of the 2006 ACM SIGGRAPH symposium on Videogames - Sandbox '06*, 169–178.

Repenning, A., Webb, D., & Ioannidou, A. (2010). Scalable game design and the development of a checklist for getting computational thinking into public schools. *Proceedings of the 41st ACM technical symposium on Computer science education - SIGCSE '10*, 265–269.

Riley, D. D., & Hunt, K. A. (2014). *Computational Thinking for the Modern Problem Solver*. Boca Raton, FL: CRC Press.

Román-González, M., Pérez-González, J.-C., & Moreno-León, J. (2018). Extending the nomological network of computational thinking with non-cognitive factors. *Computers in Human Behavior*, *80*, 441–459.

Saarikoski, P. (2011). Computer Courses in Finnish Schools during 1980-1995. *In History of Nordic Computing HiNC3, 3rd IFIP WG9.7 Working Conference on History of Nordic Computing, Stockholm 18th–20th October 2010.*, 9.

Sarikka, H. (2014). *Kielentäminen matematiikan opetuksen ja oppimisen tukena* (Diplomityö). Tampereen teknillinen yliopisto.

Selby, C. (2012). Promoting computational thinking with programming. *ACM. Proceedings of the 7th Workshop in Primary and Secondary Computing Education on - WiPSCE '12*, 74–77.

Selby, C. (2014). *How can the teaching of programming be used to enhance computational thinking skills?* (Phd, University of Southampton).

Noudettu osoitteesta https://eprints.soton.ac.uk/366256/ (Viitattu 12.2.2019)

Selby, C., & Woollard, J. (2013). *Computational Thinking: The Developing Definition*. Noudettu osoitteesta http://eprints.soton.ac.uk/id/eprint/356481 (Viitattu 11.9.2017)

Sentance, S., & Csizmadia, A. (2017). Computing in the curriculum: Challenges and strategies from a teacher's perspective. *Education and Information Technologies*, *22*(2), 469–495.

Seppälä, O. (2012). *Advances in Assessment of Programming Skills*. Aalto yliopisto.

Settle, A., Franke, B., Hansen, R., Spaltro, F., Jurisson, C., Rennert-May, C., & Wildeman, B. (2012). Infusing computational thinking into the middle- and high-school curriculum. *ACM. Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education - ITiCSE '12*, 22–27.

Sfard, A. (2008). *Thinking as Communicating. Human Development, the Growth of Discourses, and Mathematizing.* United States of America: Cambridge University Press.

Silfverberg, H., Portaankorva-Koivisto, P., & Yrjänäinen, S. (2005). Matematiikka kielenä ja kielikasvatuksena. Teoksessa L. Jalonen, T. Keranto, & K. Kaila (Toim.), *Matematiikan ja luonnontieteiden opetuksen tutkimuksen tutkimuspäivät Oulussa 25.-26.11.2004. Matemaattisten aineiden opettajan taitotieto - haaste vai mahdollisuus?* Oulun yliopisto, Oulu: Kasvatustieteiden ja opettajankoulutuksen yksikkö, Oulun yliopisto.

Singh, S. (2012). Developing e-skills for competitiveness, growth and employment in the 21st century: The European perspective. *International Journal of Development Issues*, *11*(1), 37–59.

Sneider, C., Stephenson, C., Schafer, B., & Flick, L. (2014). Teacher's Toolkit: Exploring the science Framework and NGSS: Computational thinking in the science classroom. *Science Scope*, (Vol. 38, No. 3), 10–15.

Soare, R. I. (a1996). Computability and Recursion. *The Bulletin of Symbolic Logic*, *2*(3), 284–321.

Soare, R. I. (b1996). Computability and Recursion. *The Bulletin of Symbolic Logic*, *2*(3), 284–321.

Soare, R. I. (1999). *The history and concept of computability. Teoksessa Handbook of Computability Theory. Sivut 3-36 tai 1-35.* (E. R. Griffor, Toim.). Noudettu osoitteesta http://www.people.cs.uchicago.edu/~soare/History/

Spelke, E. S. (2005). Sex differences in intrinsic aptitude for mathematics and science? A critical review. *American Psychologist*, *60*(9), 950–958.

Stager, G. S. (2016). Seymour Papert (1928–2016). *Nature*, *537*, 308.

Striphas, T. (2015). Algorithmic culture. *European Journal of Cultural Studies*, *18*(4–5), 395–412.

Sysło, M. M., & Kwiatkowska, A. B. (2008). The Challenging Face of Informatics Education in Poland. Teoksessa R. T. Mittermeir & M. M. Sysło (Toim.), *Informatics Education – Supporting Computational Thinking. Third International Conference on Informatics in Secondary Schools – Evolution and Perspectives, ISSEP 2008 Torun, Poland, LNCS 5090, July 1-4, 2008 Proceedings. Roland T. Mittermeir Maciej M. Sysło (Eds.)* (ss. 1–18). Germany: Springer-Verlag Berlin Heidelberg.

Sysło, M. M., & Kwiatkowska, A. B. (2013). Informatics for All High School Students A Computational Thinking Approach. Teoksessa *Informatics in Schools. Sustainable Informatics Education for Pupils of all Ages: 6th International Conference on Informatics in Schools: Situation, Evolution, and Perspectives, LNCS 7780, ISSEP 2013, Oldenburg, Germany, February 26–March 2, 2013. Proceedings. I. Diethelm and R.T.*

Mittermeir (Eds.): ISSEP 2013 Diethelm, I., Mittermeir, R.T. & SpringerLink (Online service) 2013, Informatics in Schools. Sustainable Informatics Education for Pupils of all Ages: 6th International Conference on Informatics in Schools: Situation, Evolution, and Perspectives, ISSEP 2013, Oldenburg, Germany, February 26–March 2, 2013. Proceedings, Springer Berlin Heidelberg, Berlin, Heidelberg. (ss. 43–56). Springer-Verlag Berlin Heidelberg.

Taipale, A. (2010). *Matematiikan, lukemisen ja kirjoittamisen vaikeuksien päällekkäistyminen nuoruusiässä*. Joensuun yliopisto.

The Center for Computational Thinking at Carnegie Mellon. (ei pvm.). What is computational thinking? Noudettu 10. tammikuuta 2019, osoitteesta https://www.cs.cmu.edu/~CompThink/ (Viitattu 10.1.2019)

The Design-Based Research Collective. (2003). Design-Based Research: An Emerging Paradigm for Educational Inquiry. *Educational Researcher*, *32*(1), 5–8.

The great principles of computing. (2010). *American Scientist*, *98*(5), 369–372.

The Royal Society. (2012). *Shut down or restart? The way forward for computing in UK schools*.

Thiruvathukal, G. K. (2013). What's in an Algorithm? *Computing in Science & Engineering*, *15*(4), 4–5.

Tucker, A., Deek, F., Jones, J., McCowan, D., Stephenson, C., & Verno, A. (2006). A Model Curriculum for K-12 Computer Science: Final Report of the ACM K-12 Task Force Curriculum Committee. Second Edition. *Computer Science Association*, 60.

Tuomi, P., Multisilta, J., Saarikoski, P., & Suominen, J. (2018). Coding skills as a success factor for a society. *Education and Information Technologies*, *23*, 419–434.

Van Gorp, M. J., & Grissom, S. (2001). An Empirical Evaluation of Using Constructive Classroom Activities to Teach Introductory Programming. *Computer Science Education*, *11*(3), 247–260.

Vilkka, H. (2015). *Tutki ja kehitä* (4. painos). Jyväskylä: PS-kustannus.

vpython.org. (ei pvm.). VPython. 3D Programming for Ordinary Mortals. Noudettu 12. heinäkuuta 2018, osoitteesta http://vpython.org/ (Viitattu 12.7.2018)

Vygotski, L. (1982). *Ajattelu ja kieli* (K. Helkama & A. Koski-Jännes, Käänt.). Weilin+Göös.

Wake, G. (2016). Mathematics, modelling and students in transition. *Teaching Mathematics and its Applications*, *35*(3), 172–186.

Wakefield, D. (1999). *Se Habla Mathematics? Consideration of Math as a Foreign Language*. 11.

Wakefield, D. (2000). *Math as a Second Language*. Teoksessa *Volume 64:3, Spring 2000* (ss. 272–279). The Educational forum.

Webb, D. C. (2010). Troubleshooting assessment: An authentic problem solving activity for it education. *Procedia - Social and Behavioral Sciences*, 903–907.

Weinberg, A. E. (2013). *Computational thinking: an investigation of the existing scholarship and research*. 97.

Weintrop, D. (2018). Introducing Computational Thinking Across the Curriculum with Virtual Reality. *Kong, S.C., Andone, D., Biswas, G., Crick, T., Hoppe, H.U., Hsu, T.C., Huang, R.H., Li, K.Y., Looi, C.K., Milrad, M., Sheldon, J., Shih, J.L., Sin, K.F., Tissenbaum, M., & Vahrenhold, J. (Eds.). (2018). Proceedings of the International Conference on Computational Thinking Education 2018. Hong Kong: The Education University of Hong Kong*, 83–88.

Weintrop, D., Beheshti, E., Horn, M., Orton, K., Jona, K., Trouille, L., & Wilensky, U. (2016). Defining Computational Thinking for Mathematics

and Science Classrooms. *Journal of Science Education and Technology, Springer Netherlands*, *25*(1), 127–147.

Werner, L., Denner, J., & Campe, S. (2012). The Fairy Performance Assessment: Measuring Computational Thinking in Middle School. *ACM. Proceedings of the 43rd ACM Technical Symposium on Computer Science Education - SIGCSE '12*, 215–220.

Wilson, C., Sudol, L. A., Stephenson, C., & Stehlik, M. (2010). Running on empty: The Failure to Teach K-12 Computer Science in the Digital Age. *The Association for Computing Machinery (ACM), The Computer Science Teachers Association (CSTA)*, 72.

Wing, J. M. (2006). Computational Thinking. *Communications of the ACM*, *March 2006*(Vol 49, No. 3), 33–35.

Wing, J. M. (2008). Computational thinking and thinking about computing. *The Royal Society*, *366*, 3717–3725.

Wing, J. M. (a2011). Computational thinking - What and Why? *The Link, News from the School of Computer Science. Carnegie Mellon University*, *Spring 2011*(Issue 6.0), 20–23.

Wing, J. M. (b2011). Computational Thinking - What and Why? Teoksessa *The Link. News from the School of Computer Science. Closing the K-12 Gap* (ss. 20-23 (28)). Carnegie Mellon University.

Wolz, U., Stone, M., Pearson, K., Pulimood, S. M., & Switzer, M. (2011). Computational Thinking and Expository Writing in the Middle School. *ACM Transactions on Computing Education*, *11*(2), 1–22.

Yadav, A., Hong, H., & Stephenson, C. (2016). Computational Thinking for All: Pedagogical Approaches to Embedding 21st Century Problem Solving in K-12 Classrooms. *TechTrends, Springer US, New York*, *60*(6), 565–568.

Yadav, A., Stephenson, C., & Hong, H. (2017). Computational thinking for teacher education. *Communications of the ACM*, *60*(4), 55–62.

Yagci, M. (2019). A Valid and Reliable Tool for Examining Computational Thinking Skills. *Education and Information Technologies*, *24*(1), 929–951.

Yrjönsuuri, R. (1998). Opiskelutaito ja matematiikka. Teoksessa S. Kaartinen (Toim.), *Matemaattisten aineiden opetus ja oppiminen. Matematiikan ja luonnontieteiden opetuksen tutkimusseura ry:n päivillä Oulussa 18.-20.10.1996 pidetyt esitelmät ja alustukset. Oulun yliopiston kasvatustieteiden tiedekunnan opetusmonisteita ja selosteita 78/1998. Oulun yliopiston kasvatustieteiden tiedekunta, Oulu. Sivuja 212.* (ss. 88–105). Oulu: Oulun yliopistopaino.

Zaman, A., Farooq, R., Hussain, A., Ghaffar, A., & Satti, A. N. (2017). Logical thinking in mathematics: a study of secondary school students in pakistan. *Journal of the Research Society of Pakistan*, *54*(1).

Liite 1(1)

Wing 2006	ESIMERKKEJÄ SISÄLLÖISTÄ: ongelmanratkaisu, hajottaminen, esittäminen, mallintaminen, abstraktio, heuristinen päättely, etsiminen, suunnittelu, käsitteellistäminen, rekursio ja rinnakkaisuus VIITATAAN MYÖS: algoritmit, yleistäminen, arviointi
Csizmadia ym. 2015	KÄSITTEET: looginen päättely, algoritmit, hajottaminen, mallit, abstraktio ja arviointi ASENTEET / LÄHESTYMIS-TAVAT: kokeellisuus ja pelillisyys, suunnittelu ja luominen, virheiden etsiminen ja korjaaminen, sinnikkyys ja yhteistyö TEKNIIKAT: reflektointi, koodaus, suunnittelu, analysointi ja soveltaminen
Barr & Stephenson 2011 **Mannila ym. 2014**	YDINKÄSITTEET: datan kerääminen, datan analysointi, datan esittäminen, ongelman hajottaminen, abstraktio, algoritmit ja proseduurit, automaatio, rinnakkaisuus ja simulaatio KYVYT: ratkaisujen suunnittelu ongelmiin (käyttäen abstraktiota, automaatiota, algoritmien luomista, tiedonkeruuta ja analysointia), suunnitelmien toteuttaminen (ohjelmointi tarvittaessa), testaaminen ja debuggaus, mallintaminen, simulaatioiden ajaminen, järjestelmäanalyysien tekeminen, käytännön ja kommunikoinnin reflektoiminen, sanaston käyttäminen, abstraktioiden tunnistaminen ja liikkuminen abstraktiotasojen välillä, innovaatio, tutkiminen ja luovuus yli oppiainerajojen, ongelmanratkaisu ryhmässä, monipuolisten oppimisstrategioiden käyttäminen
Grover & Pea, 2013	abstraktio, yleistys, mallit, simulaatiot, systemaattinen informaation prosessointi, symbolijärjestelmät, esitykset, algoritmit, hajotus, modulaarisuus, iteratiivinen ajattelu,rekursiivinen ajattelu, rinnakkainen ajattelu, ehtologiikka, tehokkuus, suoritusrajoitukset, virheiden havaitseminen, virheiden etsiminen ja virheiden korjaus
Selby & Woollard 2013	HYVÄKSYTTÄVÄT: kognitiivinen prosessi, ajatteluprosessi, abstraktio, hajottaminen, algoritminen suunnittelu, algoritminen ajattelu (AT), arviointi, yleistys HYLÄTYT, ESIMERKKEJÄ: heuristinen ajattelu, iteratiivinen ajattelu, looginen ajattelu, matemaattinen ajattelu, proseduraalinen ajattelu, rinnakkaisajattelu, tekninen ajattelu, analyysi, automaatio,ennaltaehkäisy, hypoteesit, innovointi, järjestelmien suunnittelu, mallintaminen, ongelmanratkaisu, simulaatio, tunnistaminen, visualisaatio, tietojenkäsittelynkäsitteet, rekursio, redundanssi, ohjelmointi, muuttujat, tyypit, virheenkorjaus, etsintä ja testaaminen
Lee ym. 2011	abstraktio, automaatio, analyysi, mallintaminen, simulaatio, robotiikka ja pelisuunnittelu
Angeli ym. 2016	abstraktio, yleistys, hajotus, algoritmit (peräkkäisyys, suoritusjärjestyksen hallinta) ja virheiden etsiminen ja korjaus
Weintrop ym. 2015	kyky käsitellä avoimia ongelmia, sinnikkyys työskennellessä vaikeiden ongelmien parissa, luottamus monimutkaisuuden käsittelemiseen, ideoiden esittäminen laskennallisesti merkityksellisillä tavoilla, laajojen ongelmien hajottaminen pienempiin ongelmiin, abstraktioiden luominen tietyn ongelman näkökohtiin, ongelmien muotoilu uudelleen tunnistettavaksi ongelmaksi, vahvuuksien ja heikkouksien arviointi datan tai järjestelmän esityksessä, algoritmisten ratkaisujen muodostaminen sekä monimerkityksellisyyden havaitseminen ja osoittaminen algoritmeissa
Brennan & Resnick 2012	LASKENNALLISET KÄSITTEET jonot, silmukat, rinnakkaisuus, tapahtumat, ehdot, operaattorit, data LASKENNALLISET KÄYTÄNNÖT kasautuvuus, iteratiivisuus, testaus, virheenetsintä, virheen korjaus, uudelleenkäyttö, uudelleenmiksaus, abstraktiot ja modularisointi LASKENNALLISET NÄKÖKULMAT esittäminen, yhdistäminen ja kyseenalaistaminen

Liite 2(1). Tutkimuslomake.

TUTKIMUKSEEN OSALLISTUJAN SAATEKIRJE

Hei,

Opiskelen Tampereen yliopistossa matemaattisten aineiden aineenopettajaksi. Pro gradu -tutkielmani aiheena on *”Algoritmisen ajattelun kehittäminen ohjelmoinnin ja kielentämisen avulla matematiikan opetuksessa”*. Algoritminen ajattelu on ajankohtainen aihe peruskoulun opetussuunnitelmauudistusten myötä, ja ohjelmointi on liitetty osaksi matematiikan oppiainetta. Kielentämistä harjoittamalla voit saada oppimisen apuvälineen matematiikkaan. Tutkimalla algoritmista ajatteluasi kielentämiskokeilulla on mahdollista saada arvokasta tietoa algoritmisen ajattelun kehittämiseen opetuksessa.

Tutkimuksen tarkoitus ja toteutus

Tutkielman tutkimuskohderyhmänä ovat yläkoulun oppilaat ja lukiolaiset. Tutkimuksen aineisto kerätään kielennettävien perusgeometriaa sisältävien ohjelmointitehtävien ja kyselylomakkeen avulla. Tutkimuksen tarkoituksena on selvittää, minkälaisena algoritminen ajattelu näyttäytyy tutkittavilla, miten kielentäminen tuo ohjelmoinnissa algoritmisen ajattelun esille ja miten tutkittavat kokivat tehtävät. Lisäksi tutkitaan, tuoko hyvä dokumentointi ohjelmointiin palautetta ja millaisia ovat tutkittavien taustatekijöiden yhteydet algoritmisen ajattelun näkyvyydessä.

Tutkimukseen osallistuminen ja luottamuksellisuus

Osallistumisesi tutkimukseen on vapaaehtoista, ja vaatii myönteisen luvan sinulta ja ollessasi alaikäinen myös huoltajaltasi tai lailliselta edustajaltasi. Sinua ei voida yksilöiden tunnistaa tutkimustuloksista ja tutkimuksen julkaisusta. Tutkimukseen osallistuvien koulujen, rehtoreiden, opettajien, huoltajien tai laillisten edustajien ja sinun tietojasi ei mainita tutkimuksessa. Tutkimustehtävien ratkaisuistasi voidaan liittää kuvia tutkimukseen, jos tähän on saatu lupa.

Tutkimusaineisto säilytetään ja käsitellään luottamuksellisesti sekä tutkimuksen valmistumisen jälkeen hävitetään. Tutkimukseen liittyvissä asioissa voit aina olla yhteydessä tutkimuksen tekijään.

Tutkimuslupaa toivoen ja tutkimusta innolla odottaen

…

TUTKIMUKSEEN OSALLISTUVAN LUPA TUTKIMUKSELLE

Olen lukenut saatekirjeen algoritmista ajattelua koskevasta pro gradu -tutkimuksesta. Tutkimusaineisto kerätään tehtävien ja kyselylomakkeen avulla.

Ymmärrän, että minua koskeva tutkimusaineisto käsitellään ja säilytetään luottamuksellisesti sekä hävitetään tutkimuksen valmistumisen jälkeen. Tutkimukseen osallistuvaa koulua, rehtoria, opettajia, huoltajia tai laillisia edustajia ja minun tietojani ei voi yksilöiden tunnistaa tutkimustuloksista. Tutkimustehtävien ratkaisuistani voidaan liittää kuvia tutkimukseen, jos tähän on saatu lupa.

Osallistun lomakkeen täyttöön ja tutkimukseen:

☐ KYLLÄ ☐ EI

Tehtävieni ratkaisuista saa ottaa kuvia tutkimukseen:

☐ KYLLÄ ☐ EI

Paikka ja aika

Allekirjoitus

Nimenselvennys

Liite 2(2)

Ohje lomakkeen täyttämiseen

Tutustu ensin tähän ohjeeseen ja esimerkkeihin A ja B. Ratkaise sitten lomakkeen tehtävät 1-4 (käytä tarvittaessa apuna esimerkkitehtävää C), ja täytä lopuksi kyselylomake. Tee tehtävät ja täytä kysely itsenäisesti. Tee vähintään yksi tehtävä ja täytä kyselylomake tutkimukseen osallistuaksesi. Aikaa lomakkeen täyttöön kuluu noin 20 min (yksi tehtävä ja kysely) – 90 min (kaikki neljä tehtävää ja kysely). Voit tehdä tehtäviä missä tahansa järjestyksessä.

Tehtävissä aihealueena on perusgeometria, joka esiintyy ohjelmointikoodissa. Älä huolestu, jos et ole ohjelmoinut aiemmin!

Ratkaise tehtävät niin huolellisesti kuin pystyt. Tärkeää on, että kielennät ratkaisusi tehtävän oheen, eli kirjoitat matemaattisen ajattelusi ja lisäksi koodin toiminnan omin sanoin luonnollisella kielellä (ks. mallia esimerkkitehtävistä). Kirjoittamalla siis esität perusteluja ja tarvittaessa arvauksia, mitä mielestäsi matematiikassa ja ratkaisukoodeissa tapahtuu. Lisäksi voit piirtää ratkaisuusi kirjoituksen oheen havainnollistavia kuvia.

Esimerkkitehtävässä A on esitetty, mitä kielentämisellä tarkoitetaan.

Esimerkkitehtävässä B on esitetty, mitä koodin tulkitsemisella tarkoitetaan.

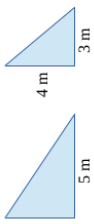
Tarvittaessa:

Esimerkkitehtävässä C on esitetty hyvin yksityiskohtaisesti kielentämisen ja koodin tulkitsemisen yhtenäinen ratkaisu. Älä huolestu esimerkin pituudesta. Esimerkissä on käsitelty kaikki varsinaisissa tehtävissä esiintyvät koodirakenteet, siksi esimerkki on pitkä.

Pyydän palauttamaan täytetyn lomakkeen kokonaisuudessaan (luvat, tehtävä(t) ja kyselylomake) opettajalle _____ **mennessä**.

ESIMERKKITEHTÄVÄ A (Älä vastaa tähän, mutta katso tästä esimerkkiä.)

Kielennä, mitä seuraavassa matemaattisessa osiossa tapahtuu.



$$\frac{5 \cdot 4}{2} + \frac{3 \cdot 4}{2} = 16$$

RATKAISU ESIMERKKITEHTÄVÄÄN A. KIELENTÄMINEN:

Kolmiolle on annettu kuvassa kantojen ja korkeuksien pituudet. Pinta-alaltaan suuremman kolmion kanta on 5 metriä ja korkeus 4 metriä. Pinta-alaltaan pienemmän kolmion kanta on 3 metriä ja korkeus 4 metriä.

Kolmiopiirrosten alapuolella on laskettu ensin suuremman kolmion pinta-ala eli kanta kertaa korkeus jaettuna kahdella. Pinta-alaksi saadaan osoittajan kertolaskulla ja sieventämällä

$$\frac{5 \cdot 4}{2} = \frac{20}{2} = \frac{10}{1} = 10$$

Vastaavasti on laskettu pienemmän kolmion pinta-ala

$$\frac{3 \cdot 4}{2} = \frac{12}{2} = \frac{6}{1} = 6$$

Kolmioiden pinta-alat on laskettu yhteen

$$10 + 6 = 16$$

ja pinta-alojen yhteenlaskettu suuruus on 16 m^2, koska pituusyksikkönä käytettiin metrejä.

ESIMERKKITEHTÄVÄ B (Älä vastaa tähän, mutta katso tästä esimerkkiä.)

Tulkitse alla oleva koodi.

```
# Tämä on kommentti. Annetaan
# muuttujalle luku arvoksi 3.
luku = 3

if luku == 3:
    print("Luku on ",luku,".")
else:
    print("Luku ei ole 3.")
```

RATKAISU ESIMERKKITEHTÄVÄÄN B. KOODIN TULKINTA:

Vihreällä #-merkinnällä on tehty koodiin kommentti, jota tietokone ei suorita.

Muuttujalle luku on annettu arvoksi 3 sijoittamalla.

Koodissa on if-else -rakenne, joka kuvaa ehtoa. Jos muuttujan luku arvo on 3, koodin suorittava tietokone tulostaa if-osan alla olevan print-osan. Tulosteena on tällöin "Luku on 3.". Mikäli luku-muuttujan arvo ei ole 3, koodin suorittava tietokone tulostaa else-osan alapuolella olevan print-osan, "Luku ei ole 3.".

ESIMERKKITEHTÄVÄ C (Älä vastaa tähän, mutta katso tästä esimerkkiä.)

Selvitä kielentämisen avulla, mitä monikulmioiden pinta-aloja tietokone laskisi suorittaessaan alla olevan koodin.

Koodi:

```
kulmienLukumaara = 3

# Annetaan sijoituksilla
# kannalle arvoksi 2
# ja korkeudelle arvoksi 4
monikulmionKanta = 2
monikulmionKorkeus = 4


if kulmienLukumaara == 3:
      # Lasketaan ja tulostetaan
      # kolmion pinta-ala
      komionPintaAla =
         monikulmionKanta *
         monikulmionKorkeus / 2
      print("Kyseessä on kolmio, jonka
      pinta-ala on ",kolmionPintaAla,".")
else:
      print("Kyseessä ei ole kolmio")

kulmienLukumaara = 4

# Lasketaan suorakulmion pinta-ala
# erillisellä funktiolla
suorakulmionPintaAla =
laskeSuorakulmionPintaAla(monikulmionKant
a,monikulmionKorkeus)

# Lasketaan ja tulostetaan
# kolmen neliön pinta-alat
for i in range(1,3):
      nelionPintaAla = i * i
      print(nelionPintaAla)
```

RATKAISU ESIMERKKITEHTÄVÄÄN. KIELENTÄMINEN:

Edetään koodissa ja sen sisältämässä matematiikassa ylhäältä alaspäin, kuten normaalisti laskiessa ja tietokoneen suorittaessa koodia.

Koodissa arvon sijoitus muuttujaan tehdään yhtäsuuruusmerkin avulla. Muuttujan nimi kuvaa yleensä muuttujan tarkoituksen.
Koodissa kulmienLukumaara-muuttujalle on sijoitettu arvo kolme. Muuttujan nimi kulmienLukumaara kuvaa monikulmion kulmien määrää.

#-merkillä ja vihreällä fontilla on tehty kaikki koodin kommentit. Tietokone ei huomioi kommentteja, kun se suorittaa koodia. Kommentteja käytetään koodin ymmärrettävyyden lisäämisen tueksi.

Koodissa monikulmionKanta-muuttujalle on sijoitettu arvo 2 ja monikulmionKorkeus-muuttujalle arvo 4.

Koodissa rakenne if-else kuvaa ehtoa. Jos if-osan ehto on tosi, eli tässä koodissa jos kulmienLukumaara-muuttujan arvo on 3, tietokone suorittaa if-osan lauseet, jotka ovat sisennettynä if-ehdon alapuolelle.
Mikäli if-osan ehto ei täyty, eli tässä koodissa kulmienLukumaara-muuttujan arvo ei ole 3, tietokone suorittaa else-osan lauseet, jotka ovat else:-kohdan alapuolella.
Tässä koodissa kulmienLukumaara-muuttujalle on ylempänä annettu arvoksi 3, joten tietokone suorittaa if-osan, mutta ei else-osaa. If-osassa lasketaan kolmion pinta-ala yleistä laskukaavaa käyttäen (kolmion pinta-ala on kanta kertaa korkeus jaettuna kahdella). Koodissa kertolaskuoperaatiota kuvaa merkki (*) ja jakolaskua merkki (/). Jotta tietokone osaa laskea kolmion pinta-alan tämän koodin muuttujilla, on kolmion pinta-ala laskettu muuttujien monikulmionKanta (arvo 2) ja monikulmionKorkeus (arvo 4) avulla. Tämän kolmion pinta-alan arvo (koodin laskusta saadaan siis 2 * 4 / 2 = 4) on sijoitettu muuttujaan kolmionPintaAla (arvo siis 4). Koodin print()-osa tulostaa sulkujen sisällä olevan osan. Tulostus on siis: Kyseessä on kolmio, jonka pinta-ala on 4.
Koska koodi ei suorita else-osaa, niin tietokone ei tulosta "Kyseessä ei ole kolmio" -kohtaa.

Koodin if-else-rakenteen jälkeen kulmienLukumaara-muuttujalle sijoitetaan uusi arvo, 4, vanhan arvon 3 tilalle.

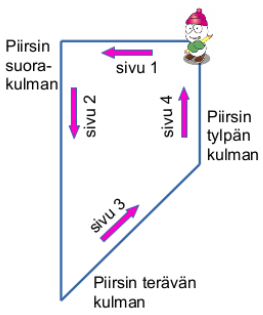
Koodissa kutsutaan funktiota laskeSuorakulmionPintaAla, joka saa parametreikseen suorakulmion pinta-alan laskemiseen tarvittavat muuttujat monikulmionKanta ja monikulmionKorkeus. Funktiota kutsumalla tietokone siirtyy erilliseen funktio-osaan (*ei ole näytetty viereisessä koodissa*), jossa se laskisi suorakulmion pinta-alan (monikulmionKanta * monikulmionKorkeus). Funktion avulla saatu suorakulmion pinta-alan arvo (8) sijoitetaan muuttujaan suorakulmionPintaAla.

Koodin for-osa käy läpi tilanteet muuttujan i lukuarvoilla 1, 2 ja 3. Neliön pinta-ala saadaan laskemalla neliön sivu kertaa sivu. Tässä lasketaan ensin neliön pinta-ala 1 kertaa 1. Pinta-alan arvo 1 tulostetaan. Seuraavaksi i saa arvon 2, jolloin neliön pinta-ala on 2 kertaa 2, ja pinta-alan arvoksi saadaan 4, joka tulostetaan. Viimeisenä i saa arvon 3, jolloin lasketaan neliön pinta-ala 3 kertaa 3, ja pinta-alaksi saadaan 9, joka tulostetaan.

Tehtävä 1

Eemil kiertää ensin kävellen puolisuunnikkaan muotoisen alueen ympäri (ks. kuva) ja mittaa samalla sen sivujen pituudet. Sitten hän laskee puolisuunnikkaan piirin ja pinta-alan.

Järjestä kirjaimia (A-K) vastaavat koodipalat oikeaan järjestykseen numeroituun taulukkoon niin, että Eemilin lasku toimii. Kielennä lisäksi tehtävä (voit halutessasi havainnollistaa kirjoitustasi myös piirtämällä).



RATKAISUSI TEHTÄVÄÄN 1:

Koodin suoritusjärjestys ja kielentäminen:

1	G
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	

Sivun 1 pituudeksi saadaan funktion kaveleJaMittaaSivu() avulla arvo 3.
Muuttujan kuljettuMatka arvoksi sijoitetaan sivu1:n arvo eli kuljettuMatka = sivu1 = 3.

A	`if kuljettuMatka > 13.0:`
	`  kaannyVasemmalle(45)`
	`else:`
	`  kaannyVasemmalle(135)`

B	`sivu4 = kaveleJaMittaaSivu()`
	`# sivu4 saa mitaksi 2`
	`kuljettuMatka = sivu1 + sivu2 + sivu3 +`
	`sivu4`

C	`print("Piirsin suorakulman")`
D	`kaannyOikealle(225)` `# 225 astetta`
E	`print("Piirsin tylpän kulman")`
F	`print("Piirsin terävän kulman")`

G	`sivu1 = kaveleJaMittaaSivu()`
	`# sivu1 saa mitaksi 3`
	`kuljettuMatka = sivu1`

H	`puolisuunnikkaanPiiri = kuljettuMatka`
	`print("Puolisuunnikkaan piiri on ",`
	`  puolisuunnikkaanPiiri)`
	`pintaAla = sivu1 * (sivu2 + sivu4) / 2`
	`print("Puolisuunnikkaan pinta-ala on ",`
	`  pintaAla)`

I	`sivu2 = kaveleJaMittaaSivu()`
	`# sivu2 saa mitaksi 6`
	`kuljettuMatka = sivu1 + sivu2`

J	`kaannyVasemmalle(90)` `# 90 astetta`

K	`sivu3 = kaveleJaMittaaSivu()`
	`# sivu3 saa mitaksi noin 4.2`
	`kuljettuMatka = sivu1 + sivu2 + sivu3`

Tehtävä 2

Iidalla on suorakulmion muotoinen piha, jossa on neliön muotoisina piharakennuksina koirankoppi, leikkimökki ja pihavarasto. Koirankopin seinän pituus on 1 m, leikkimökin 2 m ja pihavaraston 3 m. Lisäksi pihalla on asuinrakennus, jonka pinta-ala on 150 m². Pihan sivujen mitat ovat 30 m ja 50 m. Tonttiin kuuluu myös pihan ulkopuolella oleva kolmion muotoinen pelto, jonka kannan pituus on pihan lyhyemmän sivun mitta ja korkeus on 20 m. Iida yrittää laskea kuinka suuri on pihan nurmialueen ja pellon pinta-ala yhteensä. Ratkaisukoodissa esiintyy kuitenkin virheitä.

a) Piirrä periaatekuva Iidan tontista rakennuksineen

b) Etsi Iidan koodista matemaattiset virheet ja korjaa ne

c) Kielennä Iidan ratkaisu omin sanoin (voit lisäksi piirtää välivaiheista kuvia)

Iidan koodi:

```
pihanPintaAla = 30 * 50
asunnonPintaAla = 150

# Neliönmuotoisten rakennusten pinta-alat
rakennustenPintaAla = 0
# Käy läpi luvut 1, 2 ja 3
for i in range(1,3):
      rakennustenPintaAla = rakennustenPintaAla + i*i

pellonPintaAla = 30 * 20
tontinPintaAla = pihanPintaAla + pellonPintaAla

kysyttyPintaAla = tontinPintaAla + rakennustenPintaAla
                  + asunnonPintaAla
```

RATKAISUSI TEHTÄVÄÄN 2:

Tehtävä 3

a) Suunnittele alla olevaan koodiin sopiva geometrian tehtävänanto. Voit myös halutessasi lisätä koodiin laskutoimituksia (ota tämä huomioon tekemässäsi tehtävänannossa).

b) Lisää kommentteja koodin kohtiin, joissa on valmiiksi #-merkki. Kommenttien tulee liittyä matematiikkaan ja/tai koodin toimintaan.

c) Kielennä ratkaisu (voit lisäksi havainnollistaa kirjoitusta piirroksilla).

Koodi:

```
#
luku1 = 12
luku2 = 6
luku3 = 4


#
summa = luku1 + luku1 + luku2 + luku2


#
tulo = luku2 * luku3
```

RATKAISUSI a)- ja c)-KOHDAT TEHTÄVÄÄN 3:

Liite 2(5)

Tehtävä 4

Kielennä alla oleva koodi ja kerro mitä ohjelma tekee. Voit myös havainnollistaa ratkaisuasi piirroksilla.

Koodi:

```
# Neliön sivu ja neliön sisään piirretyn ympyrän
# halkaisija
nelionSivu = 4
ympyranHalkaisija = 4

# Neliön ja ympyrän pinta-alat
# math.pi vastaa piin arvoa 3,1415...
nelionAla = nelionSivu * nelionSivu
ympyranAla = math.pi
        * (ympyranHalkaisija / 2)
        * (ympyranHalkaisija / 2)

# Ehto, erotus ja tulostus
if nelionAla > ympyranAla:
    erotusAla = nelionAla - ympyranAla
    # math.sqrt() laskee neliöjuuren
    erotusSivu = math.sqrt(erotusAla)
    print("Erotetun pinta-alan suuruus on ",
          erotusAla,". Jos erotettu pinta-ala olisi
              neliö, sen sivun pituus olisi ",
                  erotusSivu,".")
else:
    print("Virhe!")
```

RATKAISUSI TEHTÄVÄÄN 4:

if-osa:

else-osa:

Kyselylomake – vaihe 1 / 3

Taustatiedot. Vastaa seuraaviin kohtiin merkitsemällä rasti sopivaan ruutuun. Vastaa mahdollisiin tarkennuksiin kirjoittamalla vastausvaihtoehdon jäljessä olevalle viivalle.

1. Sukupuoli: ☐ nainen ☐ mies

2. Koulutustaso:
☐ yläkoulu, vuosiluokka: _____
☐ lukion lyhyt matematiikka, vuosikurssi: _____
☐ lukion pitkä matematiikka, vuosikurssi: _____

3. Matematiikan arvosana (viimeisimmässä todistuksessa):
☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐ 8 ☐ 9 ☐ 10

4. Ohjelmointikokemus:

a) Oletko ohjelmoinut aiemmin?
☐ kyllä ☐ en

b) Oletko tietotekniikkakerhossa tai koulun tietotekniikkakurssilla?
☐ kyllä ☐ en

c) Mitä ohjelmointikieliä olet käyttänyt?

_____

_____

d) Kuinka monta vuotta olet harrastanut ohjelmointia?

_____

e) Mikä on ohjelmointiharrastuksesi taajuus?
☐ alle 4 kertaa kuukaudessa ☐ 3-4 kertaa viikossa
☐ 1-2 kertaa viikossa ☐ yli 4 kertaa viikossa

f) Mikä on ohjelmointiharrastuksesi kerta-aika?
☐ alle 30 min / kerta ☐ 1 - 2 h / kerta
☐ 30 min - 1 h / kerta ☐ yli 2 h / kerta

Kyselylomake – vaihe 2 / 3

Väittämät. Vastaa seuraaviin taulukon väittämiin merkitsemällä rasti sopivaan ruutuun.

1 = Täysin eri mieltä
2 = Jokseenkin eri mieltä
3 = Jokseenkin samaa mieltä
4 = Täysin samaa mieltä

Väittämä		1	2	3	4
5.	Tehtävien ohjelmointikoodin ymmärtäminen on minulle helppoa.				
6.	Matemaattisten osuuksien löytäminen ohjelmointikoodista on minulle helppoa.				
7.	Matemaattisten osuuksien ymmärtäminen ohjelmointikoodissa on minulle helppoa.				
8.	Pidän matematiikan (geometrian) sisällyttämisestä osaksi ohjelmointikoodia.				
9.	Pidän ohjelmointia sisältävistä tehtävistä.				
10.	Ohjelmointia sisältävät tehtävät ovat mielestäni hyödyllisiä oppimisessa.				
11.	Ohjelmointikoodin toiminnan kirjoittaminen on minulle helppoa.				
12.	Ohjelmointikoodin toiminnan kirjoittaminen on mielestäni hyödyllistä oppimisessa.				
13.	Pidän ohjelmointikoodin toiminnan kirjoittamisesta.				
Väittämä		**1**	**2**	**3**	**4**
14.	Tehtävien kielentäminen on minulle tarkoitukseltaan selkeä.				
15.	Kielentäminen auttaa oman matemaattisen ajattelun ilmaisemisessa.				
16.	Kielentäminen auttaa minua käyttämään matematiikan käsitteitä.				
17.	Kielentäminen auttaa jäsentämään omaa ratkaisua.				
18.	Ratkaisun perusteleminen kirjoittamalla on minulle helppoa.				
19.	Kuvien piirtäminen ratkaisun perustelun oheen on minulle helppoa.				
20.	Pidän kielentämistä sisältävistä tehtävistä.				
21.	Kielentämistä sisältävät tehtävät ovat mielestäni hyödyllisiä oppimisessa.				

Väittämä		1	2	3	4
22.	Käytin tehtävissä loogista ajattelua.				
23.	Käytin tehtävien ratkaisemisessa apuna lomakkeen esimerkkien lisäksi muita lomakkeen tehtäviä.				
24.	Käytin tehtävissä hyväksi jakoa matemaattisiin vaiheisiin tai sääntöihin.				
25.	Käytin tehtävissä hyväksi ohjelmointikoodin jakoa vaiheisiin tai sääntöihin.				
26.	Käytin tehtävissä apuna geometrian laskujen pilkkomista pienempiin osiin.				
27.	Käytin tehtävissä apuna koodin pilkkomista pienempiin osiin.				
28.	Yritin yksinkertaistaa monimutkaisia ratkaisuja.				
29.	Hyödynsin tehtävissä aiemmin oppimaani yleistä matemaattista mallia, esimerkiksi yleisiä laskukaavoja tai samankaltaisuuksia.				
30.	Hyödynsin tehtävissä aiemmin oppimaani ohjelmoinnin mallia, esimerkiksi muita ohjelmointikieliä tai koodien keskinäisiä samankaltaisuuksia.				
31.	Hyödynsin aiempien kokemusteni ja oppimani yleistämistä tehtävien ongelmiin.				
32.	Tehtävän suunnittelu on minulle helppoa.				
33.	Tehtävän suunnittelu on mielestäni hyödyllistä.				
34.	Yritin poistaa tarpeettomia yksityiskohtia ratkaisuista. (Abstraktio)				
35.	Arvioin tehtävien omien ratkaisujeni oikeellisuutta ja täsmällisyyttä.				
36.	Tehtävien ratkaisujen arviointi on mielestäni hyödyllistä.				
37.	Tehtävien ratkaisujen arviointi on mielestäni helppoa.				

Kyselylomake – vaihe 3 / 3

38. Avoin kysymys:
Millaisia ohjelmointitehtävät olivat tasoltaan?
Olivatko jotkin tehtävien kohdat erityisen helppoja tai vaikeita, kuvaile?

39. Avoin kysymys:
Millaisia kielentämistehtävät olivat tasoltaan?
Olivatko jotkin tehtävien kohdat erityisen helppoja tai vaikeita, kuvaile?

40. Avoin kysymys:
Mitä mieltä olit ohjelmointitehtävistä?
Mitä mieltä olit ohjelmointikoodin paperitoteutuksesta?

41. Avoin kysymys:
Mitä mieltä olit kielentämistehtävistä?

42. Avoin kysymys:
Oliko kielentämisestä hyötyä (jos oli, niin millaista hyötyä)?
Voisiko kielentämistä käyttää jatkossa apuna matematiikan oppimisessa?

Kiitos vastauksistasi!

Liite 3(1). Pisteytys- ja arviointitaulukko.

Aihealueet ja kategoriat	Pisteytykset ja perustelut tehtävän maksimipisteisiin.			
	Tehtävä 1: 0-28 p	Tehtävä 2: 0-34 p	Tehtävä 3: 0-29 p	Tehtävä 4: 0-30 p
ALGORITMINEN AJATTELU	Pisteet aihealueesta: 0-8	Pisteet aihealueesta: 0-15	Pisteet aihealueesta: 0-10	Pisteet aihealueesta: 0-9
Looginen päättely	Pisteitä 0-2. Täydet pisteet: - Koodipalojen järjestyksen pitää olla oikea tai lähes oikea (kääntymisen, tekstin tulostuksen ja sivun mittauksen koodipaloilla saa olla pientä järjestyksen eroa, kunhan eteneminen on tehty tehtävän piirroksen mukaan mielekkäästi) - Koodipalojen D ja A virheellisestä järjestyksestä tai kahden A-palan oikeasta käytöstä ei vähennetä pisteitä - Yksittäisen koodipalan pienestä epäloogisuudesta ei vähennetä pisteitä, jos ratkaisu on silti järkevä ja perusteltu	Pisteitä 0-2. Täydet pisteet: - Loogisesti etenevä ja ansioitunut ratkaisu: koodi on käsitelty kokonaisuudessaan loogisesti ja perustellen, ymmärretty yhteys matematiikkaan	Pisteitä 0-2. Täydet pisteet: - Loogisesti etenevä ja ansioitunut ratkaisu: koodi on käsitelty kokonaisuudessaan loogisesti ja perustellen, ymmärretty yhteys matematiikkaan	Pisteitä 0-2. Täydet pisteet: - Loogisesti etenevä ja ansioitunut ratkaisu: koodi on käsitelty kokonaisuudessaan loogisesti ja perustellen, ymmärretty yhteys matematiikkaan
Looginen päättely kuva	-	Pisteitä 0-2. Täydet pisteet: - Kuva sisältää kaikki pyydetyt osat ja osat ovat oikein - Kuvassa on myös muita merkintöjä ja perusteluja, jotka täydentävät kuvaa suhteessa ratkaisuun/tehtävänantoon - Virhettä ei ole otettu, jos kappaleiden muodot eivät ole aivan tarkat tai kuvassa esiintyy yksittäinen todella pieni puute	-	-
Virheiden löytäminen	-	Pisteitä 0-2. Täydet pisteet: - Löydetty oikeat virheet ja merkitty ne selkeästi perustellen - Pisteitä ei ole vähennetty, jos ratkaisuun on merkitty ylimääräisiä virheitä, jotka liittyvät vain koodin syntaksiin	-	-
Virheiden korjaaminen	-	Pisteitä 0-2. Täydet pisteet: - Löydetyt oikeat virheet on korjattu oikein ja perusteltu ansiokkaasti	-	-
Analysointi/ arviointi	Pisteitä 0-2. Täydet pisteet: - Ratkaisua on yritetty perustella kokonaisuudessaan ja erityisen hyvin matematiikan ja/tai ohjelmointikoodin osalta	Pisteitä 0-2. Täydet pisteet: - Ratkaisua on yritetty perustella kokonaisuudessaan ja erityisen hyvin matematiikan ja/tai ohjelmointikoodin osalta	Pisteitä 0-2. Täydet pisteet: - Ratkaisua on yritetty perustella kokonaisuudessaan ja erityisen hyvin matematiikan ja/tai ohjelmointikoodin osalta	Pisteitä 0-2. Täydet pisteet: - Ratkaisua on yritetty perustella kokonaisuudessaan ja erityisen hyvin matematiikan ja/tai ohjelmointikoodin osalta
Hajottaminen osiin (esim. koodin), algoritmit, vaiheet	Pisteitä 0-2. Täydet pisteet: - Koodipalojen järjestystä ja vaiheita on käsitelty hyvin. - Ratkaisun kirjallisesta osuudesta ilmenee,	Pisteitä 0-3. Täydet pisteet: - Ratkaisun kirjallisesta osuudesta ilmenee, että ongelma on osattu jakaa osiin. - Ratkaisussa esiintyy tehtävänannon	Pisteitä 0-2. Täydet pisteet: - Ratkaisun kirjallisesta osuudesta ilmenee, että ongelma on osattu jakaa osiin. - Koodia ja matematiikkaa on hajotettu	Pisteitä 0-3. Täydet pisteet: - Ratkaisun kirjallisesta osuudesta ilmenee, että ongelmaa on yritetty jakaa, ja annettu koodin jaottelu osiin (4 kpl) on ymmärretty

	että ongelma on osattu jakaa osiin. - Koodipalojen sisäisistä vaiheista on kerrottu tarkemmin (esimerkiksi koodipalat A ja H).	ongelmaa purkava kuva - Koodia ja matematiikkaa on hajotettu ansiokkaasti osiin perustellen (esim. laskujen välivaiheita ja koodin vaiheita)	ansiokkaasti osiin perustellen (esim. laskujen välivaiheita, koodin vaiheita ja suhdetta suunniteltuun tehtävänantoon)	- Valmiiksi jaoteltuja osia on käsitelty määrällisesti ja sisällöllisesti ansiokkaasti - Ratkaisussa voi esiintyä ongelmaa purkava oikeellinen kuva - Koodia ja matematiikkaa on hajotettu ansiokkaasti osiin perustellen (esim. laskujen välivaiheita ja koodin vaiheita)
Kaavat, mallit	Pisteitä 0-2. Täydet pisteet: - Ratkaisussa on esitetty paljon laskuja perustellen. - Ratkaisussa on esitetty paljon kaavoja tai niihin liittyviä termejä perustellen. - Täydet pisteet voi saada, vaikka olisi kerrottu kaavoista ilman laskuja tai päinvastoin, mikäli ymmärrys matemaattisesta ratkaisusta (ja joissakin tapauksissa ohjelmointikoodista) on tuotu esille.	Pisteitä 0-2. Täydet pisteet: - Ratkaisussa on esitetty paljon laskuja perustellen. - Ratkaisussa on esitetty paljon kaavoja tai niihin liittyviä termejä perustellen. - Täydet pisteet voi saada, vaikka olisi kerrottu kaavoista ilman laskuja tai päinvastoin, mikäli ymmärrys matemaattisesta ratkaisusta (ja joissakin tapauksissa ohjelmointikoodista) on tuotu esille.	Pisteitä 0-2. Täydet pisteet: - Ratkaisussa on esitetty paljon laskuja perustellen. - Ratkaisussa on esitetty paljon kaavoja tai niihin liittyviä termejä perustellen (huomioidaan myös suunniteltu tehtävänanto). - Täydet pisteet voi saada, vaikka olisi kerrottu kaavoista ilman laskuja tai päinvastoin, mikäli ymmärrys matemaattisesta ratkaisusta (ja joissakin tapauksissa ohjelmointikoodista) on tuotu esille.	Pisteitä 0-2. Täydet pisteet: - Ratkaisussa on esitetty paljon laskuja perustellen. - Ratkaisussa on esitetty paljon kaavoja tai niihin liittyviä termejä perustellen. - Täydet pisteet voi saada, vaikka olisi kerrottu kaavoista ilman laskuja tai päinvastoin, mikäli ymmärrys matemaattisesta ratkaisusta (ja joissakin tapauksissa ohjelmointikoodista) on tuotu esille.
Suunnittelu/ luovuus	-	-	Pisteitä 0-2. Täydet pisteet: - Ohjelmointikoodin mukaan on suunniteltu ansioitunut tehtävänanto, joka liittyy selkeästi geometriaan - Suunniteltu tehtävänanto on oikeellinen suhteessa valmiiseen (ja mahdollisesti lisättyyn) koodiin ja matematiikkaan - (Tehtävänannossa ja ohjelmointikoodin lisäyksissä on käytetty luovuutta ja mahdollisia kuvia)	-
OHJELMOINTI	Pisteet aihealueesta: 0-8	Pisteet aihealueesta: 0-6	Pisteet aihealueesta: 0-7	Pisteet aihealueesta: 0-8
Muuttujat/lausekkeet/ lauseet	Pisteitä 0-2. Täydet pisteet: - Ratkaisusta ilmenee muuttujien merkityksen ymmärtäminen (esimerkiksi nimien ymmärtäminen, muuttujaan sijoitetun arvon ymmärtäminen). - Summamuuttujan riittävä ymmärtäminen (summamuuttuja kuljettuMatka kerryttää itseensä lisää arvoa sivujen pituuksien mukaan). - Ratkaisusta ilmenee ohjelmointikoodin lauseiden hyvä ymmärtäminen (esimerkiksi laskujen ymmärtäminen). - (Muuttujan termiä on osattu käyttää kopioiden tai itsenäisesti hyvin).	Pisteitä 0-2. Täydet pisteet: - Ratkaisusta ilmenee muuttujien merkityksen ymmärtäminen (esimerkiksi nimien ymmärtäminen, muuttujaan sijoitetun arvon ymmärtäminen). - Summamuuttujan riittävä ymmärtäminen - Ratkaisusta ilmenee ohjelmointikoodin lauseiden hyvä ymmärtäminen (esimerkiksi laskujen ymmärtäminen). - (Muuttujan termiä on osattu käyttää kopioiden tai itsenäisesti hyvin).	Pisteitä 0-2. Täydet pisteet: - Ratkaisusta ilmenee muuttujien merkityksen ymmärtäminen (esimerkiksi nimien ymmärtäminen, muuttujaan sijoitetun arvon ymmärtäminen) - Ratkaisusta ilmenee ohjelmointikoodin lauseiden hyvä ymmärtäminen (esimerkiksi laskujen ymmärtäminen) - (Muuttujan termiä on osattu käyttää kopioiden tai itsenäisesti hyvin) - Mahdolliset koodiin lisätyt muuttujat ja lauseet on tehty ansiokkaasti ja niitä on käytetty oikein tai lähes oikein	Pisteitä 0-2. Täydet pisteet: - Ratkaisusta ilmenee muuttujien merkityksen ymmärtäminen (esimerkiksi nimien ymmärtäminen, muuttujaan sijoitetun arvon ymmärtäminen) - Ratkaisusta ilmenee ohjelmointikoodin lauseiden hyvä ymmärtäminen (esimerkiksi laskujen ymmärtäminen) - (Muuttujan termiä on osattu käyttää kopioiden tai itsenäisesti hyvin)
Funktiot	Pisteitä 0-2. Täydet pisteet: - Funktioiden ymmärtäminen ja ymmärtämisen ilmaiseminen kirjallisesti	-	Pisteitä 0-1. Täydet pisteet: - Valmiiseen koodiin on lisätty koodia, joka sisältää funktion rakenteen	Pisteitä 0-2. Täydet pisteet: - Funktioiden ymmärtäminen ja ymmärtämisen ilmaiseminen kirjallisesti

	(kääntymisfunktio, sivujen pituuksia mittaava funktio, tulostusfunktio).		- Funktio on tehty koodiin oikein tai lähes oikein	(tulostusfunktio) - Rakenteiden math.pi ja math.sqrt() käsittely kielnetämisessä, vaikka molemmat rakenteet eivät ole funktioita.
Ehtolauseet	Pisteitä 0-2. Täydet pisteet: - Ehtolausetta on käytetty oikein ja oikeissa kohdissa tehtävän koodipaloja järjesteltäessä (koodipalan A if-else-rakenne). - Ehto on rakenteena ymmärretty ja perusteltu kirjallisesti.	-	Pisteitä 0-1. Täydet pisteet: - Valmiiseen koodiin on lisätty koodia, joka sisältää ehtorakenteen - Ehtolause on tehty koodiin oikein tai lähes oikein	Pisteitä 0-2. Täydet pisteet: - Ehto on rakenteena ymmärretty ja perusteltu kirjallisesti - Sekä if-osa että else-osa on ymmärretty oikein tehtävän ja ohjelmointikoodin suhteen (mikä on if-osan ehto, milloin päädytään else-osaan ja millä perusteella)
Silmukat	-	Pisteitä 0-2. Täydet pisteet: - Silmukkaa on käsitelty ansiokkaasti ja se on ymmärretty ratkaisun kannalta riittävällä tasolla - Silmukasta on ymmärretty riittävällä tasolla silmukkamuuttuja *i* ja/tai rakennustenPintaAla-summamuuttuja	Pisteitä 0-1. Täydet pisteet: - Valmiiseen koodiin on lisätty koodia, joka sisältää silmukan rakenteen - Silmukka on tehty koodiin oikein tai lähes oikein	-
Kommentit	Pisteitä 0-2. Täydet pisteet: - Kommenttien sisältö on ymmärretty ja otettu kirjallisessa perustelussa huomioon. - Täysiin pisteisiin ei vaadita kommentin tarkoituksen ilmaisemista (”ohjelma ei suorita kommenttia” tms.), vaan sen sisällön ymmärtäminen ja ilmaiseminen riittävät.	Pisteitä 0-2. Täydet pisteet: - Kommenttien sisältö on ymmärretty ja otettu kirjallisessa perustelussa huomioon. - Täysiin pisteisiin ei vaadita kommentin tarkoituksen ilmaisemista. - Erityisesti kommenteista ymmärrettävä neliönmuotoisten rakennusten käsittely	Pisteitä 0-2. Täydet pisteet: - Kaikki tehtävän kommenttikohdat on täytetty koodiin tai esitetty kielentämis-tekstissä selkeästi kommentteihin viitaten - Kommentit liittyvät koodiin ja/tai matematiikkaan ja ovat tehtävän kannalta järkeviä/hyödyllisiä - Täysiin pisteisiin ei vaadita kommentin tarkoituksen ilmaisemista. - (Mahdolliset omat kommentit)	Pisteitä 0-2. Täydet pisteet: - Kommenttien sisältö on ymmärretty ja otettu kirjallisessa perustelussa huomioon ts. kommenttien sisältöä on käsitelty ja hyödynnetty ansiokkaasti. - Täysiin pisteisiin ei vaadita kommentin tarkoituksen ilmaisemista.
KIELENTÄMINEN	Pisteet aihealueesta: 0-12	Pisteet aihealueesta: 0-13	Pisteet aihealueesta: 0-12	Pisteet aihealueesta: 0-13
Ilmaistu perusteluja/ matemaattista ajattelua/ käytetty kielentämistä/ käytetty piirroksia/ tehty mielekäs tehtävänanto:	Pisteet 0-2. Täydet pisteet: - Matemaattinen ajattelu on ilmaistu selkeästi ja johdonmukaisesti - Perusteluja on kirjallisesti paljon ja ne on esitetty tehtävän kannalta mielekkäästi - Perustelujen ohessa käytetty mahdollisesti havainnollisia piirroksia	Pisteitä 0-3. Täydet pisteet: - Matemaattinen ajattelu on ilmaistu selkeästi ja johdonmukaisesti - Perusteluja on kirjallisesti paljon ja ne on esitetty tehtävän kannalta mielekkäästi - Perustelujen ohessa käytetty mahdollisesti havainnollistavia piirroksia - Virhettä ei ole otettu, mikäli on kielennetty alkuperäinen virheellinen ratkaisu, jos koodin virheet on perusteltu	Pisteitä 0-3. Täydet pisteet: - Matemaattinen ajattelu on ilmaistu selkeästi ja johdonmukaisesti - Perusteluja on kirjallisesti paljon ja ne on esitetty tehtävän kannalta mielekkäästi - Perustelujen ohessa käytetty mahdollisesti havainnollistavia piirroksia	Pisteitä 0-3. Täydet pisteet: - Matemaattinen ajattelu on ilmaistu selkeästi ja johdonmukaisesti - Perusteluja on kirjallisesti paljon ja ne on esitetty tehtävän kannalta mielekkäästi - Perustelujen ohessa käytetty mahdollisesti havainnollistavia piirroksia
Matematiikan löytäminen ja ymmärtäminen ohjelmointikoodista	Pisteet 0-2. Täydet pisteet: - Ohjelmointikoodissa olevat matemaattiset toimenpiteet ja laskut on kirjattu kielentäen - Ratkaisusta ilmenee, että matematiikka on ymmärretty ansiokkaasti	Pisteitä 0-2. Täydet pisteet: - Ohjelmointikoodissa olevat matemaattiset toimenpiteet ja laskut on kirjattu kielentäen - Ratkaisusta ilmenee, että matematiikka on ymmärretty ansiokkaasti	Pisteitä 0-2. Täydet pisteet: - Ohjelmointikoodissa olevat matemaattiset toimenpiteet ja laskut on kirjattu kielentäen - Ratkaisusta ilmenee, että matematiikka on ymmärretty ansiokkaasti	Pisteitä 0-2. Täydet pisteet: - Ohjelmointikoodissa olevat matemaattiset toimenpiteet ja laskut on kirjattu kielentäen - Ratkaisusta ilmenee, että matematiikka on ymmärretty ansiokkaasti
Ohjelmoinnin käsitteet	Pisteet 0-2. Täydet pisteet: - Käytetty ohjelmoinnin käsitteitä - Itsenäinen ja oikeellinen termien käyttö	Pisteitä 0-2. Täydet pisteet: - Käytetty ohjelmoinnin käsitteitä - Itsenäinen ja oikeellinen termien käyttö	Pisteitä 0-1. Täydet pisteet: - Käytetty ohjelmoinnin käsitteitä - Itsenäinen ja oikeellinen termien käyttö	Pisteitä 0-2. Täydet pisteet: - Käytetty ohjelmoinnin käsitteitä - Itsenäinen ja oikeellinen termien käyttö

Matemaattiset käsitteitä	Pisteet 0-2. Täydet pisteet: - Käytetty matemaattisia käsitteitä paljon ja ansiokkaasti - Itsenäinen ja oikeellinen termien käyttö	Pisteitä 0-2. Täydet pisteet: - Käytetty matemaattisia käsitteitä paljon ja ansiokkaasti - Itsenäinen ja oikeellinen termien käyttö	Pisteitä 0-2. Täydet pisteet: - Käytetty matemaattisia käsitteitä paljon ja ansiokkaasti - Itsenäinen ja oikeellinen termien käyttö	Pisteitä 0-2. Täydet pisteet: - Käytetty matemaattisia käsitteitä paljon ja ansiokkaasti - Itsenäinen ja oikeellinen termien käyttö
Matemaattiset kaavat	Pisteet 0-2. Täydet pisteet: - Käytetty ja selitetty matemaattiset kaavat suurelta osin ansiokkaasti - Täysiin pisteisiin riittää, että kaavat on selitetty symboleilla tai käsitteillä tai laskujen ja kaavaperustelujen yhdistelmänä	Pisteitä 0-2. Täydet pisteet: - Käytetty ja selitetty matemaattiset kaavat suurelta osin ansiokkaasti - Täysiin pisteisiin riittää, että kaavat on selitetty symboleilla tai käsitteillä tai laskujen ja kaavaperustelujen yhdistelmänä	Pisteitä 0-2. Täydet pisteet: - Käytetty ja selitetty matemaattiset kaavat suurelta osin ansiokkaasti - Täysiin pisteisiin riittää, että kaavat on selitetty symboleilla tai käsitteillä tai laskujen ja kaavaperustelujen yhdistelmänä	Pisteitä 0-2. Täydet pisteet: - Käytetty ja selitetty matemaattiset kaavat suurelta osin ansiokkaasti - Täysiin pisteisiin riittää, että kaavat on selitetty symboleilla tai käsitteillä tai laskujen ja kaavaperustelujen yhdistelmänä
Kielentämällä toteutettu jäsennelty/ loogisesti etenevä ratkaisu	Pisteet 0-2. Täydet pisteet: - Ratkaisu etenee loogisesti ja ratkaisun johdonmukaisuus on erittäin selkeä myös lukijalle - Ratkaisu on ymmärrettävä ilman lisäinformaatiota	Pisteitä 0-2. Täydet pisteet: - Ratkaisu etenee loogisesti ja ratkaisun johdonmukaisuus on erittäin selkeä myös lukijalle - Ratkaisu on ymmärrettävä ilman lisäinformaatiota	Pisteitä 0-2. Täydet pisteet: - Ratkaisu etenee loogisesti ja ratkaisun johdonmukaisuus on erittäin selkeä myös lukijalle - Ratkaisu on ymmärrettävä ilman lisäinformaatiota	Pisteitä 0-2. Täydet pisteet: - Ratkaisu etenee loogisesti ja ratkaisun johdonmukaisuus on erittäin selkeä myös lukijalle - Ratkaisu on ymmärrettävä ilman lisäinformaatiota

Liite 4(1). Tutkittavien taustatiedot kyselylomakkeen vastausten perusteella.

	n	% kaikista	% vastanneista
Sukupuoli			
Naiset	31	44,9	44,9
Miehet	34	49,3	49,3
Muut	4	5,8	5,8
Yhteensä	69	100,0	100,0
Koulutustaso			
7. lk	15	21,7	21,7
8. lk	20	29,0	29,0
9. lk	27	39,1	39,1
Lukio	7	10,1	10,1
Yhteensä	69	100,0	100,0
Matematiikan arvosana			
5	4	5,8	6,5
6	6	8,7	9,7
7	8	11,6	12,9
8	17	24,6	27,4
9	19	27,5	30,6
10	8	11,6	12,9
Yhteensä	62	89,9	100,0
Aiempi ohjelmointikokemus			
Ei ohjelmoinut	36	52,2	55,4
Ohjelmoinut vähän	13	18,8	20,0
Ohjelmoinut paljon	16	23,2	24,6
Yhteensä	65	94,2	100,0
Ohjelmoinut runsaasti	8	11,6	
Tietotekniikkakerho tai -kurssi			
Ei	38	55,1	58,5
Kyllä	27	39,1	41,5
Yhteensä	65	94,2	100,0

Liite 5(1). Väitekysymysten tulokset.

1 = Täysin eri mieltä
2 = Jokseenkin eri mieltä
3 = Jokseenkin samaa mieltä
4 = Täysin samaa mieltä

	Väittämä	1	(1,5)	2	(2,5)	3	(3,5)	4	Yhteensä	Moodi	Md
Ohjelmointi											
5.	Tehtävien ohjelmointikoodin ymmärtäminen on minulle helppoa.	19 (30,6 %)	-	27 (43,5 %)	2 (3,2 %)	9 (14,5 %)	-	5 (8,1 %)	62	2	2
6.	Matemaattisten osuuksien löytäminen ohjelmointikoodista on minulle helppoa.	13 (21,0 %)	-	20 (32,3 %)	1 (1,6 %)	20 (32,3 %)	1 (1,6 %)	7 (11,3 %)	62	2 ja 3	2
7.	Matemaattisten osuuksien ymmärtäminen ohjelmointikoodissa on minulle helppoa.	16 (25,4 %)	-	25 (39,7 %)	3 (4,8 %)	10 (15,9 %)	-	9 (14,3 %)	63	2	2
8.	Pidän matematiikan (geometrian) sisällyttämisestä osaksi ohjelmointikoodia.	21 (33,9 %)	1 (1,6 %)	16 (25,8 %)	-	20 (32,3 %)	-	4 (6,5 %)	62	1	2
9.	Pidän ohjelmointia sisältävistä tehtävistä.	24 (39,3 %)	1 (1,6 %)	15 (24,6 %)	-	20 (32,8 %)	-	1 (1,6 %)	61	1	2
10.	Ohjelmointia sisältävät tehtävät ovat mielestäni hyödyllisiä oppimisessa.	14 (22,6 %)	-	19 (30,6 %)	4 (6,5 %)	20 (32,3 %)	-	5 (8,1 %)	62	3	2
11.	Ohjelmointikoodin toiminnan kirjoittaminen on minulle helppoa.	24 (39,3 %)	-	21 (34,4 %)	-	14 (23,0 %)	-	2 (3,3 %)	61	1	2
12.	Ohjelmointikoodin toiminnan kirjoittaminen on mielestäni hyödyllistä oppimisessa.	10 (16,1 %)	-	26 (41,9 %)	1 (1,6 %)	22 (35,5 %)	-	3 (4,8 %)	62	2	2
13.	Pidän ohjelmointikoodin toiminnan kirjoittamisesta.	24 (39,3 %)	-	19 (31,1 %)	-	11 (18,0 %)	-	7 (11,5 %)	61	1	2
Kielentäminen											
14.	Tehtävien kielentäminen on minulle tarkoitukseltaan selkeä.	17 (27,9 %)	-	18 (29,5 %)	3 (4,9 %)	15 (24,6 %)	-	8 (13,1 %)	61	2	2
15.	Kielentäminen auttaa oman matemaattisen ajattelun ilmaisemisessa.	14 (22,6 %)	-	21 (33,9 %)	2 (3,2 %)	18 (29,0 %)	1 (1,6 %)	6 (9,7 %)	62	2	2
16.	Kielentäminen auttaa minua käyttämään matematiikan käsitteitä.	14 (22,6 %)	-	28 (45,2 %)	1 (1,6 %)	13 (21,0 %)	-	6 (9,7 %)	62	2	2
17.	Kielentäminen auttaa jäsentämään omaa ratkaisua.	14 (23,3 %)	1 (1,7 %)	19 (31,7 %)	1 (1,7 %)	15 (25,0 %)	-	10 (16,7 %)	60	2	2
18.	Ratkaisun perusteleminen kirjoittamalla on minulle helppoa.	14 (22,6 %)	1 (1,6 %)	24 (38,7 %)	-	18 (29,0 %)	-	5 (8,1 %)	62	2	2
19.	Kuvien piirtäminen ratkaisun perustelun oheen on minulle helppoa.	9 (14,5 %)	-	19 (30,6 %)	1 (1,6 %)	25 (40,3 %)	2 (3,2 %)	6 (9,7 %)	62	3	3
20.	Pidän kielentämistä sisältävistä tehtävistä.	24 (38,7 %)	-	18 (29,0 %)	4 (6,5 %)	13 (21,0 %)	1 (1,6 %)	2 (3,2 %)	62	1	2
21.	Kielentämistä sisältävät tehtävät ovat mielestäni hyödyllisiä oppimisessa.	11 (17,7 %)	-	17 (27,4 %)	2 (3,2 %)	26 (41,9 %)	-	6 (9,7 %)	62	3	3
Algoritminen ajattelu											
22.	Käytin tehtävissä loogista ajattelua.	9 (14,8 %)	1 (1,6 %)	9 (14,8 %)	-	24 (39,3 %)	-	18 (29,5 %)	61	3	3
23.	Käytin tehtävien ratkaisemisessa apuna lomakkeen esimerkkien lisäksi muita lomakkeen tehtäviä.	24 (39,3 %)	-	18 (29,5 %)	1 (1,6 %)	12 (19,7 %)	-	6 (9,8 %)	61	1	2

24.	Käytin tehtävissä hyväksi jakoa matemaattisiin vaiheisiin tai sääntöihin.	12 (20,0 %)	-	19 (31,7 %)	-	19 (31,7 %)	-	10 (16,7 %)	60	2 ja 3	2
25.	Käytin tehtävissä hyväksi ohjelmointikoodin jakoa vaiheisiin tai sääntöihin.	19 (32,8 %)	-	17 (29,3 %)	4 (6,9 %)	10 (17,2 %)	-	8 (13,8 %)	58	1	2
26.	Käytin tehtävissä apuna geometrian laskujen pilkkomista pienempiin osiin.	19 (31,7 %)	-	17 (28,3 %)	2 (3,3 %)	17 (28,3 %)	-	5 (8,3 %)	60	1	2
27.	Käytin tehtävissä apuna koodin pilkkomista pienempiin osiin.	19 (32,2 %)	-	22 (37,3 %)	3 (5,1 %)	10 (16,9 %)	-	5 (8,5 %)	59	2	2
28.	Yritin yksinkertaistaa monimutkaisia ratkaisuja.	9 (14,8 %)	-	13 (21,3 %)	1 (1,6 %)	25 (41,0 %)	-	13 (21,3 %)	61	3	3
29.	Hyödynsin tehtävissä aiemmin oppimaani yleistä matemaattista mallia, esimerkiksi yleisiä laskukaavoja tai samankaltaisuuksia.	10 (16,4 %)	-	15 (24,6 %)	-	22 (36,1 %)	-	14 (23,0%)	61	3	3
30.	Hyödynsin tehtävissä aiemmin oppimaani ohjelmoinnin mallia, esimerkiksi muita ohjelmointikieliä tai koodien keskinäisiä samankaltaisuuksia.	26 (45,6 %)	1 (1,8 %)	14 (24,6 %)	1 (1,8 %)	10 (17,5 %)	-	5 (8,8 %)	57	1	2
31.	Hyödynsin aiempien kokemusteni ja oppimani yleistämistä tehtävien ongelmiin.	12 (20,0 %)	-	17 (28,3 %)	1 (1,7 %)	20 (33,3 %)	-	10 (16,7 %)	60	3	2,75
32.	Tehtävän suunnittelu on minulle helppoa.	20 (33,3 %)	2 (3,3 %)	20 (33,3 %)	2 (3,3 %)	14 (23,3 %)	-	2 (3,3 %)	60	1 ja 2	2
33.	Tehtävän suunnittelu on mielestäni hyödyllistä.	10 (16,9 %)	-	18 (30,5 %)	2 (3,4 %)	28 (47,5 %)	-	1 (1,7 %)	59	3	2,5
34.	Yritin poistaa tarpeettomia yksityiskohtia ratkaisuista. (Abstraktio)	18 (30,5 %)	-	14 (23,7 %)	3 (5,1 %)	18 (30,5 %)	-	6 (10,2 %)	59	1 ja 3	2
35.	Arvioin tehtävien omien ratkaisujeni oikeellisuutta ja täsmällisyyttä.	13 (21,7 %)	-	20 (33,3 %)	1 (1,7 %)	18 (30,0 %)	1 (1,7 %)	7 (11,7 %)	60	2	2
36.	Tehtävien ratkaisujen arviointi on mielestäni hyödyllistä.	10 (17,2 %)	-	14 (24,1 %)	2 (3,4 %)	22 (37,9 %)	-	10 (17,2 %)	58	3	3
37.	Tehtävien ratkaisujen arviointi on mielestäni helppoa.	16 (27,6 %)	-	22 (37,9 %)	3 (5,2 %)	13 (22,4 %)	-	4 (6,9 %)	58	2	2

Liite 6(1). Tehtäväkohtaisten aihealueiden ja kategorioiden pisteet

Aihealue / Kategoria	Tehtävä 1 (n = 62)		Tehtävä 2 (n = 55)		Tehtävä 3 (n = 44)		Tehtävä 4 (n = 44)	
	Pisteytys	ka / ka-% (SD)	Pisteytys	ka / ka-% (SD)	Pisteytys	ka / ka-% (SD)	Pisteytys	ka / ka-% (SD)
Algoritminen ajattelu	**0-100 %**	**35,1 (33,4)**	**0-100 %**	**45,9 (35,3)**	**0-100 %**	**41,1 (36,0)**	**0-100 %**	**47,5 (39,5)**
Virheiden etsiminen	-	-	0-2 p	0,87 (0,90)	-	-	-	-
Virheiden korjaaminen	-	-	0-2 p	0,73 (0,83)	-	-	-	-
Looginen päättely	0-2 p	1,29 (0,86)	0-2 p	0,80 (0,87)	0-2 p	1,05 (0,75)	0-2 p	1,07 (0,93)
Looginen päättely (kuva)	-	-	0-2 p	1,36 (0,83)	-	-	-	-
Suunnittelu	-	-	-	-	0-2 p	0,66 (0,86)	-	-
Analysointi / arviointi	0-2 p	0,77 (0,91)	0-2 p	0,91 (0,85)	0-2 p	0,77 (0,83)	0-2 p	1,00 (0,89)
Hajottaminen osiin, algoritmit ja vaiheet	0-2 p	0,44 (0,74)	0-3 p	1,47 (1,07)	0-2 p	1,09 (0,86)	0-3 p	1,39 (1,06)
Kaavat ja mallit	0-2 p	0,31 (0,67)	0-2 p	0,75 (0,80)	0-2 p	0,55 (0,82)	0-2 p	0,82 (0,90)
Ohjelmointi	**0-100 %**	**25,2 (37,3)**	**0-100 %**	**15,8 (28,6)**	**0-100 %**	**25,0 (20,2)**	**0-100 %**	**40,6 (33,7)**
Muuttujat/lausekkeet	0-2 p	0,37 (0,71)	0-2 p	0,47 (0,74)	0-2 p	0,93 (0,73)	0-2 p	1,11 (0,78)
Silmukat	-	-	0-2 p	0,22 (0,60)	0-1 p	0,00 (0,00)	-	-
Funktiot	0-2 p	0,45 (0,72)	-	-	0-1 p	0,00 (0,00)	0-2 p	0,61 (0,69)
Ehtolauseet	0-2 p	0,56 (0,86)	-	-	0-1 p	0,00 (0,00)	0-2 p	0,66 (0,81)
Kommentit	0-2 p	0,63 (0,89)	0-2 p	0,25 (0,62)	0-2 p	0,82 (0,84)	0-2 p	0,86 (0,88)
Kielentäminen	**0-100 %**	**25,9 (35,0)**	**0-100 %**	**38,3 (31,0)**	**0-100 %**	**35,8 (32,7)**	**0-100 %**	**44,2 (35,8)**
Ilmaistu perusteluja/matemaattista ajattelua/käytetty kielentämistä/käytetty kuvia	0-2 p	0,82 (0,90)	0-3 p	1,58 (0,94)	0-3 p	1,07 (1,04)	0-3 p	1,48 (0,93)
Matematiikan löytäminen ja ymmärtäminen ohjelmointikoodista	0-2 p	0,42 (0,74)	0-2 p	0,76 (0,77)	0-2 p	0,95 (0,75)	0-2 p	0,89 (0,90)
Käytetty ohjelmoinnin käsitteitä	0-2 p	0,27 (0,63)	0-2 p	0,18 (0,48)	0-1 p	0,18 (0,45)	0-2 p	0,52 (0,73)
Käytetty matemaattisia käsitteitä	0-2 p	0,60 (0,86)	0-2 p	0,87 (0,80)	0-2 p	1,02 (0,90)	0-2 p	1,16 (0,94)
Käytetty ja selitetty matemaattiset kaavat	0-2 p	0,29 (0,64)	0-2 p	0,64 (0,75)	0-2 p	0,43 (0,73)	0-2 p	0,80 (0,88)
Kielentämällä toteutettu jäsennelty/loogisesti etenevä ratkaisu	0-2 p	0,71 (0,95)	0-2 p	0,95 (0,93)	0-2 p	0,64 (0,87)	0-2 p	0,91 (0,94)
Koko tehtävä	**0-100 %**	**28,3 (34,4)**	**0-100 %**	**37,7 (30,9)**	**0-100 %**	**33,9 (28,9)**	**0-100 %**	**44,2 (35,8)**

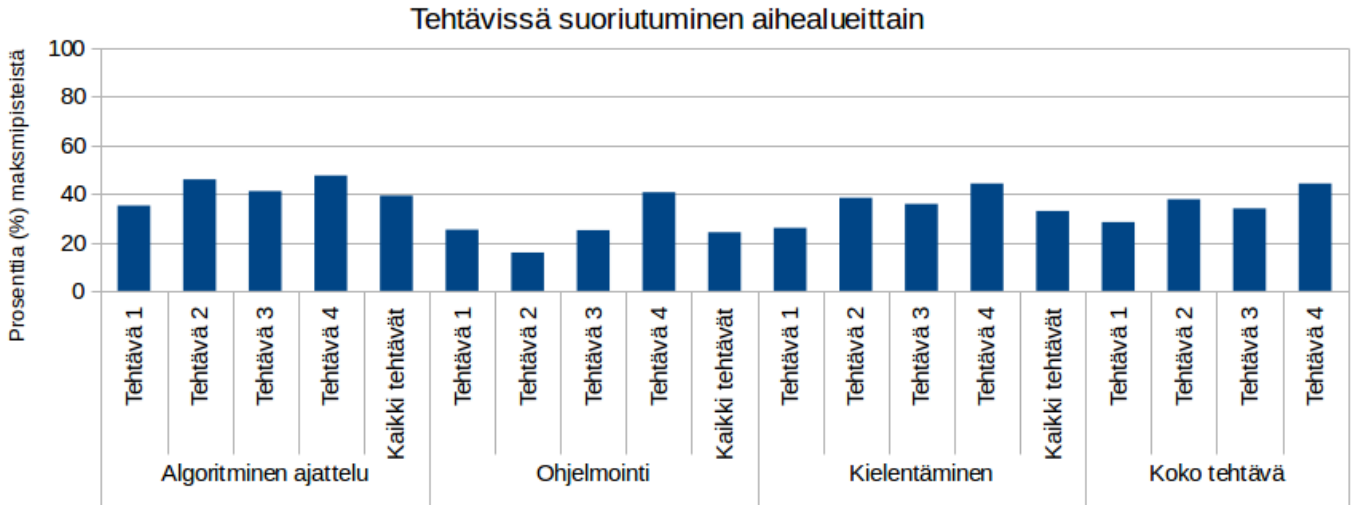
Liite 7(1). Tehtäväkohtaisten aihealueiden pisteet vertaillen eri sukupuolten, koulutusasteiden, ohjelmointitaustojen ja matematiikan arvosanojen kesken.

Tehtävä	Aihealue	Kaikki	Sukupuoli			Koulutusaste				
		ka-% (SD)	Nainen	Mies	p-arvo	7. lk	8. lk	9. lk	Lukio	p-arvo
Tehtävä 1	Algoritminen ajattelu	35,1 (33,4)	30,1 (28,6)	39,1 (36,8)	0,569	26,9 (29,2)	29,4 (32,8)	35,6 (31,2)	66,7 (43,1)	0,205
	Ohjelmointi	25,2 (37,3)	20,8 (33,8)	29,0 (39,4)	0,508	16,4 (33,2)	25,0 (36,2)	21,6 (34,9)	60,4 (47,7)	0,152
	Kielentäminen	25,9 (35,0)	21,6 (32,0)	29,6 (36,5)	0,377	15,4 (30,6)	23,5 (34,4)	26,0 (33,4)	55,6 (44,6)	0,267
	Koko tehtävä	28,3 (34,4)	23,8 (30,6)	32,1 (36,6)	0,587	19,0 (30,4)	25,6 (33,8)	27,5 (31,9)	60,1 (44,7)	0,239
Tehtävä 2	Algoritminen ajattelu	45,9 (35,3)	49,4 (30,8)	41,7 (39,6)	0,333	25,6 (24,3)	38,1 (39,0)	53,3 (32,8)	81,3 (27,2)	**0,015**
	Ohjelmointi	15,8 (28,6)	11,8 (23,3)	17,9 (31,7)	0,813	5,6 (10,9)	20,2 (35,9)	14,6 (26,2)	33,3 (42,5)	0,461
	Kielentäminen	38,3 (31,0)	39,7 (26,9)	35,6 (34,1)	0,414	23,1 (22,2)	34,1 (35,0)	41,4 (28,0)	72,3 (29,6)	**0,034**
	Koko tehtävä	37,7 (30,9)	39,1 (26,2)	35,2 (34,5)	0,289	21,1 (19,9)	33,4 (35,6)	41,9 (28,3)	69,4 (28,3)	**0,035**
Tehtävä 3	Algoritminen ajattelu	41,1 (36,0)	45,0 (37,9)	35,2 (36,0)	0,472	11,4 (12,2)	38,0 (37,5)	48,4 (34,2)	80,0 (34,6)	**0,025**
	Ohjelmointi	25,0 (20,2)	21,4 (16,4)	26,1 (23,5)	0,666	10,2 (10,8)	24,8 (22,6)	26,3 (18,0)	52,4 (8,2)	**0,031**
	Kielentäminen	35,8 (32,7)	38,0 (34,0)	31,5 (33,2)	0,558	11,9 (13,5)	32,8 (36,4)	40,8 (28,3)	75,0 (36,3)	**0,036**
	Koko tehtävä	33,9 (28,9)	35,2 (29,1)	30,4 (30,2)	0,625	11,0 (11,2)	31,6 (30,9)	38,6 (26,2)	68,9 (28,0)	**0,025**
Tehtävä 4	Algoritminen ajattelu	47,5 (39,5)	51,9 (42,6)	43,1 (37,9)	0,583	7,9 (10,6)	55,6 (40,8)	51,1 (37,4)	68,9 (42,6)	**0,036**
	Ohjelmointi	40,6 (33,7)	43,1 (31,0)	38,0 (36,4)	0,542	12,5 (10,2)	45,8 (37,8)	41,9 (31,2)	62,5 (37,5)	0,102
	Kielentäminen	44,2 (35,8)	47,0 (36,1)	41,5 (36,8)	0,595	9,9 (7,3)	52,6 (41,6)	45,4 (30,5)	67,7 (39,7)	0,051
	Koko tehtävä	44,2 (35,8)	47,4 (36,1)	41,1 (36,4)	0,467	10,0 (7,7)	51,7 (39,8)	46,2 (32,0)	66,7 (39,7)	0,079
Kaikki tehtävät	Algoritminen ajattelu	39,2 (32,6)	41,2 (32,3)	36,3 (33,6)	0,444	22,1 (21,8)	34,8 (35,7)	44,3 (28,6)	68,4 (39,5)	**0,044**
	Ohjelmointi	24,1 (26,3)	22,4 (24,2)	25,0 (28,3)	0,914	11,4 (18,6)	25,3 (28,8)	24,0 (23,3)	48,9 (32,5)	0,064
	Kielentäminen	32,9 (30,7)	34,1 (30,2)	30,9 (31,8)	0,556	17,0 (19,8)	30,9 (35,3)	35,6 (25,6)	61,5 (38,6)	0,057

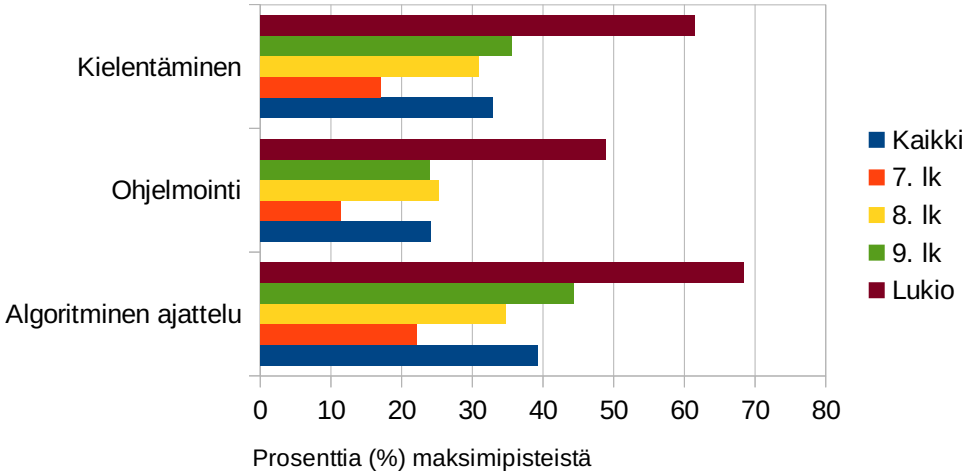
Liite 7(2)

Tehtävä (n)	Aihealue	Aiempi ohjelmointikokemus				Matematiikan arvosana						
		Ei	Vähän	Paljon	p-arvo	5	6	7	8	9	10	p-arvo
Tehtävä 1	Algoritminen ajattelu	36,0 (31,2)	25,0 (34,8)	39,2 (37,5)	0,414	12,5 (25,0)	15,0 (16,3)	30,4 (43,8)	27,3 (22,0)	41,2 (32,7)	75,0 (31,5)	**0,013**
	Ohjelmointi	26,1 (36,2)	17,5 (36,9)	28,3 (39,9)	0,653	0,0 (0,0)	7,5 (16,8)	26,8 (45,9)	17,2 (27,3)	30,9 (40,0)	62,5 (40,8)	**0,037**
	Kielentäminen	26,8 (34,7)	19,2 (38,5)	28,3 (33,0)	0,539	8,3 (16,7)	3,3 (4,6)	23,8 (41,8)	18,8 (29,3)	29,4 (32,6)	69,1 (34,6)	**0,016**
	Koko tehtävä	29,2 (33,3)	20,4 (36,7)	31,4 (35,1)	0,410	7,1 (14,3)	7,9 (8,1)	26,5 (43,3)	20,8 (25,3)	33,2 (33,9)	68,9 (35,2)	**0,014**
Tehtävä 2	Algoritminen ajattelu	47,1 (34,2)	46,7 (36,9)	40,0 (40,7)	0,758	0,0 (0,0)	6,7 (5,4)	48,3 (43,0)	37,5 (28,9)	58,8 (31,8)	78,1 (28,5)	**0,004**
	Ohjelmointi	15,1 (28,3)	14,6 (28,8)	15,3 (28,8)	0,866	0,0 (0,0)	0,0 (0,0)	16,7 (23,6)	7,3 (17,2)	18,8 (29,1)	40,5 (46,0)	0,190
	Kielentäminen	38,0 (29,9)	38,5 (33,2)	35,9 (34,0)	0,931	0,0 (0,0)	11,5 (7,7)	44,2 (3,4)	31,3 (23,6)	44,7 (28,5)	67,0 (32,9)	**0,018**
	Koko tehtävä	38,0 (29,5)	37,9 (32,6)	34,1 (34,8)	0,812	0,0 (0,0)	7,4 (5,1)	41,2 (37,3)	29,8 (23,3)	46,3 (28,1)	67,2 (30,2)	**0,004**
Tehtävä 3	Algoritminen ajattelu	40,0 (36,8)	25,7 (30,5)	48,0 (41,0)	0,461	0,0 (0,0)	10,0 (10,0)	27,5 (12,6)	22,3 (20,5)	64,2 (37,3)	84,0 (25,1)	**0,002**
	Ohjelmointi	21,4 (19,3)	18,4 (15,9)	34,3 (24,5)	0,346	0,0 (0,0)	9,5 (8,2)	28,6 (16,5)	14,3 (15,4)	35,7 (19,7)	45,7 (12,0)	**0,004**
	Kielentäminen	33,7 (34,4)	25,0 (23,1)	42,5 (37,6)	0,563	4,2 (5,9)	8,3 (8,3)	25,0 (11,8)	17,3 (20,0)	56,3 (32,8)	75,0 (26,4)	**0,002**
	Koko tehtävä	31,8 (29,7)	22,9 (21,6)	41,0 (33,8)	0,460	1,7 (2,4)	8,9 (8,4)	25,8 (12,0)	17,7 (17,3)	52,2 (28,5)	68,7 (20,1)	**0,001**
Tehtävä 4	Algoritminen ajattelu	52,2 (38,3)	47,2 (51,0)	36,1 (35,2)	0,489	3,7 (6,4)	15,6 (21,7)	38,9 (23,6)	32,5 (33,2)	62,0 (35,9)	96,8 (8,4)	**<0,001**
	Ohjelmointi	41,9 (32,8)	39,1 (37,5)	37,5 (36,5)	0,827	4,2 (7,2)	15,0 (20,5)	43,8 (26,5)	24,0 (19,4)	55,2 (33,5)	82,1 (17,5)	**0,001**
	Kielentäminen	46,5 (35,7)	45,2 (44,2)	37,8 (33,7)	0,734	7,7 (13,3)	15,4 (18,0)	38,5 (43,5)	31,4 (27,6)	55,8 (33,1)	90,1 (15,2)	**0,001**
	Koko tehtävä	47,0 (35,2)	44,2 (43,9)	37,2 (34,4)	0,531	5,6 (9,6)	15,3 (19,7)	40,0 (33,0)	29,7 (26,4)	57,5 (33,4)	90,0 (11,5)	**<0,001**
Kaikki tehtävät	Algoritminen ajattelu	41,3 (31,0)	35,5 (37,4)	34,9 (34,7)	0,632	3,8 (7,6)	13,2 (9,6)	28,1 (35,9)	30,1 (21,5)	49,4 (30,4)	84,1 (16,7)	**<0,001**
	Ohjelmointi	24,1 (25,1)	23,7 (29,5)	23,4 (28,2)	0,891	0,8 (1,6)	8,3 (10,7)	23,1 (29,4)	14,3 (15,5)	29,4 (25,5)	59,4 (24,0)	**0,002**
	Kielentäminen	33,5 (29,6)	32,1 (36,0)	30,0 (31,3)	0,798	4,1 (8,1)	10,0 (7,2)	25,3 (34,3)	23,8 (21,4)	39,8 (27,4)	76,7 (21,8)	**<0,001**

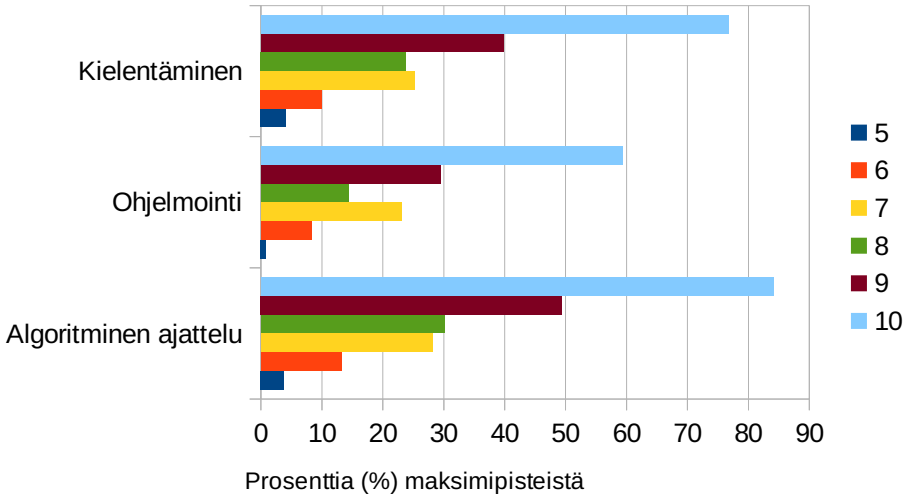
Liite 8(1). Aihealueiden tehtäväkohtaiset pisteet sekä eri koulutusasteiden ja matematiikan arvosanojen vertailut.



Aihealueista suoriutuminen koulutusasteiden ja luokka-asteiden mukaan



Aihealueista suoriutuminen matematiikan arvosanan mukaan



Liite 9(1). Tutkittavien laatimia ratkaisuja tehtäviin.

T50 tehtävä 1

RATKAISUSI TEHTÄVÄÄN 1:

Piirsin terävän kulman

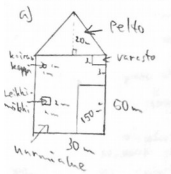
Koodin suoritusjärjestys ja kielentäminen:

1	G	Sivun 1 pituudeksi saadaan funktion kaveleJaMittaaSivu() avulla arvo 3. Muuttujan kuljettuMatka arvoksi sijoitetaan sivu1 n arvo eli kuljettuMatka = sivu1 = 3.
2	J	Käännytään 90° astetta vasemmalle
3	I	sivun 2 pituudeksi tulee funktion kaveleJaMittaaSivu() avulla 6. Muuttujan kuljettuMatka arvoksi sijoitetaan sivu1:n ja sivu2:n arvot eli saadaan 3+6
4	C	Kone tulostaa print-osan "Piirsin suorakulman"
5	D	Käännytään 225° astetta oikealle.
6	K	sivun pituudeksi saadaan funktion avulla noin 4.2. kuljettuMatka - muuttujan arvoksi tulee sivujen 1-3 yhteenlasku eli 3+6+4,2.
7	F	Kone tulostaa print-osan "Piirsin terävän kulman"
8	A	if-else-rakenne kuvaa ehtoa. Jos kuljettuMatka on yli 13, käännytään 45° vasemmalle. Muuten 135° vasemmalle. kuljettuMatka on n. 13,2 eli 45°
9	B	sivun pituudeksi saadaan funktion avulla 2. Muuttujan kuljettuMatka arvoksi sijoitetaan sivu1-4 eli saadaan 3+6+4,2+2
10	E	Kone tulostaa print-osan "Piirsin tylpän kulman"
11	H	puolisuunnikkaan piiriksi tulee kuljettu matka. Kone tulostaa print-osan "Puolisuunnikkaan piiri on". Muuttujan puolisuunnikkaanPiiri kohdalle tulee kuljettu matka. Muuttujan pintaAla - arvoksi tulee lasku sivu1 * (sivu2+ sivu4)/2, jossa käytetään aiemmin mitattujen sivujen pituuksia. Kone tulostaa print-osan "Puolisuunnikkaan pinta-ala on". Muuttujan pintaAla kohdalle tulee sen arvoksi asetettu lasku.

```
E  print("Piirsin tylpän kulman")
F  print("Piirsin terävän kulman")
G  sivu1 = kaveleJaMittaaSivu()
   # sivu1 saa mitaksi 3
   kuljettuMatka = sivu1
H  puolisuunnikkaanPiiri = kuljettuMatka
   print("Puolisuunnikkaan piiri on ",
         puolisuunnikkaanPiiri)
   pintaAla = sivu1 * (sivu2 + sivu4) / 2
   print("Puolisuunnikkaan pinta-ala on "
         pintaAla)
I  sivu2 = kaveleJaMittaaSivu()
   # sivu2 saa mitaksi 6
   kuljettuMatka = sivu1 + sivu2
J  kaannyVasemmalle(90)
   # 90 astetta
K  sivu3 = kaveleJaMittaaSivu()
   # sivu3 saa mitaksi noin 4.2
   kuljettuMatka = sivu1 + sivu2 + sivu3
```

T68 tehtävä 2

RATKAISUSI TEHTÄVÄÄN 2:



b) Rakennusten pinta-ala on 14

Pellon pinta-ala = $\frac{30 \cdot 20}{2}$

Kysytty pinta-ala = tontin pinta-ala − rakennusten pinta-ala − aitauksen pinta-ala

c) lasketaan ensin pihan pinta-ala $30_m \cdot 50_m = 1500\,m^2$ ja lisätään siihen pellon pinta-ala $\frac{10_m \cdot 20_m}{2} = 300\,m^2$

heliön muotoisten piharakennusten pinta-ala on
$3 \cdot 5 + 1 \cdot 1 + 2 \cdot 2 = 14\,(m^2)$

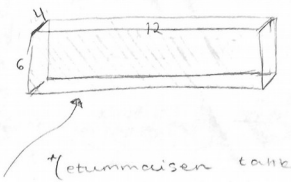
nyt tiedetään kaikki tarvittava, kuten tehtävän pinta-ala $150\,m^2$ on annettu.

kysytty pinta-ala on (pelto + piha − rakennukset)

$1800\,m^2 - 164\,m^2 = 1636\,m^2$

T46 tehtävä 3

RATKAISUSI a)- ja c)-KOHDAT TEHTÄVÄÄN 3:



↑ (etummaisen tahkon)

a) Laske suorakulmion piiri, kun lyhyemmän sivun pituus on 6 ja pidemmän sivun pituus on 12.

Laske pienemmän tahkon pinta-ala, kun kanta on 4 ja korkeus on 6.

c) Etummaisen tahkon piiri lasketaan sivujen summana.

Pinta-ala lasketaan kertomalla kanta ja korkeus

T59 tehtävä 4

RATKAISUSI TEHTÄVÄÄN 4:

Alussa todetaan, että yksi neliön sivu on 4 ja neliön sisällä olevan ympyrän halkaisija on 4

Tässä vaiheessa lasketaan neliön ja ympyrän pinta-alat. Neliö = $4 \cdot 4 = 4^2 = 16$. Ympyrän pinta-alan laskemassa käytetään termiä "math.pi" jonka arvo on pii (3,1415...). Ympyrän pinta-ala lasketaan funktiolla $\pi \cdot (\frac{4}{2}) \cdot (\frac{4}{2}) = \pi \cdot r \cdot r = \pi \cdot r^2 = \pi \cdot 2^2$ ($\approx 12{,}6$)
(d ympyrän halkaisija) (r = ympyrän säde)
=4 =2

if-osa:

Jos neliön pinta-ala on suurempi kuin ympyrän pinta-ala, erotusalan laskeminen on mahdollista.
Termillä math.sqrt lasketaan erotusalan neliöjuuri, joka on erotusSivu. erotusSivu = $\sqrt{(16-12{,}6)} = \sqrt{3{,}4} = 1{,}84$.
print-osassa lukee, että erotetun pinta-alan suuruus on 3,4. Jos ympyrän tilalla olisi ollut neliö, tämän funktion mukaan sen sivun pituus tulisi olla 1,84.

else-osa:

Else-osaa ei ole, koska funktio tämi if-lohkolassa