Testausdokumentti

Koodin testaus:

Ohjelman koodin testaamiseen käytetään JUnit yksikkötestausta. JUnitilla testataan että luokat käyttäytyvät odotetulla tavalla tilanteissa, joista ohjelman edellytetään selviytyvän. Lisäksi ohjelmaa on ajettu debug tilassa varmistaen, että kaikki näyttäisi toimivan juuri niin kuin pitää.

Tulostukseen liittyviä toimintoja tai käyttöliittymää ei ole testattu koneellisesti lainkaan, mutta ohjelmaa on kokeiltu ajaa useilla eri syötteillä ja varmistettu, että tulokset vastaavat odotettuja.

Testit voidaan helposti toistaa ajamalla JUnit testit.

Suorituskyvyn testaus:

Ohjelman suorituskyvyn testaukseen käytetään luodun Sekuntikello luokan antamia aikoja toimintojen suoritusajoille. Sekuntikello luokka käyttää ajan mittaamiseen javan system.nanotime toimintoa jolla voidaan määrittää nanosekunteina kulunut aika jostakin hetkestä. System.nanotimen palauttama aika vaikuttaisi päivittyvän n. 480 ns välein, mutta tämä lienee järjestelmäkohtaista.

Lisäksi mittauksissa havaittiin ilmeisesti sattumanvaraisesti esiintyviä, yleensä ensimmäisiin mittauksiin kohdistuvia, kummallisuuksia. Tämä johtunee jostakin javan ominaisuudesta, kuten muistin varaamisesta, mutta koska en tunne aihetta, en lähde spekuloimaan pitemmälle. Tekemällä mittauksiin tarpeeksi toistoja ja tarpeeksi monta erilaista tietojoukkoa, voidaan virheelliset tulokset huomata ja karsia helposti.

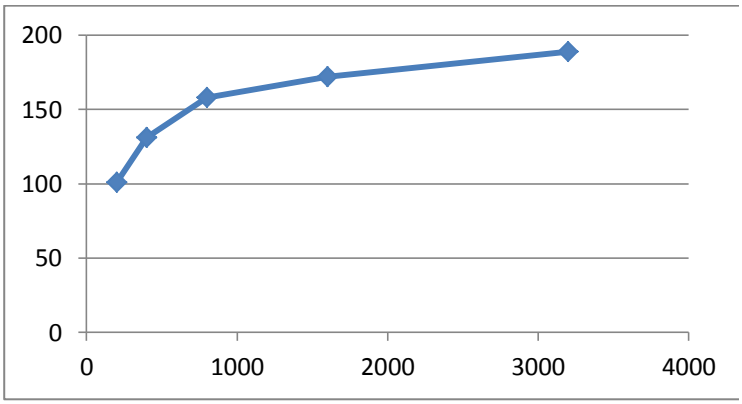
Suorituskykytestauksen tulokset saadaan ajamalla ohjelma. Käyttäjältä ei edellytetä mitään toimia. Ohjelman suoritus voi kestää hitailla järjestelmillä melko kauan.

Suorituskykytestauksen tulokset:

Lisäysajat:

Ilmoitetut ajat ovat keskimääräinen lisäysaika solmua kohti puussa.

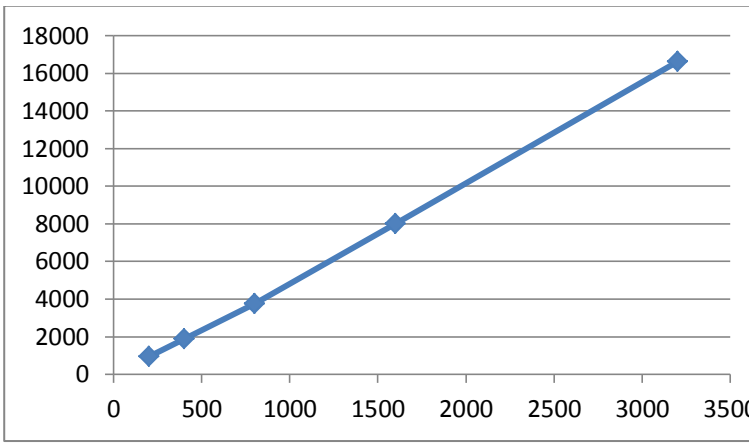
Binäärinen hakupuu:



koko	aika
200	101
400	131
800	158
1600	172
3200	189

Havaitaan että keskimäärin sattumanvaraisia alkioita lisätessä binäärinen hakupuu selviytyy lisäämisestä logaritmisesti. Vaikka alkioiden määrä tuplataan, lisäysaika nousee vain vähän.

Kuitenkin, jos arvot lisätään kasvavassa järjestyksessä puun suorituskyky romahtaa ja aika kasvaa lineaarisesti:



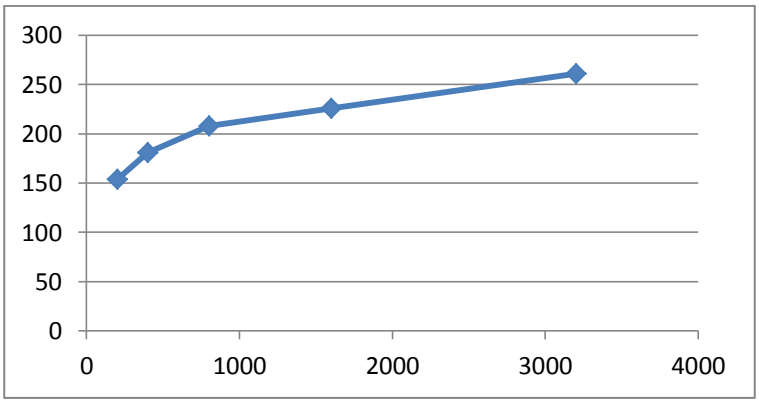
koko	aika
200	942
400	1871
800	3753
1600	7986
3200	16607

Puu muistuttaa tällöin 2 suuntaista linkitettyä listaa ja toimii erittäin hitaasti. Lisättävien arvojen lukumäärän kaksinkertaistaminen melko tarkasti nelinkertaistaa keskimääräisen lisäysajan.

AVL puu:

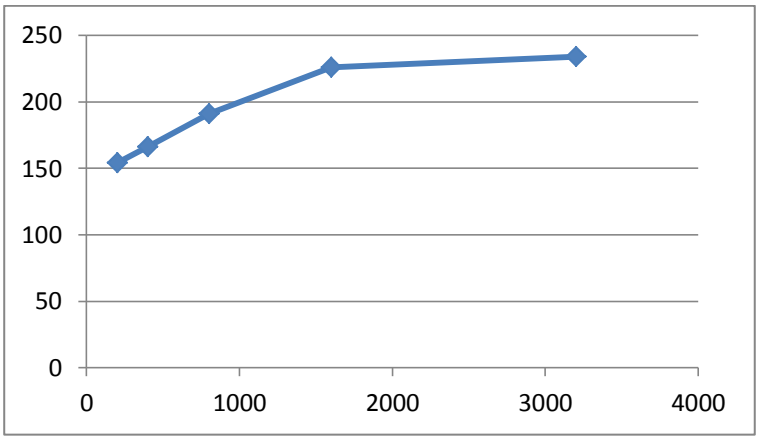
Avl puun tasapainotuksesta johtuen tietojoukolla ei ole niin merkittävästi väliä ja mielenkiintoisesti se selviytyy sattumanvaraisista hitaammin kuin järjestetyistä. Kuitenkin molemmissa tapauksissa suoritusaika kasvaa logaritmisesti:

Sattumanvaraiset alkiot:



koko	aika
200	154
400	181
800	208
1600	226
3200	261

Järjestetyt alkiot:

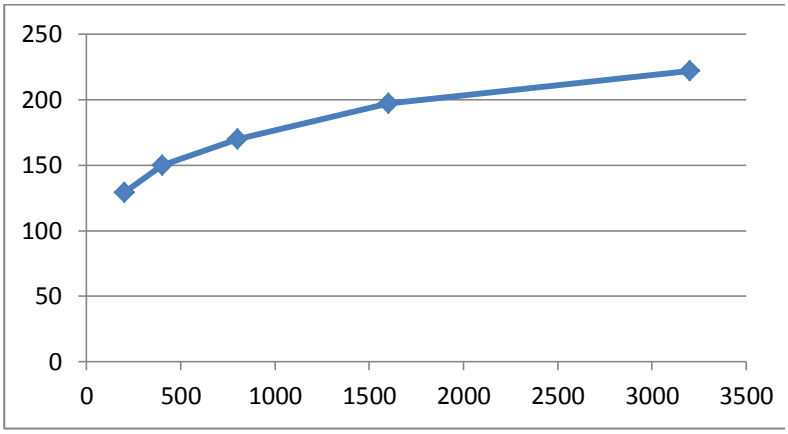


koko	aika
200	154
400	166
800	191
1600	226
3200	234

Punamusta puu:

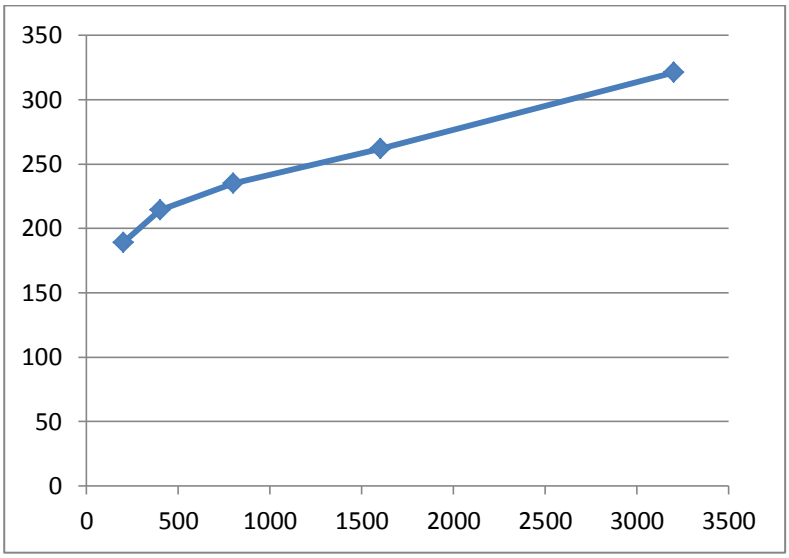
Odotetusti myös punamusta selviää logaritmisesti kaikissa tilanteissa:

sattumanvaraiset:



koko	aika
200	129
400	150
800	170
1600	197
3200	222

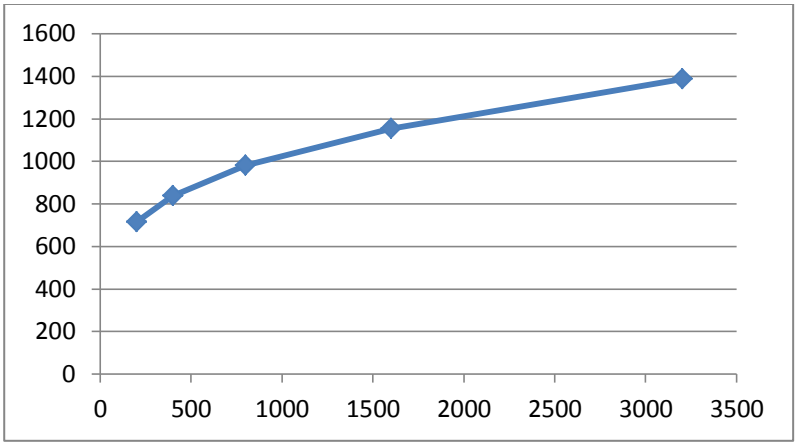
Järjestetyt:



koko	aika
200	189
400	214
800	235
1600	262
3200	321

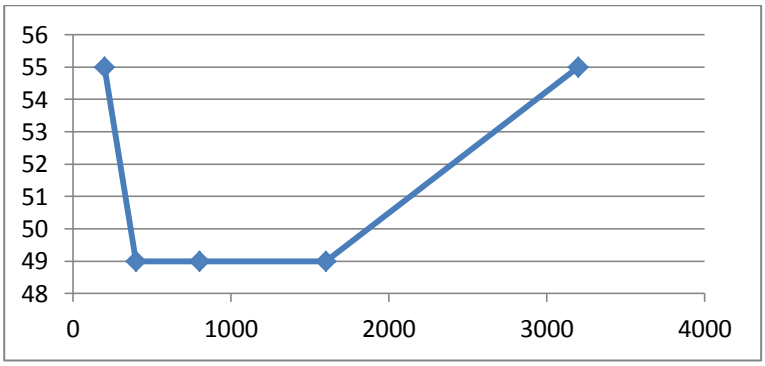
Splay puu:

Splay puu selviää sattumanvaraisista odotetusti aivan kuten tavallinenkin hakupuu:



koko	aika
200	716
400	839
800	981
1600	1154
3200	1388

Mutta järjestetyt ovat vakioaikaisia. Tämä johtuu siitä että splay puu tuo uuden solmun aina juureksi, joten tässä tapauksessa lisäys tehdään aina juuren viereen. (Huomaa kuinka pieniä ajat ovat ennen kuin ihmettelet kaavion muotoa)



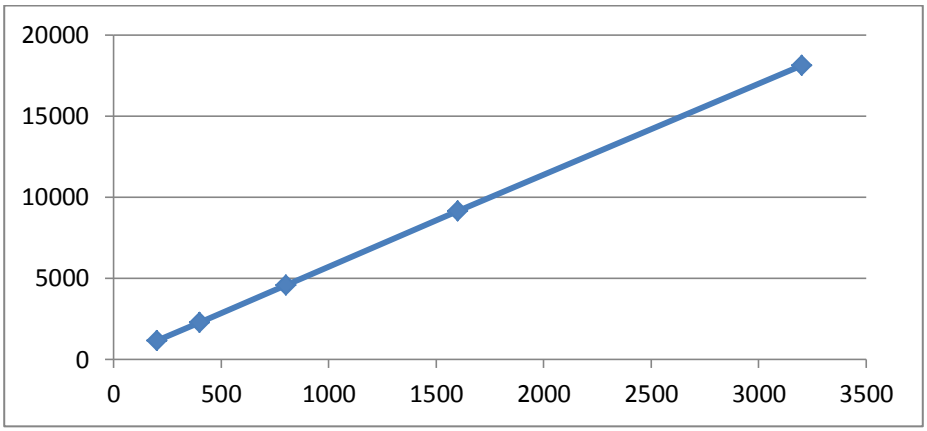
koko	aika
200	55
400	49
800	49
1600	49
3200	55

Hakuajat:

Puusta haetaan kaikki siihen lisätyt alkiot. Lisäys on tehty suuruusjärjestyksessä:

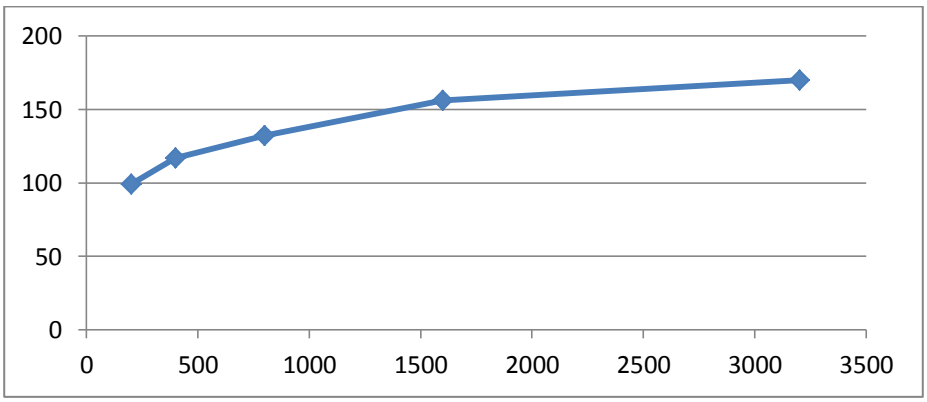
Binäärinen:

Havaitaan että binäärinen suoriutuu hauista erittäin huonosti, koska sen korkeus on sama kuin solmujen määrä lisäysjärjestyksestä johtuen.



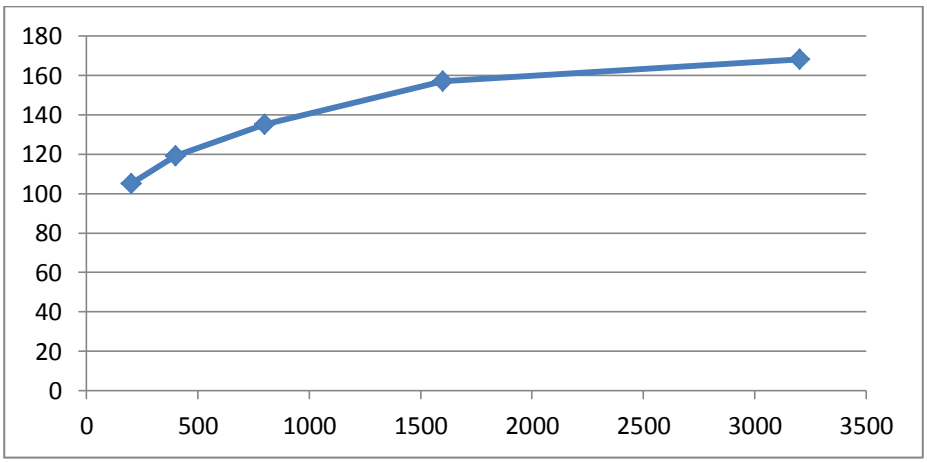
koko	aika
200	1146
400	2287
800	4576
1600	9133
3200	18141

AVL puu suoriutuu hauista odotetusti logaritmisessa ajassa:



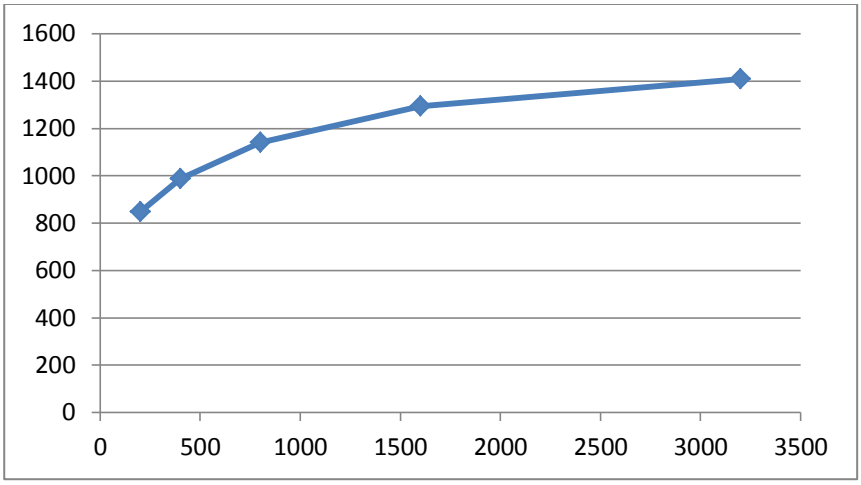
koko	aika
200	99
400	117
800	132
1600	156
3200	170

Punamusta puu selviytyy myös odotetusti logaritmisesti:



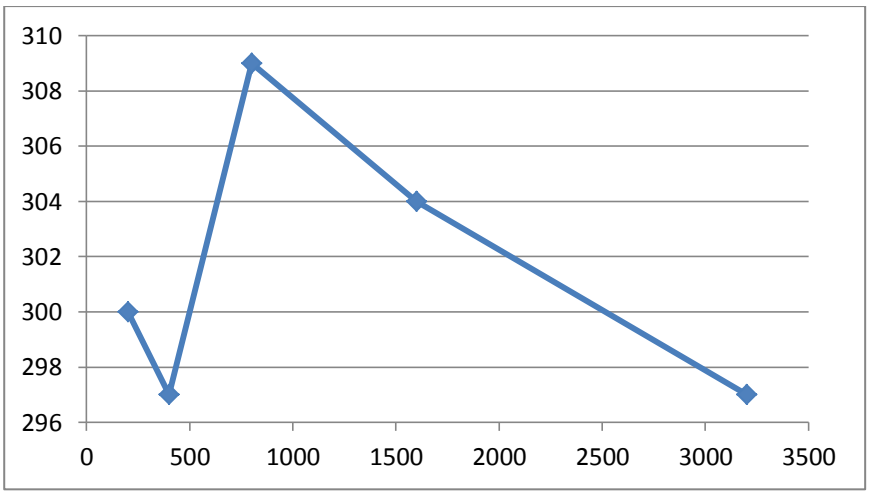
koko	aika
200	105
400	119
800	135
1600	157
3200	168

Splay puu selviytyy kaikista hauista keskimäärin pahimmassakin tapauksessa logaritmisesti. Tämä johtuu siitä että hakujen yhteydessä suoritettavat splay kutsut tasapainottavat puuta:



koko	aika
200	848
400	987
800	1140
1600	1293
3200	1408

Kuitenkin käänteisessä järjestyksessä haun puu hoitaa vakioajassa. Tämä johtuu siitä että viimeisin lisätty alkio on puun juuri ja tässä tapauksessa seuraava haettava arvoon suoraan juuren vasempana lapsena.



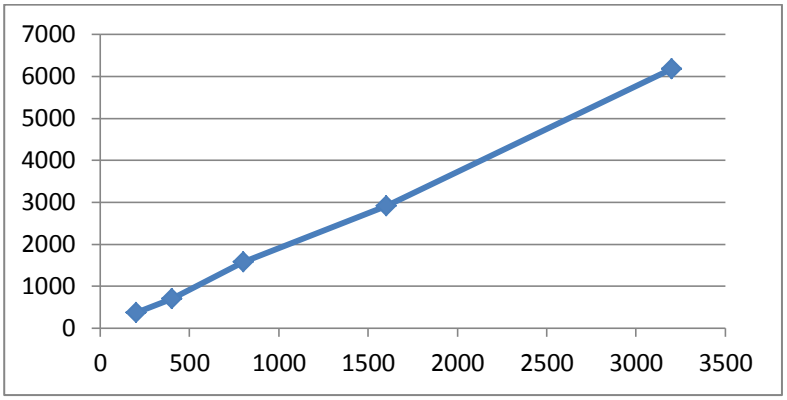
koko	aika
200	300
400	297
800	309
1600	304
3200	297

Poisto:

Binäärinen:

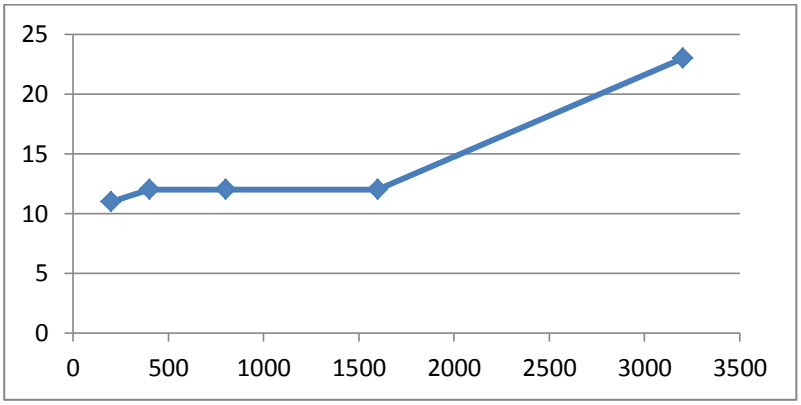
Ajat ovat järjestyksessä rakennetusta puusta sattumanvaraisia hakuja.

Binäärinen poistaa alkiot oletetusti pahimmassa tapauksessa lineaarisessa ajassa:



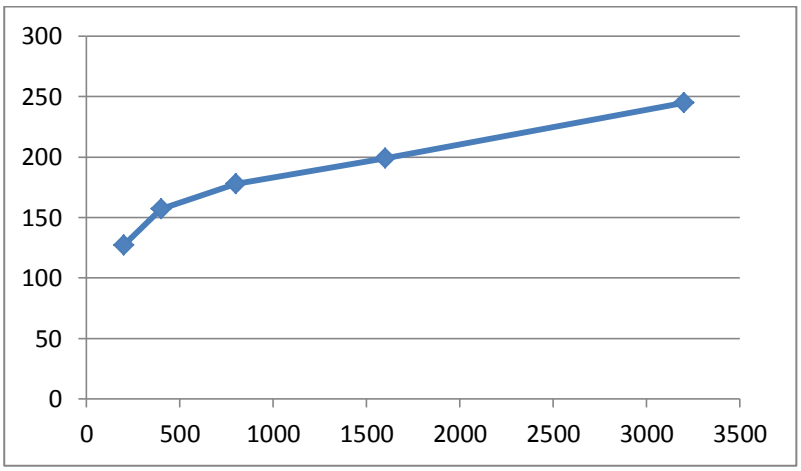
koko	aika
200	370
400	703
800	1579
1600	2910
3200	6174

Kuitenkin, jos alkiot poistetaan käänteisessä järjestyksessä lisäykseen nähden, suoriutuu puu vakioajassa:



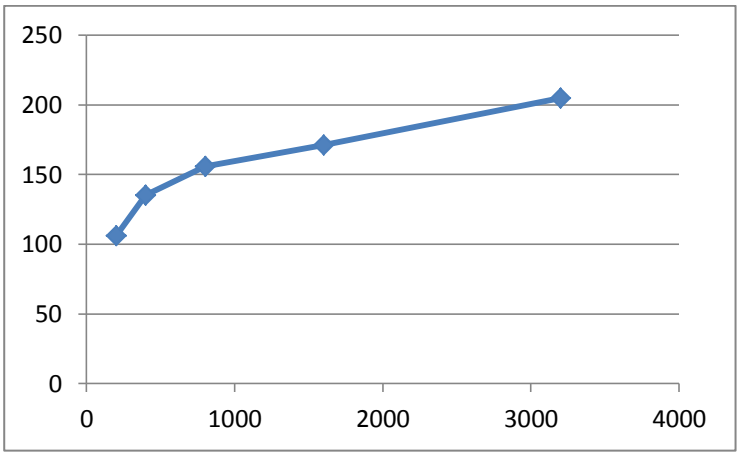
koko	aika
200	11
400	12
800	12
1600	12
3200	23

AVL puu pysyy logaritmisessa ajassa kuten odotettua:



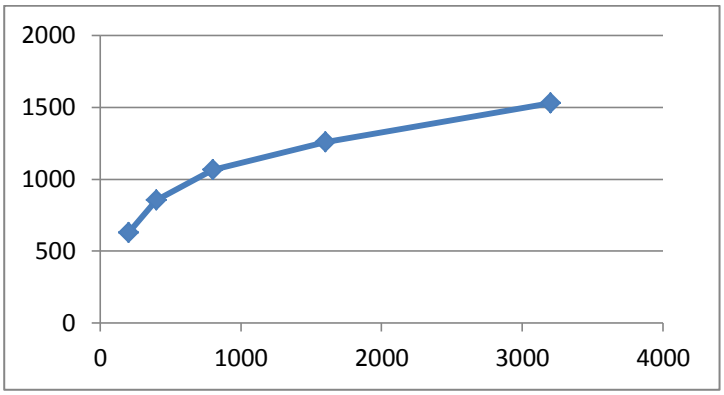
koko	aika
200	127
400	157
800	178
1600	199
3200	245

Kuten myös punamusta puu:



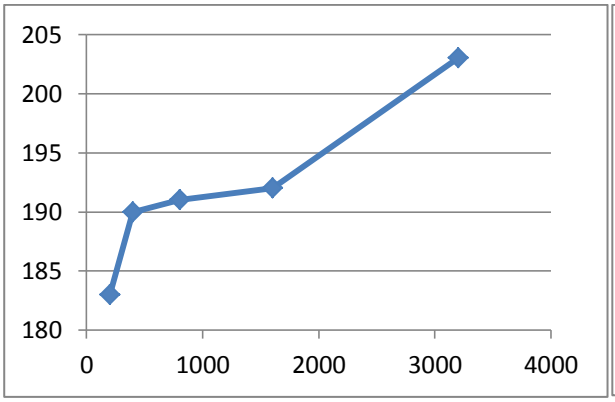
koko	aika
200	106
400	135
800	156
1600	171
3200	205

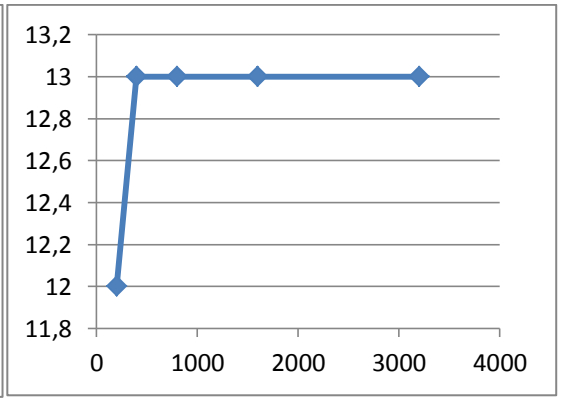
Splay puu yltää jälleen logaritmiseen aikaan splayn aiheuttaman tasapainotuksen ansiosta:



koko	aika
200	629
400	853
800	1065
1600	1257
3200	1528

Ja pääsee vielä parempiin tuloksiin jos alkiot poistetaan järjestyksessä (vasen) tai käänteisessä järjestyksessä(oikea):





koko	aika
200	183
400	190
800	191
1600	192
3200	203

koko	aika
200	12
400	13
800	13
1600	13
3200	13

Yhteenveto:

Havaitaan että muutamaa poikkeustapausta lukuun ottamatta tavallinen binäärihakupuu on erittäin hidas muihin puihin verrattuna. Tämä johtuu siitä että puun korkeus on täysin sen armoilla missä järjestyksessä alkiot lisätään siihen ja kaikkien operaatioiden aika luonnollisesti heijastaa puun korkeutta.

Molemmat tasapainotetut puut selviytyvät kaikissa eri tilanteissa erittäin tasaisesti. Tämä johtuu siitä että niiden tasapainoehdot pitävät korkeuden aina tietyn rajan alapuolella. Jos korkeudelle voidaan määrittää solmujen määrään logaritminen yläraja ja kaikkien operaatioiden suoritusaika on suhteessa korkeuteen niin ilman muuta ne selviävät logaritmisessa ajassa.

Splay puu on mielenkiintoinen kummajainen. Splay operaation aiheuttaman tasapainotuksen ansioista se näyttäisi selviytyvän kaikista operaatiojonoista logaritmisessa ajassa. koska jokainen toiminto aiheuttaa splay operaation kutsun, puu tasapainottuu itsestään, vaikkei sillä varsinaista tasapainoehtoa olekaan. Tämä aiheuttaa myös sen että samojen alkioiden toistuva käsittely tulee erittäin nopeaksi, koska ne löytyvät läheltä juurta. Samoin käy monien järjestettyjen joukkojen kanssa.