Tampereen yliopisto

Katri Leinonen

PIKALAJITTELU JA LOMITUSLAJIT-TELU

Kandidaatintyö
Informaatioteknologian ja viestinnän tiedekunta
Huhtikuu 2021

TIIVISTELMÄ

Katri Leinonen: Pikalajittelu ja lomituslajittelu
Kandidaatintyö
Tampereen yliopisto
Tieto- ja sähkötekniikan kandidaatin tutkinto-ohjelma
Huhtikuu 2021

Järjestysalgoritmit ovat hyvin yleisiä algoritmeja ohjelmissa. Sen vuoksi on tärkeää, että ohjelmaan on valittu oikea algoritmi ja että se on toteutettu mahdollisimman tehokkaasti. Jos käyttötapaukseen on valittu epäsopiva algoritmi, ohjelman tehokkuus voi heikentyä huomattavasti.

Hajota ja hallitse -suunnitteluperiaatteen mukainen algoritmi jakaa alkuperäisen ongelman pienempiin osaongelmiin, jotka voidaan ratkaista itsenäisesti. Kun osaongelmat on ratkaistu, voidaan ne yhdistää takaisin yhteen. Näin saadaan alkuperäisen ongelman ratkaisu. Pikalajittelu ja lomituslajittelu ovat hajota ja hallitse -suunnitteluperiaatteen mukaisia järjestysalgoritmeja. Näiden algoritmien toteutuksissa on kuitenkin eroavaisuuksia, minkä vuoksi niiden soveltuvuus eri käyttötapauksiin vaihtelee.

Tämän työn tarkoituksena on selvittää pikalajittelun ja lomituslajittelun soveltuvuus eri käyttötapauksiin. Tässä käytetään apuna niin aiempia tutkimuksia kuin itse suoritettuja suoritusaikatestejä. Tutkimusta varten suoritusaikatestit toteutettiin lomituslajittelulle, single-pivot-pikalajittelun kolmelle erilaiselle toteutukselle ja yhdelle dual-pivot-pikalajittelun toteutukselle. Suoritusaikatesteissä käytettiin sekä satunnaisessa järjestyksessä olevaa dataa että järjestyksessä olevaa dataa.

Suoritusaikatesteistä voidaan havaita, että mikäli ohjelma käsittelee usein järjestyksessä olevaa dataa tai melkein järjestyksessä olevaa dataa, perinteinen pikalajittelu ei ole hyvä vaihtoehto taulukon järjestämiseksi. Jos tiedetään, että ohjelma käsittelee satunnaisesti melkein järjestyksessä olevaa dataa, on kokonaistehokkuuden kannalta järkevintä käyttää lomituslajittelua, koska sillä ei ole yhtä tehotonta huonointa tapausta kuin pikalajittelulla.

Aiempien tutkimuksien perusteella sopivan järjestysalgoritmin valitsemisessa kannattaa huomioida myös muut asiat kuin pelkkä tehokkuus. Jos ohjelma käsittelee hyvin suuria tietojoukkoja, jotka eivät mahdu kerrallaan prosessoivan tietokoneen muistiin, kannattaa käyttää lomituslajittelun ulkoista järjestelyä. Tällä tavoin järjestettävää tietojoukkoa ei tarvitse lukea kokonaisuudessaan prosessoivan tietokoneen muistiin. Lomituslajittelu on valittava myös, jos ohjelma tarvitsee stabiilin järjestysalgoritmin, koska pikalajittelu ei ole stabiili järjestysalgoritmi.

Avainsanat: Hajota ja hallitse -suunnitteluperiaate, pikalajittelu, lomituslajittelu, asymptoottinen suoritusaika

SISÄLLYSLUETTELO

.

1. JOHDANTO

Järjestysalgoritmit ovat hyvin yleisiä algoritmeja ohjelmissa. Sen vuoksi on tärkeää, että ohjelmaan on valittu oikea algoritmi ja että se on toteutettu mahdollisimman tehokkaasti. Jos käyttötapaukseen on valittu epäsopiva algoritmi, ohjelman tehokkuus voi vähentyä huomattavasti.

Algoritmin tehokkuutta tarkastellaan asymptoottisen suoritusajan perusteella. Asymptoottinen suoritusaika kuvaa algoritmin tehokkuuden rajoja suhteessa tietojoukon kokoon, jota algoritmi käsittelee. Tässä työssä tehokkuutta tarkastellaan O-notaation avulla. O-notaatiolla voidaan määritellä funktio, joka kuvaa algoritmin suoritusajan kasvua suhteessa tietojoukon suuruuteen. O-notaation on verrannollinen perinteiseen aritmeettisen vertailuoperaatioon ”pienempi tai yhtä suuri kuin”, eli se rajoittaa algoritmin kasvua ylhäältäpäin [1].

Hajota ja hallitse -suunnitteluperiaatteen mukainen algoritmi jakaa alkuperäisen ongelman pienempiin osaongelmiin, jotka voidaan ratkaista itsenäisesti. Kun osaongelmat on ratkaistu, voidaan ne yhdistää takaisin yhteen. Näin saadaan alkuperäisen ongelman ratkaisu. Pikalajittelu ja lomituslajittelu ovat hajota ja hallitse -suunnitteluperiaatteen mukaisia järjestysalgoritmeja. Näiden algoritmien toteutuksissa on kuitenkin eroavaisuuksia, minkä vuoksi niiden soveltuvuus eri käyttötapauksiin vaihtelee.

Tämän työn tutkimuskysymys on selvittää, miten pikalajittelu ja lomituslajittelu soveltuvat eri käyttötapauksiin. Tässä käytetään apuna niin aiempia tutkimuksia kuin itse suoritettuja suoritusaikatestejä. Luvussa 2 tarkastellaan yksityiskohtaisemmin pikalajittelun ja lomituslajittelun toteutuksia ja eroavaisuuksia. Kolmannessa luvussa esitellään suoritettavat suoritusaikatestit, joiden tuloksia tarkastellaan luvussa 4. Luku 5 sisältää yhteenvedon.

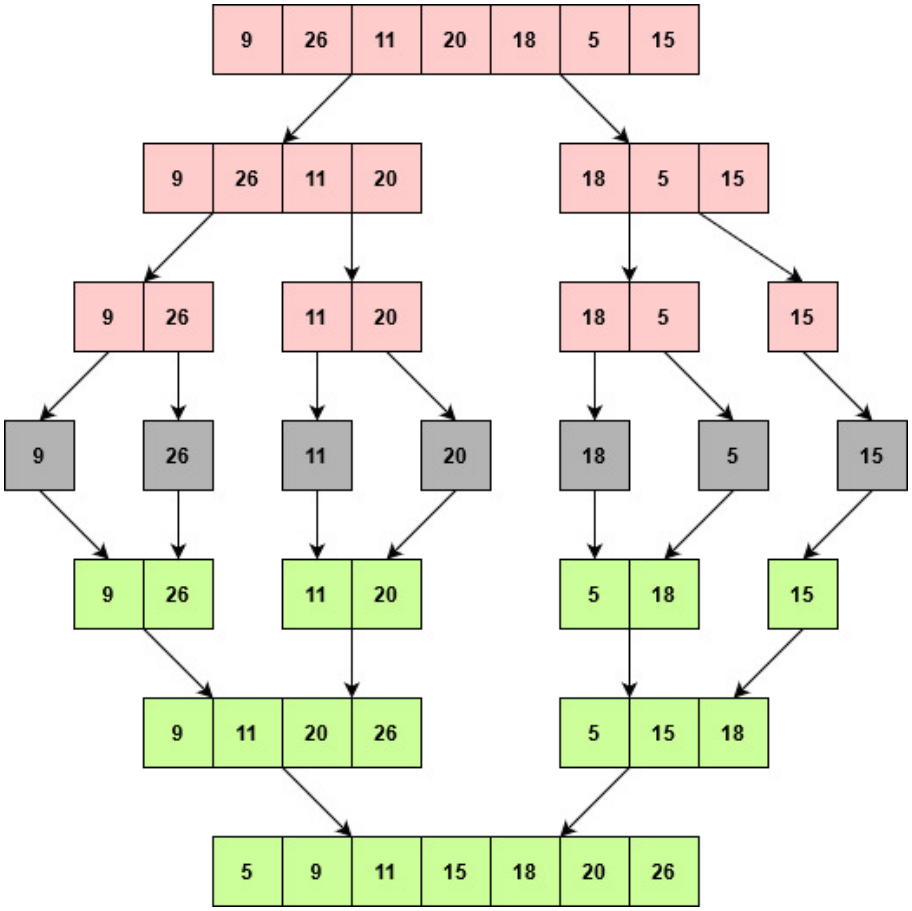
2. HAJOTA JA HALLITSE -ALGORITMIT

Hajota ja hallitse -suunnitteluperiaatteessa alkuperäinen ongelma jaetaan itsenäisiksi osaongelmiksi, jotka voidaan rekursiivisesti ratkaista. Ratkaistut osaongelmat yhdistetään ja näin saadaan alkuperäisen ongelman ratkaisu.

Useat järjestysalgoritmit käyttävät hajota ja hallitse -suunnitteluperiaatetta [2]. Näistä kaksi hyvin tunnettua ovat pikalajittelu (engl. quicksort) ja lomituslajittelu (engl. merge sort). Seuraavissa alaluvuissa tarkastellaan näitä algoritmeja yksityiskohtaisemmin.

2.1 Lomituslajittelu

Lomituslajittelu on järjestysalgoritmi, joka jakaa ongelman aina kahteen yhtä suureen osaan eli taulukkoon. Algoritmi kutsuu rekursiivisesti itseään uudelleen, kunnes jokainen taulukko sisältää vain yhden alkion. Tämän jälkeen algoritmi yhdistää aina kaksi saman kokoista taulukkoa keskenään, kunnes alkuperäinen ongelma on kokonaan järjestyksessä. Tämä lomituslajittelun perustoimintaperiaate on esitetty kuvassa 1.



Kuva 1. Lomituslajittelun perustoimintaperiaate

Lomituslajittelun suoritusaika ei riipu järjestettävien alkioiden alkuperäisestä järjestyksestä, minkä vuoksi sillä ei ole erityisen huonoa suoritusaikaa huonoimmassakaan tapauksessa. Tämän vuoksi lomituslajittelun tehokkuus kuvattuna O-notaation avulla on O(NlogN) [3].

Muita lomituslajittelun hyviä puolia on se, että se on rinnakkaistettavissa oleva algoritmi. Tämä kuitenkin vaatii koordinointia, koska alkuperäisen kutsun pitää odottaa molempien puolien rekursiivisten kutsujen päättymistä ennen kuin se voi yhdistää järjestetyt puoliskot. [3]

Toinen lomituslajittelun etu on se, että sitä voi käyttää myös silloin, kun järjestettävä data ei mahdu kerralla kokonaan prosessoivan tietokoneen muistiin. Tällaista järjestämistä kutsutaan ulkoiseksi järjestämiseksi (engl. external sorting). Lomituslajittelun ei tarvitse tarkastella lajiteltavaa tietojoukkoa kokonaisuudessaan, minkä vuoksi se ei tarvitse niin paljon muistia. Näin lomituslajittelulla voidaan järjestää hyvinkin suuria tietojoukkoja.

Lomituslajittelu on myös stabiili järjestysalgoritmi. Tämä tarkoittaa sitä, että samanarvoiset alkiot pitävät alkuperäiset suhteelliset sijaintinsa toisiinsa nähden. Näin lomituslajittelu tuottaa aina saman lopputuloksen samalle datalle. [3]

2.2 Pikalajittelu

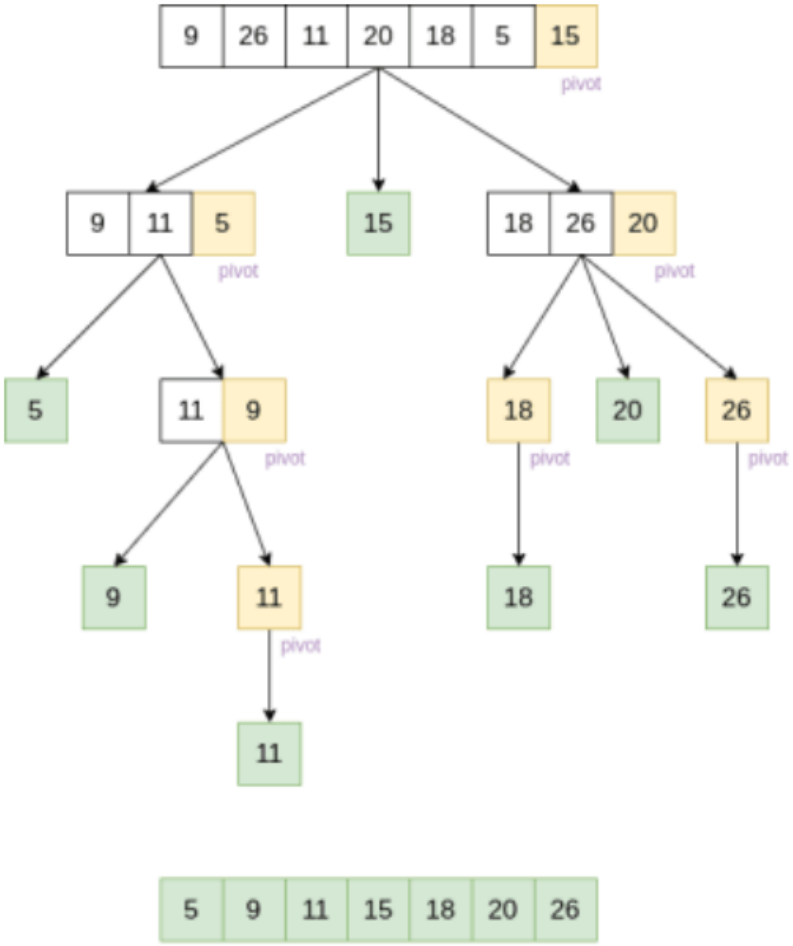
Pikalajittelu on järjestysalgoritmi, joka jakaa ongelman osataulukoihin pivot-arvojen avulla. Pivot-arvo on vapaasti suomennettuna sarana-arvo, jota käytetään pikalajittelussa vertailuarvona muille arvoille. Taulukon muut alkiot jaetaan osataulukoihin vertailemalla niitä pivot-arvoon. Pivot-arvoa pienemmät alkiot lajitellaan ensimmäiseen osataulukkoon ja pivot-arvoa suuremmat alkiot toiseen osataulukkoon. Algoritmi kutsuu rekursiivisesti itseään uudelleen, kunnes taulukko on kokonaan järjestyksessä.

Pikalajittelun voi toteuttaa monella eri tavalla riippuen siitä, kuinka pivot-arvo valitaan. Seuraavissa alaluvuissa tarkastellaan pikalajittelun toteutuksia, joissa käytetään joko yhtä tai useampaa pivot-arvoa.

2.2.1 Single-pivot

Single-pivot-pikalajittelussa valitaan aina yksi pivot-arvo, jonka perusteella jako suoritetaan. Pivot-arvon valinta voidaan toteuttaa pääsääntöisesti neljällä eri tavalla: valitaan aina taulukon ensimmäinen alkio, valitaan aina taulukon viimeinen alkio, valitaan satun-

nainen alkio pivot-arvoksi tai valitaan taulukon keskimmäinen alkio pivot-arvoksi. Kuvassa 2 on esitelty pikalajittelun perustoimintaperiaate, kun pivot-arvoksi valitaan aina taulukon viimeinen alkio.



Kuva 2. Pikalajittelun perustoimintaperiaate, jossa pivot-arvoksi valitaan aina taulukon viimeinen alkio [4]

Parhaimmassa tapauksessa pikalajittelu jakaisi taulukon aina kahteen yhtä suureen osaan pivot-arvon avulla, mutta tämä harvoin toteutuu. Parhaimmassa tapauksessa pikalajittelun asymptoottinen tehokkuus on O(NlogN).

Huonoimmassa tapauksessa pivot-arvo on pienempi, suurempi tai yhtä suuri kuin kaikki muut taulukon alkiot. Tässä tapauksessa kaikki taulukon alkiot lajitellaan taulukon samaan puoliskoon ja tämän puoliskon rekursiivinen funktiokutsu pitää prosessoida kaikki alkiot. Jos taulukossa on N alkiota, tässä tapauksessa rekursiivinen funktiokutsu pitää järjestää N – 1 alkiota. Tämän vuoksi pikalajittelun asymptoottinen tehokkuus huonoimmassa tapauksessa on $O(N^2)$. [3]

Koska huonoimmassa tapauksessa pikalajittelun tehokkuus on huomattavasti heikompi kuin esimerkiksi lomituslajittelun, pivot-arvon valinta kannattaa toteuttaa mahdollisimman järkevästi ja huolella. Esimerkiksi jos ohjelma käsittelee usein järjestyksessä tai melkein järjestyksessä olevia taulukoita, ensimmäisen tai viimeisen arvon valitseminen pivot-arvoksi voi aiheuttaa sen, että algoritmin tehokkuus on lähellä huonointa mahdollisinta.

Jos halutaan kasvattaa hyvän pivot-arvon valitsemisen todennäköisyyttä, yksi vaihtoehto on sekoittaa järjestettävän taulukon alkiot ennen pivot-arvon valitsemista. Silloin on hyvin epätodennäköistä, että taulukon ensimmäinen tai viimeinen alkio on koko taulukon suurin tai pienin alkio. Sekoittamisen asymptoottinen tehokkuus taulukoille on O(N) [3], joten tällä ei ole vaikutusta pikalajittelun O-notaation asymptoottiseen suoritusaikaan. Käytännössä tämä vie kuitenkin aikaa varsinkin isoille taulukoille, minkä vuoksi useat ohjelmoijat eivät käytä tätä menetelmää [3].

Toinen vaihtoehto, jolla voi kasvattaa hyvän pivot-arvon valitsemisen todennäköisyyttä, on tarkastella järjestettävän taulukon ensimmäistä, keskimmäistä ja viimeistä arvoa ja valita näistä pivot-arvoksi se, joka on arvoltaan toiseksi suurin. Tällä tavalla ei voida varmistaa täysin, että pivot-arvo ei olisi melkein pienin tai suurin alkio koko taulukosta. Huonon pivot-arvon valitseminen on kuitenkin huomattavasti epätodennäköisempää toimittaessa tällä tavalla.

Pivot-arvon valitseminen huolellisesti on tärkeää myös prosessoivan tietokoneen muistin kannalta. Jos pivot-arvo valitaan erittäin huonosti isolle tietojoukolle, rekursiivisten kutsujen sarjasta tulee liian syvä. Tällöin ohjelma käyttää loppuun aktivaatiotietuepinolle varatun tilan, minkä seurauksena ohjelma kaatuu. [3]

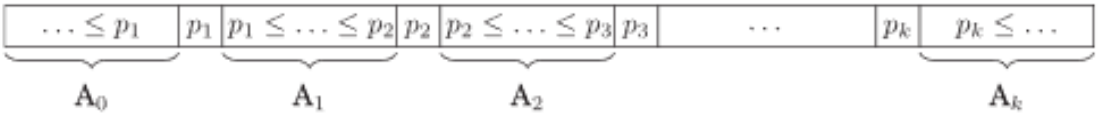
Koska pikalajittelun asymptoottinen tehokkuus huonoimmassa tapauksessa on O(N) ja on mahdollista, että prosessoivan tietokoneen muisti loppuu kesken, useiden kirjastojen tarjoamat pikalajittelun toteutukset voivat olla kryptografisesti epävarmoja. Jos pikalajittelu käyttää yksinkertaista pivot-arvon valintamenetelmää, esimerkiksi valitsemalla aina taulukon viimeinen arvo pivot-arvoksi, hyökkääjä voi käyttää tätä tietoa hyödyksi. Tällöin hyökkääjä voi syöttää ohjelmalle suuren tietojoukon, joka toteuttaa huonoimman mahdollisen tapauksen pikalajittelussa ja mahdollisesti käynnistää palvelunestohyökkäyksen. Tätä ei oteta usein huomioon ohjelmia suunniteltaessa, mutta tämä on mahdollista tehdä. [3]

Pikalajittelun yksi eduista on se, että se on rinnakkaistettavissa. Pikalajittelun jakaessa taulukon kahteen osaan pivot-arvon avulla on näistä mahdollista aloittaa itsenäiset prosessit eri prosessoreille. Teoriassa prosessoiva tietokone voisi käyttää N-prosessoria

järjestääkseen taulukot asymptoottisessa ajassa O(logN). Käytännössä tämä ei kuitenkaan ole mahdollista, koska tietokoneissa on rajattu määrä prosessoreita. Tämän vuoksi rinnakkaistettavuus ei muuta asymptoottisen tehokkuuden O-notaatiota, mutta ero on havaittavissa käytännön suoritusaikatesteissä. [3]

2.2.2 Multi-pivot

Toisin kuin single-pivot-pikalajittelussa, multi-pivot pikalajittelussa valitaan useampi kuin yksi pivot-arvo, jonka perusteella lajittelu suoritetaan. Kuvassa 3 on esitetty, kuinka taulukko jaetaan käyttäen k-pivottia. Segmentissä *i*, joka on merkitty A_i, koostuu alkioista, jotka ovat suurempia tai yhtä suuria kuin pivot p_i ja pienempiä kuin p_{i+1} [5]. Ennen kuin alkiot voidaan lajitella pivot-arvojen perusteella, on pivot-arvot järjesteltävä keskenään oikeaan järjestykseen.



Kuva 3. Esimerkki taulukon jakamisesta käyttäen k-pivottia [5]

Multi-pivot pikalajitteluista eniten huomiota on saanut Yariskavskiyn dual-pivot-pikalajittelu vuonna 2009, kun siitä tuli uusi standardi pikalajittelu Oraclen Java 7:n ajonaikaisessa kirjastossa [6]. Tässä algoritmissa hyödynnetään kahta pivot-arvoa, jonka avulla taulukko jaetaan kolmeen osaan, joissa algoritmia kutsutaan rekursiivisesti uudelleen.

Kuten single-pivot-pikalajittelussa niin multi-pivot pikalajittelussa pivot-arvojen valitseminen taulukosta voidaan toteuttaa usealla eri tavalla. Esimerkiksi dual-pivot-pikalajittelussa ensimmäiseksi pivot-arvoksi voidaan valita taulukon ensimmäinen alkio ja toiseksi pivot-arvoksi taulukon viimeinen alkio.

Toinen mahdollinen tapa, jota käytetään useasti, on valita pivot-arvot pienestä joukosta taulukon alkiosta. Tässä tapauksessa pivot-arvot valitaan *2k + 1* kokoisesta näytteenkoosta, jossa *k* kuvaa pivot-arvojen määrää. [5] Esimerkiksi dual-pivot-pikalajittelussa valitaan viisi satunnaista alkiota, joista toiseksi ja neljänneksi suurin alkio valitaan pivot-arvoiksi. Tällä tavoin voidaan varmistaa, että valitut pivot-arvot eivät ole taulukon suurimpia tai pienimpiä alkioita, mikä vähentäisi algoritmin tehokkuutta.

Dual-pivot-pikalajittelun keskiarvoinen asymptoottinen suoritusaika on hieman parempi kuin single-pivot-pikalajittelun [6], mutta sen huonoimman tapauksen suoritusaika on sama kuin single-pivot-pikalajittelussa eli $O(N^2)$ [7].

3. SUORITUSAIKATESTIT

Tässä työssä on tarkoituksena tarkastella pikalajittelun ja lomituslajittelun soveltuvuutta eri käyttötapauksiin suoritusaikatestien avulla. Ohjelmissa 1 – 5 on esitelty suoritusaikatesteissä käytettyjen algoritmien pseudokoodit. Itse suoritusaikatesteissä algoritmit toteutettiin C++ kielellä. Ympäristönä käytettiin Qt Creator -ympäristöä Tampereen yliopiston tarjoamalla Linux-etätyöpöydällä. Ajanmittaamiseen käytettiin std::chrono-kirjastoa.

Ohjelma 1 on pseudokoodi lomituslajittelusta. Ohjelmassa 1 on mainittu *yhdistä-funktio*, jota käytetään tässä pseudokoodissa kuvaamaan yleisellä tasolla funktiota, joka yhdistää kaksi järjestyksessä olevaa taulukkoa yhdeksi isoksi järjestyksessä olevaksi taulukoksi. *Yhdistä-funktion* lopputuloksena on yksi järjestyksessä oleva taulukko.

```
lomituslajittelu( taulukko, vasen, oikea )
    jos oikea > vasen
        keskikohta = ( vasen + oikea ) / 2
        lomituslajittelu( taulukko, vasen, keskikohta)
        lomituslajittelu( taulukko, keskikohta + 1, oikea)
        yhdistä(taulukko, vasen, keskikohta, oikea )
```

Ohjelma 1. Lomituslajittelun pseudokoodi

Ohjelma 2 on pseudokoodi pikalajittelusta, jossa käytetään taulukon viimeistä alkiota pivot-arvona. Tämä pikalajittelu on toteutettu Nico Lomuton pikalajittelun pohjalta ja sitä kutsutaan yleisesti Lomuton jakamisrakenteeksi (engl. Lomuto Partitioning Scheme) [8].

```
jaa( taulukko, pieni, suuri )
    pivot = taulukko[suuri]
    i = pieni   //varattu paikka vaihtamiselle
    kunnes j := pieni on suuri – 1 tee
        jos taulukko[j] <= pivot
            vaihda taulukko[i] ja taulukko[j]
            i = i + 1
    vaihda taulukko[i] ja taulukko[suuri]
    palauta i

pikalajittelu( taulukko, pieni, suuri )
    jos pieni < suuri
        pivot = jaa( taulukko, pieni, suuri)
        pikalajittelu( taulukko, pieni, pivot – 1 )
        pikalajittelu( taulukko, pivot + 1, suuri )
```

Ohjelma 2. Pseudokoodi pikalajittelusta, jossa pivot-arvo valitaan taulukon lopusta

Ohjelma 3 on pseudokoodi pikalajittelusta, jossa käytetään satunnaisesti valittua pivot-arvoa. Tämäkin algoritmin toteutus on tehty Lomuton pikalajittelun [8] pohjalta, mutta siihen on lisätty yksi ylimääräinen funktio *jaa_satunnainen*. Tämän funktion tarkoitus on arpoa taulukosta satunnainen alkio pivot arvoksi, joka siirretään taulukon viimeiseksi arvoksi *jaa-funktiota* varten, joka on samanlainen kuin ohjelmassa 2.

```
jaa( taulukko, pieni, suuri )
    pivot = taulukko[suuri]
    i = pieni   //varattu paikka vaihtamiselle
    kunnes j := pieni on suuri – 1 tee
        jos taulukko[j] <= pivot
            vaihda taulukko[i] ja taulukko[j]
            i = i + 1
    vaihda taulukko[i] ja taulukko[suuri]
    palauta i

jaa_satunnainen( taulukko, pieni, suuri )
    r = satunnainen numero väliltä pieni suuri
    vaihda taulukko[r] ja taulukko[suuri]
    palauta jaa( taulukko, pieni, suuri )

pikalajittelu( taulukko, pieni, suuri )
    jos pieni < suuri
        pivot = jaa_satunnainen( taulukko, pieni, suuri)
        pikalajittelu( taulukko, pieni, pivot – 1 )
        pikalajittelu( taulukko, pivot + 1, suuri )
```

Ohjelma 3. Pseudokoodi pikalajittelusta, jossa käytetään satunnaista alkiota pivot-arvona

Ohjelma 4 on pseudokoodi pikalajittelusta, jossa pivot-arvoksi valitaan mediaani taulukon kolmesta alkiosta. Tämä tapahtuu siten, että vertaillaan taulukon ensimmäistä, keskimmäistä ja viimeistä alkiota ja valitaan näistä keskisuurin pivot-arvoksi. Tämä on toteutettu hyvin samalla tavalla kuin ohjelma 3. Funktio *jaa_satunnainen* on vain korvattu toisella funktiolla *etsi_mediaani*, joka tarkastelee taulukon ensimmäistä, keskimmäistä ja viimeistä alkiota ja siirtää näistä keskisuurimman taulukon viimeiseksi.

```
jaa ( taulukko, pieni, suuri )
    pivot = taulukko[suuri]
    i = pieni   //varattu paikka vaihtamiselle
    kunnes j := pieni on suuri – 1 tee
        jos taulukko[j] <= pivot
            vaihda taulukko[i] ja taulukko[j]
            i = i + 1
    vaihda taulukko[i] ja taulukko[suuri]
    palauta i

etsi_mediaani ( taulukko, pieni, suuri )
    keskikohta = pieni + ( suuri – pieni ) / 2
    jos taulukko[pieni] > taulukko[keskikohta]
        vaihda taulukko[pieni] ja taulukko[keskikohta]
    jos taulukko[pieni] > taulukko[suuri]
        vaihda taulukko[pieni] ja taulukko[suuri]
    jos taulukko[keskikohta] > taulukko[suuri]
        vaihda taulukko[keskikohta] ja taulukko[suuri]
    vaihda taulukko[keskikohta] ja taulukko[suuri]

    palauta jaa( taulukko, pieni, suuri )

pikalajittelu( taulukko, pieni, suuri )
    jos pieni < suuri
        pivot = etsi_mediaani( taulukko, pieni, suuri)
        pikalajittelu( taulukko, pieni, pivot – 1 )
        pikalajittelu( taulukko, pivot + 1, suuri )
```

Ohjelma 4. Pseudokoodi mediaani kolmesta alkiosta pikalajittelu

Ohjelma 5 on pseudokoodi dual-pivot-pikalajittelusta, jossa taulukon ensimmäinen ja viimeinen alkio valitaan pivot-arvoiksi. Tämä algoritmi ei ole Yariskavskiyn dual-pivot-pikalajittelu, joka mainittiin luvussa 2.2.2.

```
jaa_dual( taulukko, pieni, suuri, vasenpivot )
    jos taulukko[pieni] > taulukko[suuri]
        vaihda taulukko[pieni] ja taulukko[suuri]

    //varataan paikkoja vaihtamiselle
    j = pieni + 1
    g = suuri – 1
    k = pieni + 1

    //p on vasenpivot ja q on oikeapivot
    p = taulukko[pieni]
    q = taulukko[suuri]

    kun k on pienempi tai yhtä suuri kuin g tee

        //jos alkiot ovat pienempiä kuin vasenpivot
        jos taulukko[k] <= p
            vaihda taulukko[k] ja taulukko[j]
            j = j + 1

        //jos alkiot ovat suurempia kuin oikeapivot
        muuten jos taulukko[k] >= q
            kun taulukko[g] > q ja k < g tee
                g = g – 1
                vaihda taulukko[k] ja taulukko[g]
                g = g – 1
                jos taulukko[k] < p
                    vaihda taulukko[k] ja taulukko[j]
                    j = j + 1
        k = k + 1

    //palautetaan pivotit oikeille paikoille
    j = j – 1
    g = g + 1

    vaihda taulukko[pieni] ja taulukko[j]
    vaihda taulukko[suuri] ja taulukko[g]
    vasenpivot = j // koska ei voida palauttaa kahta pivottia
    palauta g

pikalajittelu( taulukko, pieni, suuri )
    jos pieni < suuri
        int vasenpivot, oikeapivot
        pivot = jaa_dual( taulukko, pieni, suuri, &vasenpivot)
        pikalajittelu( taulukko, pieni, vasenpivot – 1 )
        pikalajittelu( taulukko, vasenpivot + 1, oikeapivot – 1)
        pikalajittelu( taulukko, oikeapivot + 1, suuri )
```

Ohjelma 5. Pseudokoodi dual-pivot-pikalajittelusta, jossa pivot arvoiksi valitaan taulukon ensimmäinen ja viimeinen alkio

Tarkoituksena on havainnollistaa näiden algoritmien välillä, kuinka paljon alkuperäisen datan järjestyksellä on vaikutusta algoritmin tehokkuuteen ja kuinka suurissa datamäärissä nämä erot ovat merkittäviä. Tämän vuoksi suoritusaikatestit suoritetaan useilla eri kokoisilla datamäärillä ja satunnaisella sekä valmiiksi järjestyksessä olevalla datalla.

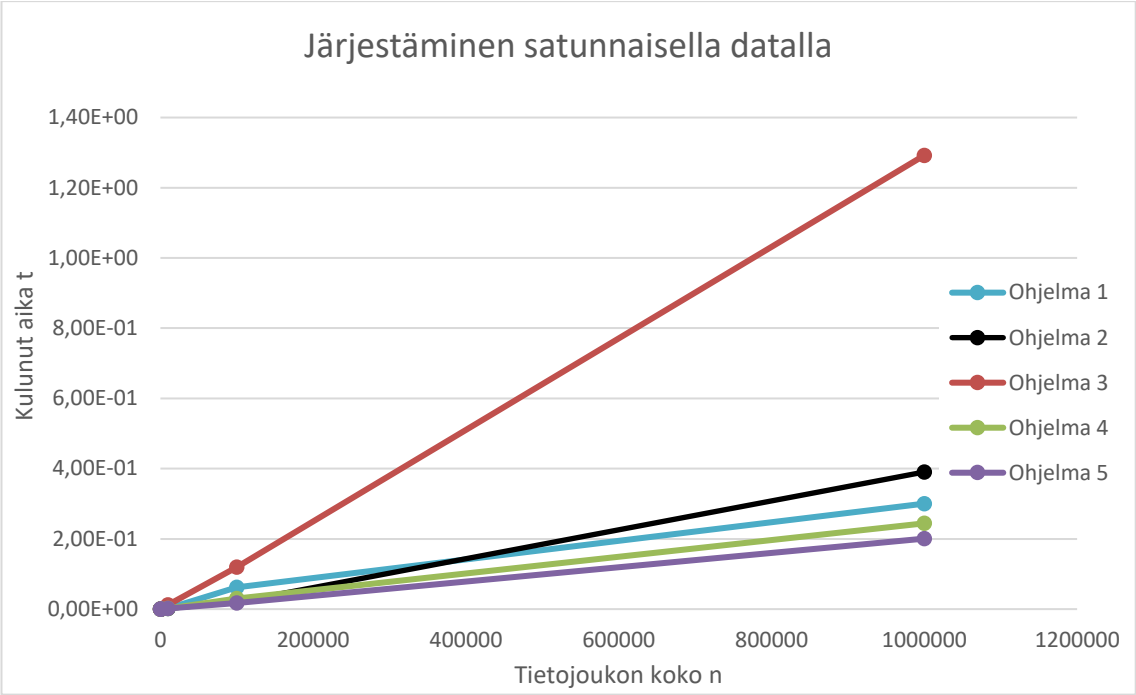
4. TULOKSET JA NIIDEN TARKASTELUT

Suoritusaikatestien tulokset satunnaiselle datalle on esitetty taulukkomuodossa taulukossa 1 ja kaavion muodossa kuvassa 4. Taulukon 1 ja kuvaajan 4 perusteella voidaan havaita, että testatuista algoritmeista ohjelma 5 eli dual-pivot-pikalajittelu on kaikista tehokkain suurille tietojoukoille. Single-pivot-pikalajittelun toteutukset ovat hieman hitaampia kuin dual-pivot-pikalajittelu, ja tämä korostuu entistä enemmän, mitä suurempia tietojoukkoja käsitellään.

Testatuista algoritmeista selkeästi heikoiten suoriutui ohjelma 3, jossa pivot arvo valitaan satunnaisesti taulukosta ja siirretään se taulukon viimeiseksi alkioksi jakamista varten. Tämä johtunee siitä, että satunnaisesti valitseminen on raskas komento suorittaa suhteessa esimerkiksi kolmen alkion vertailuun.

Taulukko 1. Suoritusaikatestien tulokset ohjelmille satunnaisella datalla

	Ohjelma 1	Ohjelma 2	Ohjelma 3	Ohjelma 4	Ohjelma 5
10	$1.754e^{-6}$	$1.043e^{-6}$	$16.859e^{-6}$	$1.043e^{-6}$	$1.197e^{-6}$
100	$14.823e^{-6}$	$9.606e^{-6}$	$111.448e^{-6}$	$8.979e^{-6}$	$7.842e^{-6}$
1000	$0.169804e^{-3}$	$0.166008e^{-3}$	$1.13883e^{-3}$	$0.127124e^{-3}$	$0.101563e^{-3}$
10 000	$2.10533e^{-3}$	$1.81417e^{-3}$	$11.864e^{-3}$	$1.64479e^{-3}$	$1.70396e^{-3}$
100 000	0.0624135	0.0201446	0.119391	0.0305	0.0175477
1 000 000	0.300346	0.390396	1.29222	0.244206	0.200679

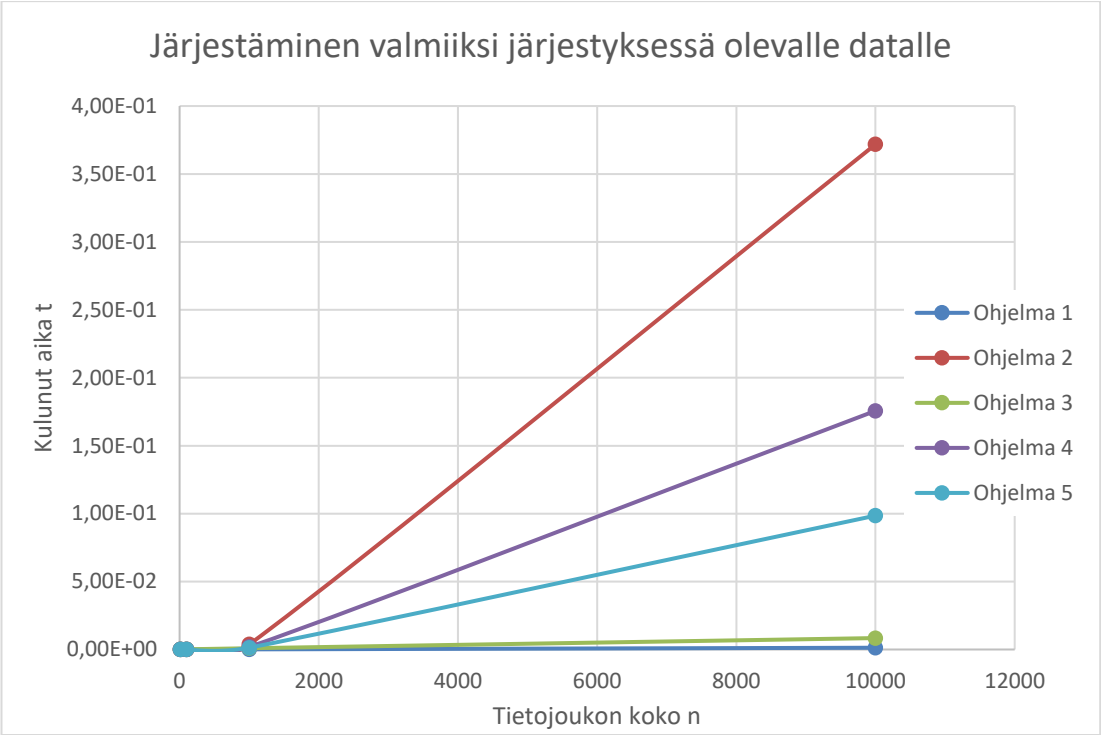


Kuva 4. Järjestäminen satunnaisella datalla

Suoritusaikatestien tulokset valmiiksi järjestyksessä olevalle datalle on esitetty taulukko muodossa taulukossa 2 ja kaavion muodossa kuvassa 5. Huomaa, että kuvassa 5 ei ole esitetty sadantuhannen ja miljoonan alkion kokoisten taulukoiden suoritusaikatestien tuloksia. Tämä johtuu siitä, että nämä tulokset ovat sen verran suuria muihin tuloksiin nähden, minkä vuoksi näiden kuvaaminen visuaalisesti samassa kaaviossa ei ole toimiva tapa.

Taulukko 2. Suoritusaikatestien tulokset ohjelmille järjestyksessä olevalla datalla

	Ohjelma 1	Ohjelma 2	Ohjelma 3	Ohjelma 4	Ohjelma 5
10	$1.454e^{-6}$	$1.107e^{-6}$	$15.124e^{-6}$	$1.279e^{-6}$	$1.285e^{-6}$
100	$10.432e^{-6}$	$38.99e^{-6}$	$141.539e^{-6}$	$21.218e^{-6}$	$11.928e^{-6}$
1000	$0.104609e^{-3}$	$3.7038e^{-3}$	$0.932404e^{-3}$	$1.82389e^{-3}$	$1.32433e^{-3}$
10 000	$1.30386e^{-3}$	$371.781e^{-3}$	$8.3974e^{-3}$	$175.645e^{-3}$	$98.5239e^{-3}$
100 000	0.0155945	35.7896	0.0864597	17.837	9.7875
1 000 000	0.22083	N/A	1.20297	N/A	N/A



Kuva 5. Järjestäminen valmiiksi järjestyksessä olevalle datalle

Näistä tuloksista on havaittavissa, että valmiiksi järjestyksessä olevan datan järjestäminen on kaikista tehokkainta ohjelmalla 1 eli lomituslajittelulla. Tämä johtuu siitä, että lomituslajittelu ei valitse pivot-arvoa, jonka perusteella järjestäminen suoritetaan, vaan järjestäminen suoritetaan aina samalla tavalla riippumatta datasta.

Kuten luvussa 2.2.1 on mainittu, single-pivot-pikalajittelun huonoin tapaus on, kun valitaan taulukon suurin arvo pivot-arvoksi. Taulukon 2 ohjelma 2 kuvaa juuri tätä tilannetta. Ohjelma suoriutuu pienten tietojoukkojen kuten kymmenen alkion kokoisen tietojoukon järjestämisestä, mutta tietojoukon koon kasvaminen hidastaa ohjelmaa huomattavasti. Ohjelma 2 ei suoriutunut miljoonan alkion järjestämisestä ollenkaan. Tämä on merkittävä kryptografinen epävarmuus ohjelmassa. Jos hyökkääjä tietää, että ohjelma käyttää tällaista algoritmia, hän voi käyttää tietoa hyödyksi ja aiheuttaa palvelunestohyökkäyksen.

Single-pivot-pikalajitteluista suoriutui selkeästi parhaiten järjestetylle datalle ohjelma 3 eli satunnaisen alkion valitseminen pivot-arvoksi. Ohjelman 3 oli yhtä tehokas sekä järjestämättömälle että valmiiksi järjestyksessä olevalle datalle. Odotuksista poiketen ohjelma 4 eli pikalajittelu, jossa pivot-arvoksi valitaan mediaani taulukon kolmesta alkiosta, ei suoriutunut valmiiksi järjestyksessä olevan datan järjestämisessä tehokkaasti. Tämä johtunee algoritmin toteutusriippuvaisista asioista, joita tulisi hioa ennen kuin algoritmia kannattaa käyttää tällaisenaan ohjelmistossa.

Ohjelma 5 kuvaa dual-pivot-pikalajittelun huonointa tapausta, koska se valitsee aina taulukon pienimmän ja suurimman alkion pivot-arvoiksi. Sen vuoksi sitä kannattaa verrata ohjelmaan 2. Ohjelma 5 kuten ohjelma 2 ei suoriutunut miljoonan alkion kokoisen tietojoukon järjestämisestä, mutta ohjelma 5 suoriutui sadantuhannen alkion kokoisen tietojoukon järjestämisestä selkeästi nopeammin kuin ohjelma 2. Tämä johtuu siitä, vaikka ohjelma 5 on huonoin mahdollinen tapaus, rekursiivisten kutsujen sarjasta ei tule yhtä syvä kuin ohjelman 2 tapauksessa, koska käytetään kahta pivot-arvoa.

5. YHTEENVETO

Tämän työn tarkoituksena oli selvittää pikalajittelun ja lomituslajittelun soveltuvuus eri käyttötapauksiin. Suoritusaikatesteistä voidaan havaita, että mikäli ohjelma käsittelee usein järjestyksessä olevaa dataa tai melkein järjestyksessä olevaa dataa, perinteinen pikalajittelu ei ole hyvä vaihtoehto taulukon järjestämiseksi. Jos tiedetään, että ohjelma käsittelee satunnaisesti melkein järjestyksessä olevaa dataa, on kokonaistehokkuuden kannalta järkevintä käyttää lomituslajittelua, koska sillä ei ole yhtä tehotonta huonointa tapausta kuin pikalajittelulla.

Multi-pivot pikalajittelun käyttäminen voi kannattaa, jos ohjelma käsittelee jatkuvasti hyvin suuria tietojoukkoja. Tässä tapauksessa kannattaa kuitenkin huomioida, että algoritmi on toteutettu mahdollisimman tehokkaasti eikä algoritmin tehokkuus kärsi huonosti valitun pivot-arvon vuoksi.

Single-pivot-pikalajittelu on hyvä vaihtoehto, jos käsitellään pienempiä tietojoukkoja tai käsiteltävä tietojoukko on mahdollisimman sekaisesti olevaa dataa. Pivot-arvon valitseminen satunnaisesti taulukosta ei ole hyvä tapa valita pivot-arvoa, koska satunnaisen numeron arpominen on raskas operaatio.

Lisäksi sopivan järjestysalgoritmin valitsemisessa kannattaa huomioida myös muut asiat kuin pelkkä tehokkuus. Jos ohjelma käsittelee hyvin suuria tietojoukkoja, jotka eivät mahdu kerrallaan prosessoivan tietokoneen muistiin, kannattaa käyttää lomituslajittelun ulkoista järjestelyä. Tällä tavoin järjestettävää tietojoukkoa ei tarvitse lukea kokonaisuudessaan prosessoivan tietokoneen muistiin. Lomituslajittelu on valittava myös, jos ohjelma tarvitsee stabiilin järjestysalgoritmin, koska pikalajittelu ei ole stabiili järjestysalgoritmi.

Jos prosessoivan tietokoneen on mahdollista käyttää useampaa kuin yhtä prosessoria järjestysalgoritmin suorittamiseen, sekä pikalajittelua että lomituslajittelua on mahdollista käyttää. Rinnakkaistaminen vaatii kuitenkin koordinointia eri prosessoreiden kesken, joten algoritmin toteutus on syytä tehdä huolella ja oikein.

LÄHTEET

[1] Kaliski, B. (2011). O-Notation. In Encyclopedia of Cryptography and Security (Vol. 2). Saatavissa: https://doi-org.libproxy.tuni.fi/10.1007/978-1-4419-5906-5_423

[2] Conery, J., & Foglio, P. (2011). Explorations in computing: an introduction to computer science. CRC Press. Saatavissa: https://ebookcentral.proquest.com/lib/tampere/detail.action?pq-origsite=primo&docID=1648328

[3] Stephens, R. (2013). Essential Algorithms: A Practical Approach to Computer Algorithms (1st edition). Wiley. Saatavissa: https://learning.oreilly.com/library/view/essential-algorithms-a/9781118612101/

[4] QuickSort Algorithm. Nlogn. Viitattu 17.4.2021. Saatavissa: https://nlogn.in/quicksort-algorithm/

[5] Aumüller, M., Dietzfelbinger, M., & Klaue, P. (2016). How Good Is Multi-Pivot Quicksort? ACM Transactions on Algorithms, *13*(1), 1–47. Saatavissa: https://doi.org/10.1145/2963102

[6] Aumüller, M., & Dietzfelbinger, M. (2016). Optimal Partitioning for Dual-Pivot Quicksort. ACM Transactions on Algorithms, 12(2), 1–36. Saatavissa: https://doi.org/10.1145/2743020

[7] Elhaiani, S. (Päivitetty viimeksi 12.8.2020). Dual pivot Quicksort. GeeksforGeeks. Viitattu 22.3.2021. Saatavissa: https://www.geeksforgeeks.org/dual-pivot-quicksort/

[8] Azar, E., & Alebicto, M. (2016). Swift Data Structure and Algorithms (1st ed.). Packt Publishing, Limited. Saatavissa: https://learning.oreilly.com/library/view/swift-data-structure/9781785884504/