Tampereen yliopisto

Tomi Koskinen

LAJITTELUALGORITMIN VALITSEMINEN ERI TILANTEISSA

Kandidaatintutkielma
Informaatioteknologian ja viestinnän tiedekunta
Tarkastajat: Petri Kannisto
Toukokuu 2021

TIIVISTELMÄ

Tomi Koskinen: Lajittelualgoritmin valitseminen eri tilanteissa
Kandidaatintutkielma
Tampereen yliopisto
Tietotekniikan kandidaatin tutkinto-ohjelma
Toukokuu 2021

Lajittelualgoritmeja käytetään esimerkiksi hakukoneiden tulosten, kauppojen tuotteiden ja musiikkipalvelujen kappaleiden esittämisessä, joten lajittelualgoritmitien nopeus on usein suoraan verrannollinen koko ohjelmiston nopeuteen. Tässä työssä tutkitaan eri lajittelualgoritmien ominaisuuksia, hyötyjä ja haittoja relevantin kirjallisuuden ja empiiristen tutkimusten avulla. Tämän kautta yritetään selvittää, että mitkä lajittelualgoritmit ovat tehokkaimpia mihinkin tilanteisiin. Tehokkaimman lajittelualgoritmin valitsemiseen vaikuttaa esimerkiksi alkioiden määrä, saatavilla olevan muistin määrä, alkioiden suhteellisen järjestyksen ylläpitäminen, alkioiden alkuperäinen järjestys, alkioiden jakauma ja samanarvoisten alkioiden yleisyys. Työssä keskitytään yksiulotteisen taulukon alkioiden lajittelemiseen.

Työssä käydään läpi 11 lajittelualgoritmia, joiden valikoitumiseen vaikutti algoritmien hyvä tehokkuus eri tilanteissa ja niiden toistuvuus relevantissa kirjallisuudessa. Valitut lajittelualgoritmit ovat: insertion sort, merge sort, heap sort, quicksort, selection sort, shellsort, introsort, timsort, bucket sort, counting sort ja radix sort. Näistä kolme viimeistä, eli bucket sort, counting sort ja radix sort ovat vertailuun perustumattomia lajittelualgoritmeja, jotka nimensä mukaan eivät käytä alkioiden lajittelemiseen suoraa alkioiden arvojen vertailua. Työssä tutkitaan lajittelualgoritmien aika- ja tilavaativuuksia, vakautta ja suosituimpia optimointeja.

Tutkimuksista saatiin paljon tuloksia. Jokaiselle valitulle lajittelualgoritmille löytyi tilanne, jossa kyseinen algoritmi on optimaalisin valinta. Insertion sort osoittautui nopeimmaksi pienikokoisilla, alle 16 alkion pituisilla taulukoilla. Introsort on yleiskäyttöisin ja nopein epävakaa lajittelualgoritmi, soveltuen hyvin suurten alkiomäärien lajitteluun ja lajittelualgoritmiksi standardikirjastoon. Timsort taas oli yleiskäyttöisin ja nopein vakaa lajittelualgoritmi. Counting sort osoittautui hyvin tehokkaaksi, kun taulukon alkiot voivat saada rajatun määrän eri arvoja.

Avainsanat: Lajittelualgoritmit, järjestysalgoritmit, algoritmit, aikavaativuus, tilavaativuus

SISÄLLYSLUETTELO

1. JOHDANTO

Lajittelualgoritmit ovat aina olleet tärkeä osa ohjelmistoja, koska informaation löytäminen listasta on aina mielekkäämpää niin tietokoneille kuin ihmisillekin, jos lista on järjestyksessä. Lajittelualgoritmeja käytetään myös tärkeänä osana muissa toiminnoissa, kuten hakualgoritmeissa ja kun halutaan selvittää, sisältääkö kaksi eri taulukkoa samat alkiot. Lajittelua käytetään esimerkiksi hakukoneiden tulosten, kauppojen tuotteiden ja musiikkipalvelujen kappaleiden esittämisessä. Täten lajittelualgoritmitien nopeus on usein suoraan verrannollinen koko ohjelmiston nopeuteen. Pienissä alkiomäärissä (10-1000) optimaalisen lajittelualgoritmin käyttäminen nopeuttaa lajittelun suoritusaikaa milli- ja sadasosasekunneilla. Suurilla alkiomäärillä (100 000-n) optimaalisen lajittelualgoritmin käyttö saattaa olla useita sekunteja nopeampi ja täysin epäsopivan lajittelualgoritmin käyttö voi olla kymmeniä sekunteja hitaampi optimaaliseen verrattuna.

Tämä työ on kirjallisuusselvitys lajittelualgoritmeista. Työn tarkoitus on selvittää teoreettisen ja empiirisen tutkimuksen kautta tehokkaimmat lajittelualgoritmit erilaisissa olosuhteissa, joissa lajiteltavat alkiot ovat yksiulotteisessa taulukossa. Työssä ei siis mennä syvälle algoritmien yksityiskohtaisiin toteutuksiin vaan yritetään löytää paras mahdollinen lajittelualgoritmi erilaisiin yleisiin tilanteisiin. Asiat, jotka vaikuttavat lajittelualgoritmin valitsemiseen ovat: alkioiden määrä, saatavilla olevan muistin määrä, alkioiden suhteellisen järjestyksen ylläpitäminen, alkioiden alkuperäinen järjestys, alkioiden jakauma ja samanarvoisten alkioiden yleisyys.

Luvussa 2 kerrotaan lajittelualgoritmeista yleisesti ja avataan algoritmien ominaisuuksiin liittyviä termejä, jonka jälkeen luvussa 3 esitellään kattava määrä lajittelualgoritmeja samalla käyden läpi ja vertaillen niiden ominaisuuksia. Luvussa 4 tulkitaan empiirisiä tutkimuksia lajittelualgoritmien nopeuksista ja lopuksi luvussa 5 kootaan tulokset yhteen ja vastataan tutkimuskysymykseen.

2. LAJITTELUALGORITMIT YLEISESTI

Tässä työssä lajittelualgoritmit ottavat syötteenä taulukon, joka sisältää alkioita. Nämä alkiot voivat olla esimerkiksi kokonaislukuja tai merkkijonoja. Lajittelualgoritmin tehtävä on palauttaa taulukko, jossa annetut alkiot ovat järjestyksessä. Numeroiden kohdalla tämä on usein loogisesti numerojärjestys pienimmästä isoimpaan, merkkijonojen kohdalla voidaan seurata aakkosjärjestystä.

Lajittelualgoritmit voidaan jakaa vakaisiin ja epävakaisiin. Vakaat lajittelualgoritmit säilyttävät syötteenä olevan taulukon saman arvoisten alkioiden alkuperäisen suhteellisen järjestyksen [1–3]. Esimerkiksi kokonaislukuja sisältävä taulukko A = [5^a, 4^b, 2^c, 2^d, 1^e] olisi vakaalla lajittelualgoritmilla lajiteltuna muodossa [1^e, 2^c, 2^d, 4^b, 5^a] (huomaa numeroiden 2 kirjaimet), mutta epävakaalla lajittelualgoritmilla se voisi olla muotoa [1^e, 2^d, 2^c, 4^b, 5^a].

Algoritmin suoritusaikaan vaikuttaa aina käsiteltävien alkioiden määrä n ja se, kuinka monta operaatiota alkioille tehdään. Algoritmien suoritusaikaa ja muistinhallintaa voidaan arvioida asymptoottisten notaatioiden avulla. Notaatiot kuvaavat algoritmin kasvunopeutta alkioiden määrän n funktiona. Big O -notaatio kuvaa algoritmin maksimaalista operaatioiden käytön määrää eli algoritmin hitainta mahdollista aikaa [2]. Jos algoritmissa tehdään jokaiselle alkiolle 2 operaatiota eli operaatioiden määrä on 2n, tällöin Big O notaation arvo eli aikavaativuus on lineaarinen O(n). Jos algoritmissa alkiot käydään kahdessa sisäkkäisessä silmukassa läpi, minkä jälkeen vielä erikseen yhdessä silmukassa, niin operaatioiden määrä on $n^2 + n$ ja aikavaativuus $O(n^2)$. Notaatio siis jättää huomioimatta vakiokertoimet ja ottaa aina vain funktion isoimman polynomin huomioon [2]. Tällä tavalla algoritmien aikavaativuuksien vertailu on yksinkertaisempaa ja suurilla alkiomäärillä suoritusaikaan vaikuttavin tekijä jää tietoon. Tilavaativuuteen vaikuttaa operaatioiden sijaan muistin käyttö. Jos algoritmi vie vakiomäärän muistia riippumatta alkioiden määrästä, niin sen tilavaativuus on O(1), eli se on paikallaan toimiva (in-place). Jos taas algoritmi vie lineaarisesti lisämuistia jokaista alkiota kohden, niin sen tilavaativuus on O(n).

Lajittelualgoritmit voidaan myös jakaa vertailuun perustuviin lajittelualgoritmeihin ja niihin, jotka eivät perustu vertailuun. Vertailuun perustuvilla lajittelualgoritmeilla on sellainen asymptoottinen ominaisuus, että niiden aikavaativuus ei voi olla nopeampi kuin O(n log n) [1, 2]. Vertailuun perustumattomat lajittelualgoritmit taas voivat päästä jopa O(n) aikavaativuuteen [1, 2].

3. LAJITTELUALGORITMEJA

Tässä luvussa käydään läpi 11 algoritmia ja kerrotaan, milloin ne ovat tehokkaimmillaan ja milloin mahdollisesti eivät. Käsiteltävien lajittelualgoritmien valikoitumiseen vaikutti algoritmien hyvä tehokkuus eri tilanteissa ja niiden toistuvuus relevantissa kirjallisuudessa. Bubble sortia (kuplalajittelu) ei käsitellä tässä työssä, koska sillä ei ole mitään ominaisuutta, missä se olisi parempi kuin muut lajittelualgoritmit [4].

3.1 Vertailuun perustuvat lajittelualgoritmit

3.1.1 Insertion sort

Insertion sort (lisäyslajittelu) on lajittelualgoritmi, joka lajittelee taulukon kulkemalla sitä alusta loppuun samalla siirtäen alkioita oikealle paikalleen [1–3]. Jos toiveena on kirjoittaa mahdollisimman vähän koodia, insertion sort on yksinkertaisuudessaan siihen sopiva valinta [2].

Insertion sortin aikavaativuus on $O(n^2)$, joten se ei sovellu käytettäväksi tilanteisiin, joissa täytyy lajitella suuria taulukoita. Sen sijaan lyhyen ja tehokkaan sisäisen koodisilmukkansa vuoksi insertion sort on erittäin tehokas lajittelemaan pieniä määriä alkioita. Tämän takia insertion sortia käytetään myös usein muissa lajittelualgoritmeissa tai niiden yhdistelmissä, kun lajitellaan pieniä taulukoita. Se, mikä on tarpeeksi pieni määrä alkioita, riippuu käytettävästä tietokoneesta ja ohjelmointikielestä. [1–3]

Toteutuksensa vuoksi insertion sort lajittelee jo valmiiksi lajitellun taulukon lineaarisessa ajassa O(n). Tämä on hyvin tärkeä ominaisuus, sillä millään muulla vertailuun perustuvalla lajittelualgoritmilla ei sitä ole. Käytännössä ei ole harvinaista, että vastaan tulee taulukoita, jotka ovat jo lähes tai täysin valmiiksi järjestyksessä. Tällöin insertion sort on oiva valinta suuremmillekin taulukoille. [2]

Insertion sort on paikallaan toimiva, eli sillä on O(1) tilavaativuus. Insertion sort on myös vakaa lajittelualgoritmi, joten se sopii tilanteisiin, joissa alkioiden suhteellinen järjestys on tärkeää. [1–3]

3.1.2 Merge sort

Merge sort (lomituslajittelu) on lajittelualgoritmi, joka seuraa [1–3] hajota ja hallitse -suunnitteluperiaatetta. Merge sortissa taulukko jaetaan kahtia rekursiivisesti, kunnes jäljellä on vain yksittäisiä alkioita, joita vertailemalla kootaan järjestetty taulukko. [1, 2]

Merge sortin aikavaativuus on O(n log n) syötteen järjestyksestä riippumatta. O(n log n) on paras mahdollinen vertailuun perustuville lajittelualgoritmeille, minkä takia merge sort onkin hyvä valinta suurille taulukoille. Merge sortin haittapuoli on sen vaatima O(n) tilavaativuus, eli algoritmi vie lineaarisen määrän lisämuistia, mikä tarkoittaa, että merge sort ei sovellu tilanteisiin, joissa muistia ei ole tarpeeksi. [1–3]

Merge sortin tehostamiseen on olemassa muutoksia, kuten insertion sortin käyttäminen, kun jaettujen taulukoiden alkiomäärän koko menee esimerkiksi alle 15. Tämän parantaa suoritusaikaa 10-15%. Toinen muunnos on taulukon järjestyksen tarkastaminen algoritmin alussa, jolloin jo järjestyksessä olevan taulukon suoritusaika on lineaarinen, kuten insertion sortissa. Myös lomitukseen käytetyn aputaulukon kopioimiseen käytetty aika voidaan poistaa vaihtamalla syöte taulukon ja aputaulukon rooleja eri rekursiotasoilla. [1] Merge sort on insertion sortin lailla vakaa lajittelualgoritmi [1–3]

3.1.3 Heap Sort

Heap sort (kekolajittelu) luo taulukon alkioista kekorakenteen, jonka jälkeen ottaa keon juuressa olevan arvon järjestettyyn taulukkoon. Tämän jälkeen keko täytyy järjestää uudestaan, että saadaan suurin/pienin arvo taas juureen. Tätä toistetaan kunnes keko on tyhjä. [1–3]

Kuten merge sort, heap sortin aikavaativuus on O(n log n), mutta toisin kuin merge sort, heap sort on paikallaan toimiva, eli se vie vakiomäärän tilaa. Tilan loppumisesta ei siis tarvitse huolehtia, mutta heap sort on silti käytännössä hieman merge sortia hitaampi, koska sen Big O notaatiossa poistetut vakiokertoimet ovat isommat, eli heap sortissa tehdään enemmän operaatioita. [1]

Heap sorttia käytetään usein esimerkiksi sulautetuissa järjestelmissä, kun tilaa ei ole paljoa, mutta sitä käytetään vain harvoin moderneissa järjestelmissä, koska se käyttää välimuistia epätehokkaasti. [1] Toisin kuin Insertion sort ja merge sort, heap sort ei ole vakaa algoritmi [1–3]

3.1.4 Quicksort

Quicksort (pikalajittelu) valitsee pivot-alkion, jonka vasemmalle puolelle siirretään pivot-alkiota pienemmät alkiot ja oikealle puolelle suuremmat alkiot. Pivot-alkio valitaan uudelleen molemmille puolille rekursiivisesti, kunnes koko taulukko on järjestetty. [1–3]

Quicksortin aikavaativuus on huonoimmassa tapauksessa $O(n^2)$, tämä saattaa esiintyä tilanteissa, jossa taulukko on jo järjestyksessä oikein tai väärinpäin sekä silloin kun kaikki alkiot ovat samoja. Quicksort on kuitenkin keskiarvoisesti O(n log n) ja sillä on myös hyvin tehokas sisäinen koodisilmukka, eli asymptoottiset vakiokertoimet ovat pienet,

mikä johtaa siihen, että quicksort on keskiarvoisesti huomattavasti nopeampi, kuin merge sort ja heap sort. Insertion sort loistaa vielä pienillä alkiomäärillä. [1–3]

Quicksorttiin on olemassa myös monia muutoksia. Jo järjestetyn taulukon huonoimman tapauksen estämiseksi pivot-alkio voidaan valita satunnaisesti. Tämä johtaa siihen, että huonoin tapaus toteutuu vain silloin, jos satunnaisesti valittu pivot-alkio on aina kyseisen alataulukon reunassa. Toinen suosittu tapa valita pivot-alkio on ottaa 3 taulukon alkiota, ja valita niistä mediaaniarvo. Tämä aiheuttaa lisää laskemista, mutta se vähentää sitä todennäköisyyttä, että pivot-alkio olisi pienin/suurin arvo. Kolmen alkion mediaani toteutus antaa 5% keskiarvoisen tehokkuuden paranemisen. Myös quicksortissa voidaan käyttää insertion sorttia, kun päästään tarpeeksi pieniin alataulukkoihin. [1, 2]

Jos taulukossa on paljon samoja alkioita, voidaan tehdä muokkaus, jossa pivot-alkion kanssa samaa arvoa olevista alkioista luodaan kolmas ryhmä, joka voidaan vain siirtää järjestettyyn taulukkoon pivot-alkion perään [1]. Quicksort on merge sortin tavoin hajoita ja hallitse algoritmi, mutta se ei ole paikallaan toimiva eikä vakaa. Quicksortin tilavaativuus on O(log n). [1]

3.1.5 Selection sort

Selection sort (valintalajittelu) järjestää taulukon hakemalla aina pienimmän/suurimman alkion ja asettamalla sen oikeaan paikkaan taulukossa. Se onkin yksi yksinkertaisimmista lajittelualgoritmeista. [1, 2, 4]

Selection sort suorittaa keskiarvoisesti n määrän alkioiden paikkojen vaihteluja ja aina n^2 määrän alkioiden vertailuja. Täten aikavaativuus on $O(n^2)$ kaikissa tapauksissa. [1] Alkioiden paikkojen minimaalinen vaihtelu onkin selection sortin ainut etu muihin lajittelualgoritmeihin nähden. Selection sort on epävakaa ja sen tilavaativuus on O(1) [1]

3.1.6 Shellsort

Shellsort (Shell-lajittelu) perustuu insertion sortiin, mutta toisin kuin insertion sort, shellsortin ideana on taulukossa kaukana toisistaan olevien alkioiden paikkojen vaihtaminen. Koko taulukko jaetaan alitaulukoihin, jotka muodostuvat ennalta määritellyn numeron h perusteella, tämä numero merkitsee sitä paikkojen väliä, mikä alitaulukoissa olevien alkioiden etäisyys on alkuperäisessä taulukossa. Alitaulukot järjestetään insertion sortilla, jonka jälkeen h:ta aletaan pienentää, mikä johtaa alitaulukoiden koon kasvamiseen ja lopulta takaisin yhteen kokonaiseen taulukkoon. [1, 4]

Shellsort on insertion sortia huomattavasti parempi suurille alkiomäärille. Tarkkaa aikavaatimusta shellsortille ei ole saatu laskettua. Se on arvioitu olevan huonoimmassa tapauksessa noin $O(n^{3/2})$, mutta käytännössä keskiarvoisesti lähellä merge sortin tehokkuutta. [1] Shellsort on paikallaan toimiva ja epävakaa [1, 4].

3.1.7 Introsort

Introsort on hybridilajittelualgoritmi, joka käyttää quicksorttia, kunnes quicksortin rekursio ylittää log n syvyyden, jolloin introsort vaihtaa lajittelun heap sorttiin. [5] Lopulta, kun jäljellä olevien lajiteltavien alkioiden määrä laskee tietyn rajan ali, niin loput alkiot lajitellaan insertion sortin avulla. [2]

Introsortin aikavaativuus on O(n log n) ja se on paikallaan toimiva, mutta kuten quicksort, se ei ole vakaa. Introsorttia käytetään esimerkiksi C++ standardikirjaston lajittelualgoritmina. [2, 5]

3.1.8 Timsort

Timsort on hybridilajittelualgoritmi, joka käyttää merge sortia ja binääristä insertion sortia. Se on luotu ajatellen reaalimaailman dataa ja käyttääkin hyväkseen lajiteltavan taulukon alkioiden säännöllisyyksiä. [6, 7]

Pahin ja keskimääräinen tapaus on sama kuin merge sortilla O(n log n). Parhaimman O(n) tapauksen timsort saa jo valmiiksi järjestetyllä taulukolla. Merge sortin tapaan timsort on myös vakaa ja tilavaativuus on O(n). [6, 7]

3.2 Vertailuun perustumattomat lajittelualgoritmit

3.2.1 Bucket sort

Bucket sort (nippulajittelu) jakaa n määrän alkioita n määrään nippuja. Alkioiden jakaminen nippuihin tapahtuu etukäteen valitun hajautusalgoritmin avulla. Nipuissa alkiot lajitellaan oikeaan järjestykseen insertion sortin avulla ja lopuksi niput tuodaan yhteen pienimmästä suurimpaan. [2, 3]

Hajautusalgoritmin tarkoitus on jakaa alkiot mahdollisimman tasaisesti nippuihin, jos tämä onnistuu, niin bucket sortin aikavaatimus on O(n + b), jossa b on nippujen määrä. Alkioiden on siis oltava jollain tavalla tasaisesti hajautettavissa tai muuten bucket sort lähenee insertion sortin $O(n^2)$ aikavaatimusta. [2, 3] Bucket sort on vakaa ja sen tilavaativuus on O(n) ylimääräisten nippujen luonnin vuoksi [2].

3.2.2 Counting sort

Counting sort (laskentalajittelu) olettaa, että jokainen alkio on numero väliltä 1-k. lajittelu tapahtuu k pituisen aputaulukon ja siihen kasaantuvan jakauman avulla. Counting sort onkin hyvä valinta silloin, kun lajiteltavassa taulukossa k ei ole kovin suuri. Aikavaatimus ja tilavaatimus ovat molemmat O(n + k) ja algoritmi on vakaa. [3]

3.2.3 Radix sort

Radix sort (kantalukulajittelu) lajittelee d numeroisia lukuja siten, että taulukko järjestetään lukujen vähiten merkitsevän numeron mukaan järjestykseen jollain vakaalla vertailuun perustuvalla lajittelualgoritmilla. Taulukkoa järjestetään uudestaan aina seuraavaksi vähiten merkitsevän numeron mukaan, kunnes päästään lukujen loppuun ja koko taulukko on järjestyksessä. [3]

Radix sorting aikavaativuus on siis O(n · d). Käytettävä vakaa lajittelualgoritmi on usein counting sort, mikä johtaa radix sortin O(n + d) tilavaativuuteen. [3]

4. EMPIIRISET TUTKIMUKSET

Teorian tueksi tässä luvussa tutustutaan empiirisiin tutkimuksiin, jotka sisältävät luvussa 3 esitettyjä lajittelualgoritmeja. Ensin käydään tarkemmin läpi kaksi kattavaa tutkimusta. Lopuksi tutkitaan tuloksia niille lajittelualgoritmeille, joita ei käsitelty ensimmäisissä tutkimuksissa.

4.1 OpenDSA:n tutkimukset

OpenDSA on tehnyt empiirisiä aikavertailuja muutamalle yleiselle algoritmille ja niiden optimoiduille versioille heidän opetuskäyttöön tarkoitetussa hypertekstikirjassaan. Alla on taulukko 1 vertailujen tuloksista. Kaikki vertailut tehtiin 3.4 GHz Intel Pentium 4 prosessorilla ja Linux käyttöjärjestelmällä. Taulukossa näkyy taulukkojen alkioiden määrä, joka on väliltä 10–1 000 000. Viimeistä kahta saraketta lukuun ottamatta, kaikki taulukkojen alkiot ovat satunnaisia kokonaislukuja. Viimeiset kaksi saraketta käsittelevät järjestyksessä ja käänteisessä järjestyksessä olevia taulukoita, joissa on molemmissa 10 000 alkiota. Taulukon ajat ovat millisekunteina. [8]

Taulukko 1: *Lajittelualgoritmien empiirinen vertailu, perustuu lähteeseen* ***[8]***

Lajittelualgoritmi	10	100	1000	10K	100K	1M	lajiteltu	käänteisesti lajiteltu
Insertion	.00023	.007	0.66	64.98	7381.0	674420	0.04	129.05
Bubble	.00035	.020	2.25	277.94	27691.0	2820680	70.64	108.69
Selection	.00039	.012	0.69	72.47	7356.0	780000	69.76	69.58
Shell	.00034	.008	0.14	1.99	30.2	554	0.44	0.79
Shell/O	.00034	.008	0.12	1.91	29.0	530	0.36	0.64
Merge	.00050	.010	0.12	1.61	19.3	219	0.83	0.79
Merge/O	.00024	.007	0.10	1.31	17.2	197	0.47	0.66
Quick	.00048	.008	0.11	1.37	15.7	162	0.37	0.40
Quick/O	.00031	.006	0.09	1.14	13.6	143	0.32	0.36
Heap	.00050	.011	0.16	2.08	26.7	391	1.57	1.56
Heap/O	.00033	.007	0.11	1.61	20.8	334	1.01	1.04
Radix/4	.00838	.081	0.79	7.99	79.9	808	7.97	7.97
Radix/8	.00799	.044	0.40	3.99	40.0	404	4.00	3.99

Taulukosta 1 nähdään, että insertion sort todella on nopein hyvin pienillä alkiomäärillä ja se päihittää suurilla alkiomäärillä myös muut keskiarvoisesti $O(n^2)$ algoritmit, kuten bubble sortin ja selection sortin. Selection sort olisi voinut suoriutua paremmin insertion sortiin nähden, jos alkioiden koko olisi ollut suurempi, tehden alkioiden paikkojen vaihtamisesta hitaampaa [8]. Insertion sort on myös nopein järjestyksessä ja hitain käänteisessä järjestyksessä olevien taulukkojen lajittelemisessa saadessaan molemmat parhaimman ja pahimman tapauksensa.

Nopeimmat lajittelualgoritmit suurilla alkiomäärillä olivat nopeusjärjestyksessä: quicksort, merge sort, heap sort ja shellsort. Tämä ei tullut luvun 3 esittelyjen jälkeen yllätyksenä. Kummassakaan quicksortin versiossa pivot-alkiota ei valita satunnaisesti tai mediaanin avulla vaan molemmissa toteutuksissa taulukon keskimmäinen alkio valitaan [8] Tämä antaa quicksortille pahimman tapauksen sijaan parhaimman, kun taulukko on järjestetty tai käänteisesti järjestetty. Optimoidut quicksort ja merge sort käyttävät insertion sortia pienille alataulukoille, mikä johtaa niiden insertion sortia lähenevään nopeuteen pienillä alkiomäärillä.

Radix sort suoriutui yllättävän heikosti. Parempaan tulokseen olisi päästy käyttämällä bittien siirtoa. [8] Taulukosta 1 on hyvä huomata radix sortin ajankäytön nousevan täysin lineaarisesti taulukon alkioiden määrään nähden. Kaikki muut lajittelualgoritmit ylittävät lineaarisuuden. Radix sortin voisi siis olettaa päihittävän muut algoritmit alkioiden määrän kasvaessa vielä monikymmenkertaisesti.

4.2 Hulínin tutkimukset

Matej Hulínin kandidaatintutkielmassa tutkittiin laajasti eri lajittelualgoritmien tehokkuuksia erilaisilla syötetaulukoilla. Tutustumme tarkemmin kolmanteen mittaukseen, jossa vertailussa olivat: [9]

- Timsort
- Quicksort satunnaisella pivot-alkiolla (quicksort random)
- Quicksort kolmen luvun mediaani pivot-alkiolla (quicksort median)
- Merge sort
- Heap sort

Mittausympäristönä toimi C++ ohjelmointikieli, Linux käyttöjärjestelmä ja 3.0 GHz Intel Core i7-4500u prosessori. Myös käyttöjärjestelmän sivutilan käyttö otettiin pois käytöstä. Syötetaulukot koostuivat kokonaisluvuista, jotka olivat `uint_fast32_t` tietotyyppiä ja

merkkijonoista, jotka olivat `std::string` tietotyyppiä. Tässä luvussa tarkastelemme vain kokonaislukuja, koska merkkijonojen lajittelun tulokset eivät eronneet merkittävästi kokonaislukujen lajittelusta. Syötetaulukot olivat kahdeksanlaisia: [9]

- Lajiteltu (sorted)
- Käänteisesti lajiteltu (reverse sorted)
- Osittain lajiteltu (nearly sorted)
- Käänteisesti osittain lajiteltu (nearly reverse sorted)
- Satunnainen (random)
- K-monotone, jossa K pituiset sarjat taulukossa ovat lajiteltu tai käänteisesti lajiteltu
- K-shuffled, jossa K pituiset lajitellut sarjat taulukossa ovat sekoitettu
- K-restricted, jossa K on mahdollisten uniikkien arvojen määrä

Kolmannessa mittauksessa taulukkojen koko vaihteli väliltä 100 000 000 – 500 000 000. Taulukossa 2 näkyy lajiteltujen ja käänteisesti lajiteltujen taulukoiden mittaukset. Timsort loistaa lajitelluissa mittauksissa binäärisen insertion sortinsa avulla. Nopein käänteisesti

Taulukko 2: *Lajitellut ja käänteisesti lajitellut taulukot, perustuu lähteeseen* ***[9]***

	Käänteisesti lajitellut taulukot					**Lajitellut taulukot**				
	100M	**200M**	**300M**	**400M**	**500M**	**100M**	**200M**	**300M**	**400M**	**500M**
Heap sort	21.15	43.69	67.31	90.47	114.48	18.92	38.82	59.03	79.55	100.25
Merge sort	7.00	14.45	22.69	30.46	37.56	5.79	11.97	18.20	23.90	29.70
Quicksort median	2.84	5.97	9.50	12.80	15.89	1.84	3.82	5.54	7.65	9.49
Quicksort random	4.36	8.85	13.72	18.90	22.82	4.42	9.11	13.58	18.08	23.32
Timsort	3.52	7.57	12.31	16.45	20.32	0.26	0.52	0.75	1.01	1.14

lajitelluissa ja toiseksi nopein lajitelluissa mittauksissa on odotetusti quicksort medinaa-nipivotilla. Taulukon 3 osittaisen lajittelun mittauksissa mediaani-quicksort on nopein ja heapsort taas hitain molemmissa.

Taulukko 3: *Osittain lajitellut ja osittain käänteisesti lajitellut taulukot, perustuu lähteeseen* ***[9]***

	Osittain käänteisesti lajitellut taulukot					**Osittain lajitellut taulukot**				
	100M	**200M**	**300M**	**400M**	**500M**	**100M**	**200M**	**300M**	**400M**	**500M**
Heap sort	32.67	69.35	108.11	148.29	187.95	27.17	57.46	88.79	121.89	153.17
Merge sort	7.96	16.97	25.81	33.70	43.44	7.60	15.89	24.54	31.82	41.33
Quicksort median	5.17	11.61	17.04	24.55	30.29	3.14	6.50	9.87	13.14	16.70
Quicksort random	7.44	15.32	23.66	31.60	39.72	5.46	11.10	16.84	22.78	28.40
Timsort	6.91	14.39	21.31	30.01	39.79	5.99	12.83	19.50	26.59	34.48

Taulukossa 4 on satunnaisten taulukkojen mittaukset, jossa merge sort oli timsortia nopeampi ja molemmat quicksortit selvästi nopeimpia.

Taulukko 4: *Satunnaiset taulukot, perustuu lähteeseen* ***[9]***

	100M	**200M**	**300M**	**400M**	**500M**
Heap sort	93.31	205.50	327.46	457.81	587.80
Merge sort	16.38	34.33	52.05	70.37	89.50
Quicksort median	10.96	22.96	34.83	47.47	59.75
Quicksort random	12.19	25.43	38.45	51.86	65.04
Timsort	18.68	39.29	59.82	80.99	102.03

Taulukossa 5 nähdään K-monotone taulukkojen vertailut. K:n arvo oli mittauksissa 2, 4 ja 32. Merge sort on tässäkin timsortia hieman nopeampi ja quicksortit jälleen nopeimpia.

Taulukko 5: *K-monotone taulukot, perustuu lähteeseen* ***[9]***

	2-monotone				
	100M	**200M**	**300M**	**400M**	**500M**
Heap sort	93.00	206.62	330.19	454.51	585.70
Merge sort	16.10	33.11	51.10	68.57	86.71
Quicksort median	11.01	22.59	34.90	47.12	59.88
Quicksort random	12.17	24.88	37.98	51.51	65.33
Timsort	18.53	58.95	58.84	80.95	102.38
	4-monotone				
	100M	**200M**	**300M**	**400M**	**500M**
Heap sort	91.88	205.08	328.24	458.52	583.99
Merge sort	15.44	32.32	49.53	66.73	84.81
Quicksort median	10.88	22.86	32.73	46.66	59.34
Quicksort random	11.85	25.07	37.92	51.30	65.47
Timsort	18.20	38.59	58.52	79.52	101.01
	32-monotone				
	100M	**200M**	**300M**	**400M**	**500M**
Heap sort	91.78	297.34	328.58	458.82	581.90
Merge sort	14.27	30.09	45.25	61.67	78.14
Quicksort median	10.18	21.48	32.28	44.04	55.89
Quicksort random	11.27	24.11	35.98	48.48	61.49
Timsort	15.62	34.21	51.66	69.09	95.11

Taulukossa 6 on K-shuffled taulukoiden mittaukset. K:n arvo oli 16, 32 ja 64. Timsort suoriutui tästä parhaitan. Mediaani-quicksort oli lähellä toisena ja satunnainen quicksort hieman kauempana kolmantena. Merge sort oli neljäs ja suoriutui huomattavasti huonommin kuin timsort. Heap sort oli jälleen selvästi hitain.

Taulukko 6: *K-shuffled taulukot, perustuu lähteeseen* ***[9]***

	16-shuffled				
	100M	**200M**	**300M**	**400M**	**500M**
Heap sort	19.10	39.11	59.82	80.31	101.78
Merge sort	7.16	14.69	22.38	30.35	37.39
Quicksort median	3.86	7.88	11.96	16.28	20.11
Quicksort random	5.43	11.05	16.58	22.64	28.11
Timsort	2.79	5.58	8.71	11.22	15.15
	32-shuffled				
	100M	**200M**	**300M**	**400M**	**500M**
Heap sort	19.16	39.29	59.77	80.39	101.33
Merge sort	7.73	15.69	23.86	31.72	39.64
Quicksort median	4.19	8.51	12.67	17.42	21.55
Quicksort random	5.87	11.89	17.84	24.18	29.62
Timsort	3.49	6.99	10.33	13.94	17.89
	64-shuffled				
	100M	**200M**	**300M**	**400M**	**500M**
Heap sort	19.27	39.38	60.26	80.72	102.14
Merge sort	8.00	16.50	25.17	33.02	41.86
Quicksort median	4.37	8.95	13.46	17.83	22.60
Quicksort random	5.98	12.44	18.50	24.64	31.25
Timsort	4.02	8.21	12.14	16.05	21.04

Taulukossa 7 näkyy K-restricted taulukkojen mittaukset. K:n arvo eli uniikkien alkioiden määrä oli 26, 27 ja 28. K oli 28 taulukoille väliltä 300 000 000 – 500 000 000. Molemmat quicksortit olivat jälleen huomattavasti nopeimpia. Timsort oli hieman merge sortia nopeampi ja heap sort jälleen hitain.

Taulukko 7: *K-restricted taulukot, perustuu lähteeseen* ***[9]***

	100M	**200M**	**300M**	**400M**	**500M**
Heap sort	20.04	41.30	62.81	84.76	107.28
Merge sort	9.64	19.68	30.94	41.35	51.21
Quicksort median	3.96	8.25	12.83	17.18	21.44
Quicksort random	4.67	9.85	15.41	20.17	25.27
Timsort	8.50	17.55	27.31	35.91	46.67

Mediaani-quicksort oli kaikissa paitsi K-shuffled ja lajiteltujen taulukoiden mittauksissa nopein. Näissä kahdessa tapauksessa timsort oli nopein. Suurin ero muihin algoritmeihin mediaani quicksortilla oli osittain lajitelluissa taulukoissa, timsortilla taas selvin voitto oli täysin lajitelluissa taulukoissa. Merge sort oli timsortia nopeampi vain K-monotone ja satunnaisissa taulukoissa eikä erot olleet suuria. Satunnainen-quicksort ei ollut missää mittauksessa mediaani-quicksortia nopeampi. Heapsort oli kaikissa mittauksissa selvästi hitain.

4.3 Muut tutkimukset

Counting sort suoriutui yllättävästi erittäin hyvin kolmessa eri julkaisussa [10–12]. Tämä johtuu luultavasti siitä, että mittauksissa counting sortille on valmiiksi syötetty alkioiden maksimiarvo ja alkioiden maksimiarvo on matala. Yhdessä näistä julkaisuista alkioiden maksimiarvo oli 999 [10]. Myös radix sort suoriutui näissä julkaisuissa huomattavasti paremmin kuin OpenDSA:n taulukossa.

Kirjassa Algorithms in a Nutshell, 2nd Edition tutkitaan muun muassa bucket sortin variaatiota hash sortia. Hash sort tekee ennakkoon päätetyn määrän nippuja, toisin kuin bucket sort, joka tekee n määrän nippuja. Mittauksissa nippujen määräksi valittiin 17 576. Data koostui 26 kirjaimen pituisista merkkijonoista. [2] Satunnaisilla alkioilla taulukkojen koon ollessa 4 096 – 131 072 väliltä hash sort oli nopein algoritmi voittaen muun muassa

mediaani-quicksortin ja merge sortin. Hash sort oli kuitenkin näitä molempia ja insertion sortia hitaampi jo lajitetuilla taulukoilla.

5. ANALYYSI

Tässä luvussa kootaan ja analysoidaan kirjallisuuskatsauksesta ja empiirisistä tutkimuksista saatua tietoa. Ensin kootaan lajittelualgoritmien yleiset ominaisuudet, minkä jälkeen analysoidaan ja kootaan käsiteltyjen lajittelualgoritmien etuja ja haittoja.

Taulukkoon 8 on koottu kaikkien käsiteltyjen lajittelualgoritmien aika- ja tilavaativuudet, sekä tieto, että onko algoritmi vakaa. Shellsortin keskimääräistä tapausta ei ole teoreettisesti todistettu, joten perässä on kysymysmerkki. Lajittelualgoritmin vakaus on tärkeää, kun halutaan lajitella asioita kahden eri ominaisuuden perusteella. Tällöin ensimmäinen lajittelu voi olla epävakaa, mutta toisen lajittelun täytyy olla vakaa, jotta asiat pysyisivät myös ensimmäisen lajittelun mukaisessa järjestyksessä.

Taulukko 8: *Koonti lajittelualgoritmien ominaisuuksista*

Lajittelualgoritmi	**Pahin tapaus**	**Keskimääräinen tapaus**	**Tilavaativuus**	**Vakaa**
Insertion sort	$O(n^2)$	$O(n^2)$	O(1)	Kyllä
Merge sort	O(n log n)	O(n log n)	O(n)	Kyllä
Heap sort	O(n log n)	O(n log n)	O(1)	Ei
Quicksort	$O(n^2)$	O(n log n)	O(log n)	Ei
Selection sort	$O(n^2)$	$O(n^2)$	O(1)	Ei
Shellsort	$O(n^{3/2})$	O(n log n)?	O(1)	Ei
Introsort	O(n log n)	O(n log n)	O(log n)	Ei
Timsort	O(n log n)	O(n log n)	O(n)	Kyllä
Bucket sort	$O(n^2)$	O(n + b)	O(n)	Kyllä
Counting sort	O(n + k)	O(n + k)	O(n + k)	Kyllä
Radix sort	O(n · d)	O(n · d)	O(n + d)	Kyllä

Insertion sort osoittautui nopeimmaksi lajittelualgoritmiksi pienillä alkiomäärille. Se on kuitenkin merkittävästi O(n log n) lajittelualgoritmeja hitaampi suuremmilla alkiomäärillä. Ei voida yleisesti tarkkaan sanoa, että minkä alkiomäärän ylittyessä insertion sort kannattaisi vaihtaa johonkin toiseen lajittelualgoritmiin. Kun insertion sortia on käytetty muissa lajittelualgoritmeissa, siihen on vaihdettu, kun jäljellä on ollut enää maksimissaan 16 alkiota [2, 8, 9]. Insertion sort osoittautui myös nopeimmaksi jo lajitellun taulukon järjestämisessä, taulukon koosta riippumatta.

Merge sort on tehokas suurille alkiomäärille ja insertion sortilla optimoituna myös pienille määrille. Merge sort vie enemmän muistia kuin quicksort ja on quicksortia hitaampi, ellei quicksort kärsi pahimmasta tapauksestaan. Merge sorting edut ovat quicksortiin nähden varma O(n log n) tehokkuus ja vakaus.

Heap sort on myös tehokas suurille alkiomäärille, mutta häviää nopeudessa merge sortille ja quicksortille. Heap sortin etu merge sortiin ja quicksortiin verrattuna on O(1) tilavaativuus.

Suurille alkiomäärille quicksort on keskimääräisesti nopein kaikista vertailuun perustuvista lajittelualgoritmeista, mutta se on epävakaa, vie O(log n) ylimääräistä muistia ja pahimmassa tapauksessa $O(n^2)$ aikaa. Myös quicksortia voidaan optimoida käyttämällä insertion sortia pienillä alkiomäärillä. Quicksortin mediaani pivot-alkiota käyttävä toteutus on keskimääräisesti tehokkain versio quicksortista. Mediaani-quicksortin pahimman tapauksen tapahtuminen on hyvin epätodennäköistä, mutta halutessaan tätä heikkoutta on mahdollista hyväksikäyttää laittamalla mediaani-quicksortin lajittelemaan taulukon, joka on suunniteltu laukaisemaan pahimman tapauksen. Mahdollisten palvelunestohyökkäysten takia mediaani-quicksortin käyttö palvelussa on turvallisuusriski. Tämä voisi olla syy vaihtaa satunnaista pivot-alkiota käyttävään quicksortiin, mutta toisaalta kannattaa ennemmin vaihtaa introsortiin, joka vaihtaa heap sortiin, jos mediaani-quicksortin nopeus alkaa lähetä pahinta tapausta.

Selection sort on aikavaativuudeltaan aina $O(n^2)$, joten se on kilpailukykyinen vain pienillä alkiomäärillä. Selection sort tekee aina minimimäärän alkioiden paikkojen vaihtoja, joten se on parhaimmillaan, kun lajiteltavat alkiot ovat niin suuria, että niiden siirtäminen muistipaikaista toiseen vie paljon aikaa verrattuna alkioiden vertailuun.

Shellsort on suurilla alkiomäärillä hitaampi kuin O(n log n)-lajittelualgoritmit heap sort, merge sort ja quicksort. Shellsort kilpailee pääasiassa heap sortin kanssa, koska molemmat algoritmit ovat paikallaan toimivia. Shellsort voi olla heap sortia tehokkaampi suhteellisen pienillä alkiomäärillä, kuten luvun 4 OpenDSA:n taulukosta näkyy. Shellsort

on myös tehokkaampi lajittelemaan lajiteltuja ja käänteisesti lajiteltuja taulukoita verrattuna heap sortiin.

Introsortilla on kaikki quicksortin hyvät ja huonot puolet lukuun ottamatta $O(n^2)$ aikavaativuuden riskiä, mikä käytännössä tekee introsortista optimaalisimman version quicksortista. Mahdollinen huono puoli on, että introsort vaatii noin kaksinkertaisesti enemmän koodirivejä kuin quicksort [5].

Timsortilla on myös merge sortin aika- ja tilavaativuudet sekä vakaus, mutta timsort ei kuitenkaan aina ole hyvää merge sort toteutusta tehokkaampi, kuten luvussa 4 huomattiin. Timsort on kuitenkin merge sortia huomattavasti nopeampi varsinkin osittain ja täysin järjestetyllä datalla, mikä tekee timsortista yleiskäyttöisesti paremman lajittelualgoritmin.

Bucket sort toimii parhaiten, kun taulukon alkiot noudattavat tasajakaumaa. Luvun 4 hash sortin tuloksista voi olettaa, että bucket sort lajittelisi suuren määrän tasajakaumassa olevia alkioita nopeammin kuin quicksort.

Counting sort osoittautui empiirisissä tutkimuksissa varteenotettavaksi lajittelualgoritmiksi. Counting sort on parhaimmillaan, kun alkiot voivat saada suhteellisen vähän eri arvoja ja kun alkioiden maksimiarvo voidaan syöttää algoritmille, jotta counting sortin ei tarvitse itse sitä selvittää.

Radix sort on parhaillaan, kun se lajittelee suurta määrää samanpituisia lyhyitä alkioita, jos alkiot eivät ole samanpituisia, radix sort joutuu numeroiden tapauksessa lisäämään alkioiden eteen nollia.

Kaikki kolme vertailuun perustumatonta lajittelualgoritmia ovat vakaita ja vievät vähintään O(n) määrän lisämuistia. Luvun 4 tutkimusten perusteella ainakin radix sortin ja counting sortin suoritusnopeus kulkevat käsi kädessä. Taulukkoon 9 on koottu käsiteltyjen lajittelualgoritmien edut ja haitat.

Taulukko 9: *Koonti lajittelualgoritmien eduista ja haitoista*

Lajittelualgoritmi	**Edut ja haitat**
Insertion sort	+ Paras pienille taulukoille. + Ei vie lisämuistia. - Hidas suurille taulukoille.

Merge sort	+ Tehokas suurille taulukoille. - Vie O(n) lisämuistia
Heap sort	+ Tehokas suurille taulukoille. + Ei vie lisätilaa.
Quicksort	+ Keskimääräisesti nopein suurille taulukoille. - Mahdollisuus $O(n^2)$ aikavaativuuteen. - Vie O(log n) lisämuistia.
Selection sort	+ Tekee minimimäärän paikkojen vaihtoja. - Hidas suurille taulukoille.
Shellsort	+ Tehokas pienille ja keskisuurille taulukoille. + Ei vie lisämuistia. - Hidas suurille taulukoille.
Introsort	+ Keskimäärin quicksortin nopeus. + Vähintään heap sortin nopeus. - Vie O(log n) lisämuistia.
Timsort	+ Vähintään lähes merge sortin tehokkuus. + Parempi ennalta lajitellulle datalle. - Vie O(n) lisämuistia.
Bucket sort	+ Tehokas alkioille, jotka noudattavat tasajakaumaa. - Vie O(n) lisämuistia.
Counting sort	+ Tehokas, jos alkiot saavat rajatun määrän eri arvoja. - Vie O(n + k) lisämuistia
Radix sort	+ Tehokas, jos alkioiden pituus on lyhyt. - Vie O(n + d) lisämuistia

6. YHTEENVETO

Tässä työssä tutkittiin erilaisia lajittelualgoritmeja ja niiden etuja ja haittoja relevantin kirjallisuuden ja empiiristen tutkimusten avulla. Tutkimukset koottiin ja analysoitiin, minkä jälkeen lopuksi vastataan siihen, että mikä lajittelualgoritmi on tehokkain vaihtoehto mihinkin tilanteeseen.

Lajittelualgoritmien vertailu ei ole triviaalia, koska todella moni asia vaikuttaa algoritmien suorituskykyyn. Vaikuttavia asioita ovat esimerkiksi käytettävä laitteisto ja ohjelmointikieli, algoritmin monet erilaiset toteutustavat ja lajiteltavan datan tietotyyppi. Taulukkoon 10 on kuitenkin onnistuttu kokoamaan yhteenveto tilanteista ja niihin sopivista lajittelualgoritmeista. Tilanteisiin suositeltavat lajittelualgoritmit on valittu siten, että ne ovat todennäköisimmin optimaalisin valinta annetuilla tiedoilla. Esimerkiksi radix sort on tehokas algoritmi suurilla määrillä alkioita oikeissa olosuhteissa, mutta suositeltavaksi algoritmiksi valittiin introsort ja quicksort, koska ne eivät vaadi tilanteelta mitään muuta.

Pienellä määrällä alkioita insertion sort on optimaalinen valinta. Alkioiden määrän pienuus on suhteellista, mutta yleisellä tasolla voidaan sanoa, että 16 alkiota on pieni määrä. Timsort ja merge sort ovat oikea valinta, kun tarvitaan vakaata lajittelua. Ne jäävät vertailuun perustuvista lajittelualgoritmeista nopeudessa toiseksi vain epävakaille introsortille ja quicksortille. Bucket sort käyttää hyväksi alkioiden tasajakaumaa, joten se on täydellinen valinta, kun alkiot ovat tasaisesti jakautuneet. Introsortia ja timsortia molempia käytetään nykyään ohjelmointikielten standardikirjastoissa niiden monipuolisen tehokkuuden vuoksi. Introsort sopii epävakaaksi ja timsort taas vakaaksi kirjastolajittelualgoritmiksi. Insertion sort ja timsort suoriutuvat jo lajitellun taulukon lajittelemisesta nopeimmin. Introsort ja quicksort taas eivät hidastu paljoa järjestetyn ja osittain järjestetyn taulukon lajittelemisen välissä, joten ne ovat parhaat osittain järjestetyn taulukon lajittelemiseen. Introsort ja quicksort ovat myös optimaalinen valinta, kun taulukko on käänteisessä järjestyksessä.

Counting sort on optimaalinen valinta, kun samat arvot toistuvat taulukossa monta kertaa ja kun alkiot voivat ylipäätänsä saada suhteellisen vähän eri arvoja. Esimerkiksi alkioiden saadessa arvoja väliltä 0–1000, counting sort saattaa hyvinkin olla optimaalinen valinta. Selection sort on optimaalinen valinta vain silloin kun alkioita on vähän ja niiden siirtely taulukossa on niin raskasta, että insertion sort jää hitaammaksi selection sortin alkioiden paikkojen minimaalisen vaihtelun takia. Kun alkioita on todella paljon ja niissä on vain

vähän lukuja tai kirjaimia, radix sort on optimaalinen valinta. Tilanteissa, joissa ylimääräistä muistia ei ole jaettavaksi, kuten sulautetuissa järjestelmissä, heapsort ja shellsort ovat hyviä valintoja. Suurilla alkiomäärillä heapsort on näistä tehokkaampi, mutta pienillä ja keskisuurilla määrillä shellsort saattaa hyvinkin olla nopeampi. Viimeisenä taulukossa 10 on tilanne, jossa ei haluta ottaa riskejä hitaista suoritusajoista, vaan tarvitaan varmaa O(n log n) suoritusta. Tätä tarjoaa nopeusjärjestyksessä introsort, timsort, merge sort ja heap sort.

Taulukko 10: *Yhteenveto tilanteista ja niihin suositelluista lajittelualgoritmeista*

Tilanne	**Suositeltavat lajittelualgoritmit**
Pieni määrä alkioita.	insertion sort
Suuri määrä alkioita.	introsort, quicksort
Tarvitaan vakaa lajittelu.	timsort, merge sort
Taulukon alkioiden arvot ovat tasaisesti jakautuneet.	bucket sort
Yleispätevä lajittelu standardikirjastoon.	introsort, timsort
Alkiot ovat järjestyksessä.	insertion sort, timsort
Alkiot ovat osittain järjestyksessä.	introsort, quicksort
Alkiot ovat käänteisessä järjestyksessä.	introsort, quicksort
Taulukossa on paljon saman arvoisia alkioita.	counting sort
Alkioiden muistipaikkojen vaihtelu on erittäin raskasta ja alkioita on vähän.	selection sort
Alkiot voivat saada vain suhteellisen pienen määrän eri arvoja.	counting sort
Alkioiden arvot ovat lyhyitä, eli niissä on pieni määrä lukuja tai kirjaimia. Alkioita on paljon.	radix sort
Ylimääräistä muistia ei käytettävissä.	heapsort, shellsort
Varmaa O(n log n) tehokkuutta.	introsort, timsort merge sort, heap sort

LÄHTEET

[1] Sedgewick R and Wayne K. *Algorithms*. 1 ed. S.l: Pearson Education, 2011. (Viitattu 17.3.2021.) Saatavissa: https://learning.oreilly.com/library/view/algorithms-fourth-edition/9780132762564/ch02.html.

[2] Heineman GT, Pollice G and Selkow S. *Algorithms in a Nutshell: A Practical Guide*. Sebastopol: O'Reilly Media, Incorporated, 2016. (Viitattu 17.3.2021.) Saatavissa: https://learning.oreilly.com/library/view/algorithms-in-a/9781491912973/ch04.html#sorting-algorithms.

[3] Stein C, Leiserson CE, Cormen TH, et al. *Introduction to Algorithms, Third Edition*: Cambridge: The MIT Press, 2009. (Viitattu 17.3.2021.) Saatavissa: https://ebookcentral.proquest.com/lib/tampere/reader.action?docID=3339142.

[4] Knuth DE. *Art of Computer Programming, The: Volume 3: Sorting and Searching*: Addison-Wesley Professional, 1998.

[5] Musser DR. Introspective Sorting and Selection Algorithms. Software, practice & experience 1997; Vol.27, No.8: 983-993.

[6] Peters T. [Python-Dev] Sorting, https://mail.python.org/pipermail/python-dev/2002-July/026837.html (2002, Viitattu 24.4.2021.).

[7] Peters T. Timsort description, https://svn.python.org/projects/python/trunk/Objects/listsort.txt (2002, Viitattu 24.4.2021.).

[8] Anonymous OpenDSA Data Structures and Algorithms Modules Collection, Chapter 13 Sorting, https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/SortingEmpirical.html (-, Viitattu 24.4.2021.).

[9] Hulín M. *Performance Analysis of Sorting Algorithms*, Bachelor's thesis, Masaryk University, Faculty of Informatics, Brno, 2018, Saatavissa: https://is.muni.cz/th/gp4gz/?lang=en.

[10] Faujdar N and Ghrera SP. Analysis and Testing of Sorting Algorithms on a Standard Dataset. In: *2015 Fifth International Conference on Communication Systems and Network Technologies*, pp.962-967: IEEE, DOI:10.1109/CSNT.2015.98.

[11] Fenyi A, Fosu M and Appiah B. Comparative Analysis of Comparison and Non Comparison based Sorting Algorithms. International Journal of Computer Applications 2020; Vol.175, No.28, Saatavissa: https://www.ijcaonline.org/archives/volume175/number28/fenyi-2020-ijca-920813.pdf.

[12] Rao D and Ramesh B. Experimental Based Selection of Best Sorting Algorithm. International Journal of Modern Engineering Research 2012; Vol.2, No.4, Saatavissa: http://www.ijmer.com/papers/Vol2_Issue4/GD2429082912.pdf.