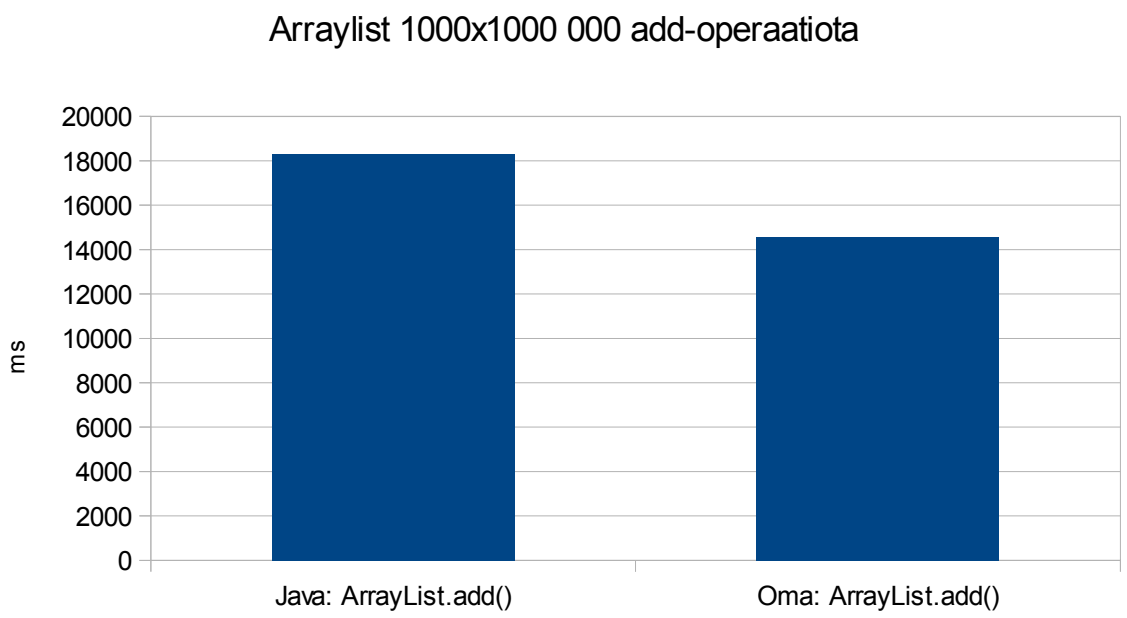
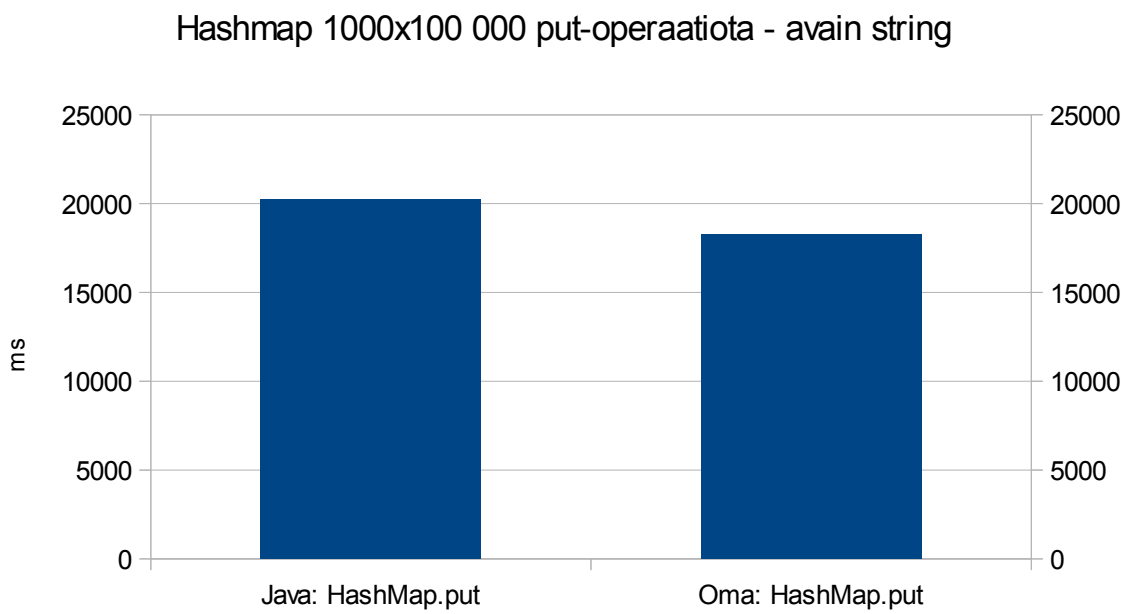
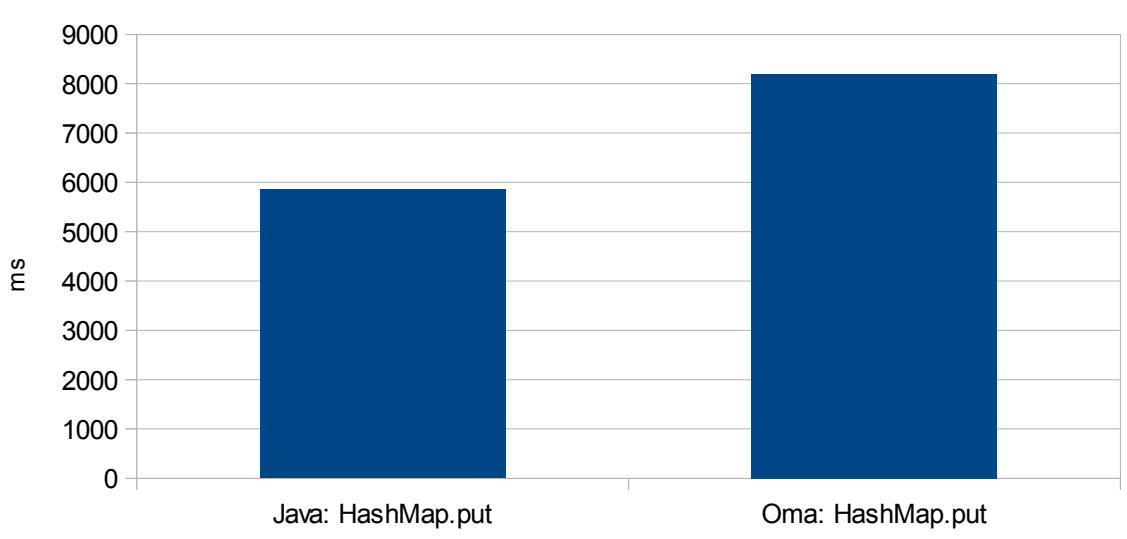
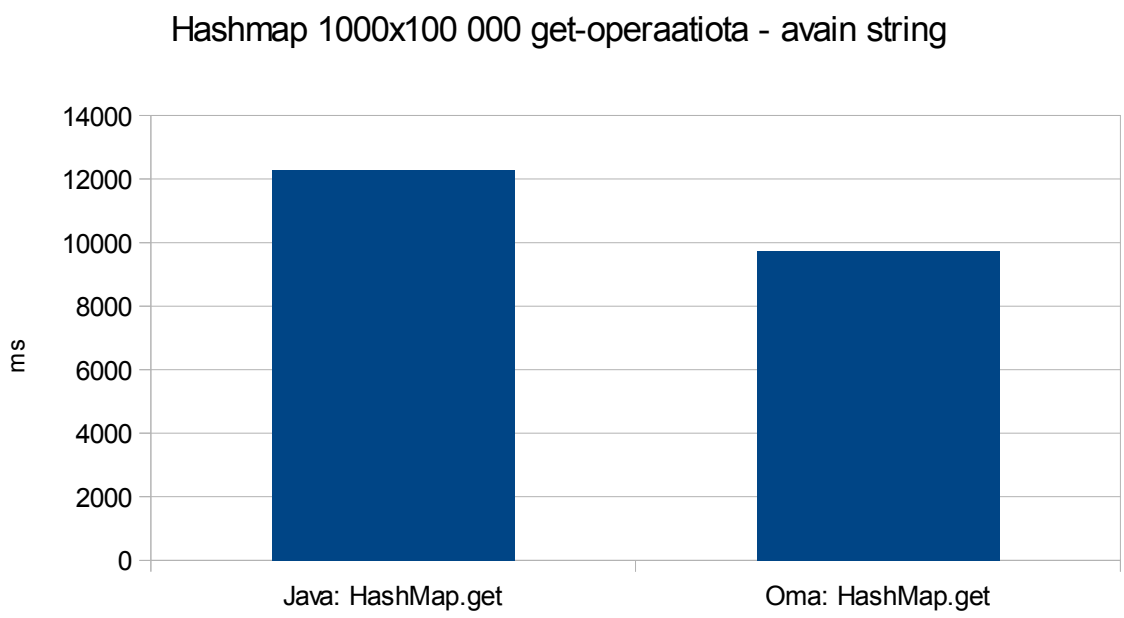
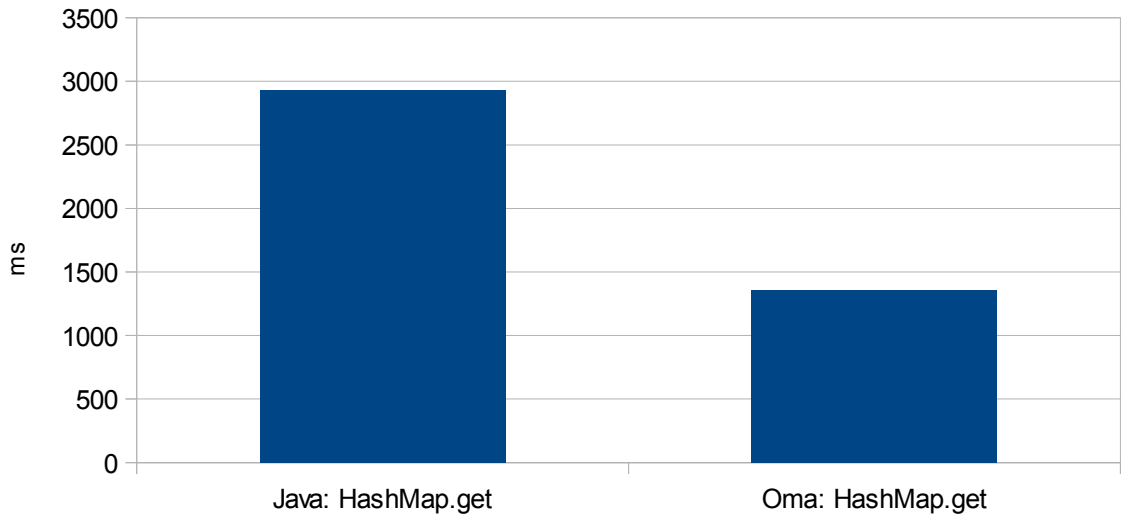
Javan tietorakenteet vs omat tietorakenteet

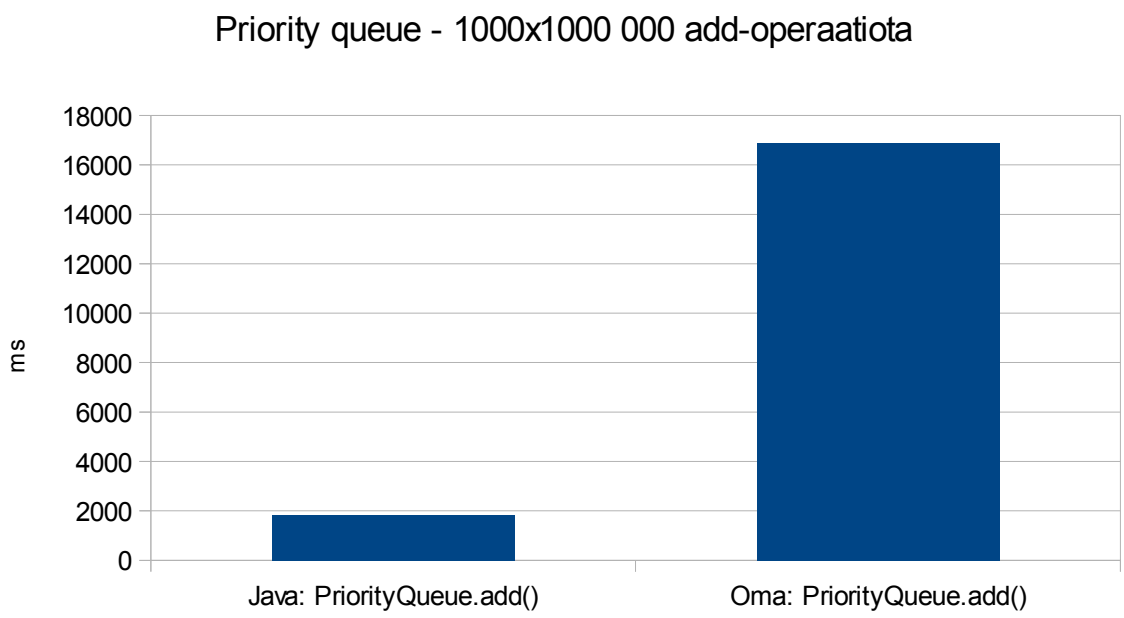


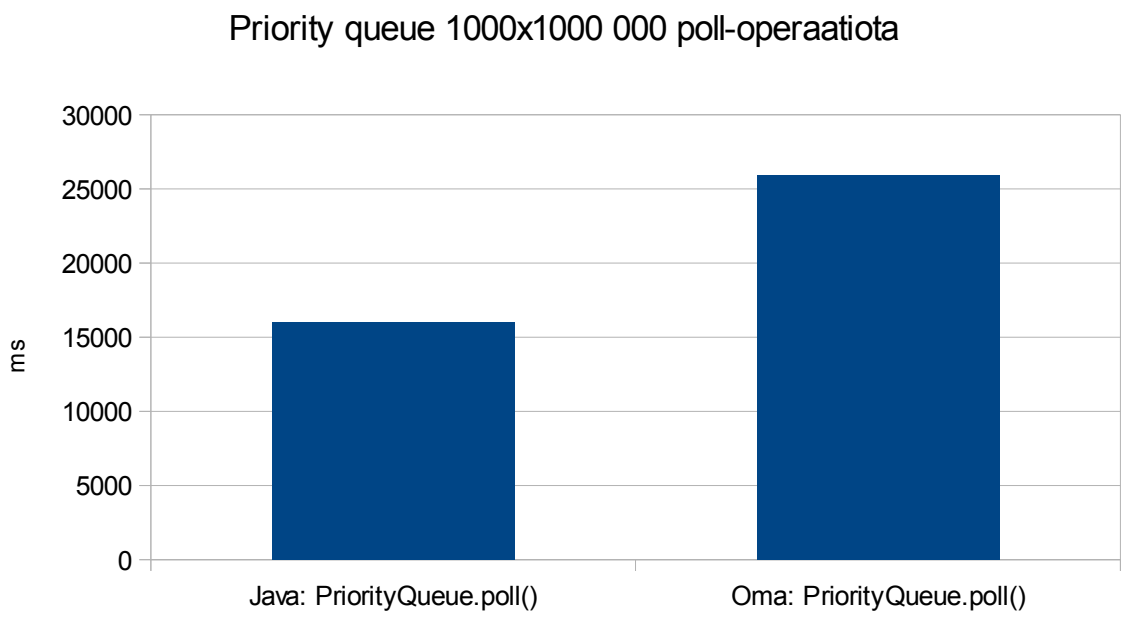





Hashmap 1000x100 000 get-operaatiota - avain string
ms
14000
12000
10000
8000
6000
4000
2000
0
Java: HashMap.get
Oma: HashMap.get

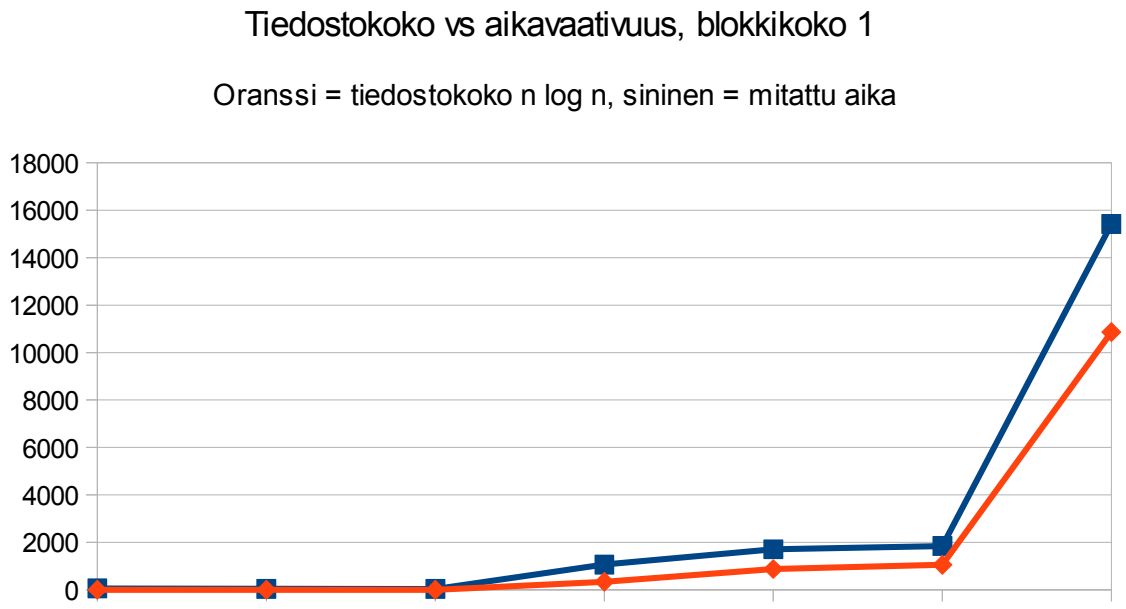

Hashmap 1000x100 000 get-operaatiota - avain Integer
ms
3500
3000
2500
2000
1500
1000
500
0
Java: HashMap.get
Oma: HashMap.get

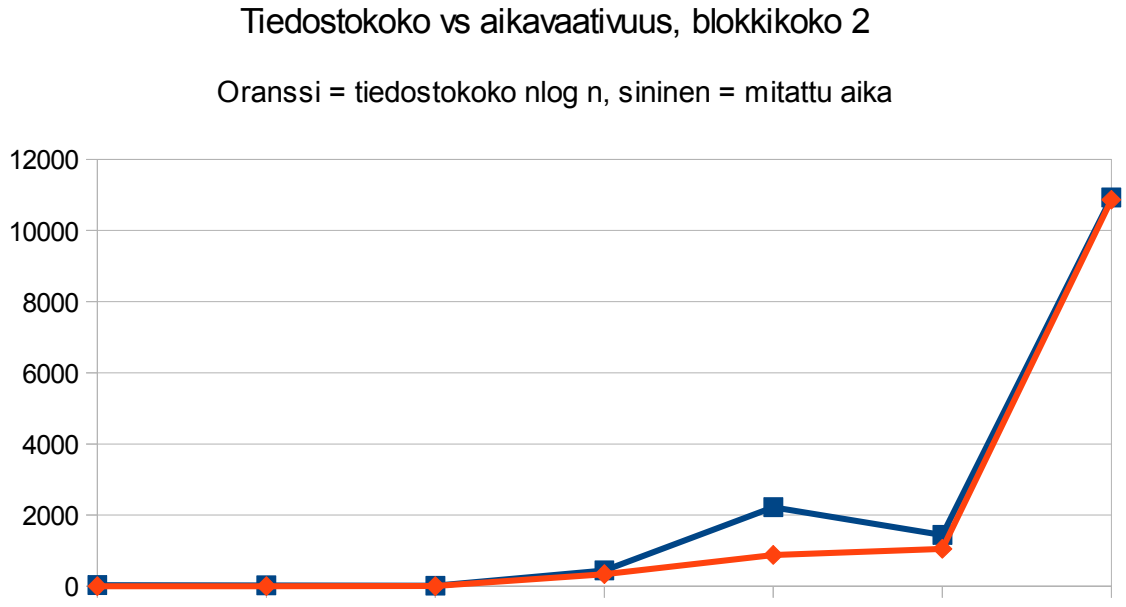

Priority queue - 1000x1000 000 add-operaatiota
ms
18000
16000
14000
12000
10000
8000
6000
4000
2000
0
Java: PriorityQueue.add()
Oma: PriorityQueue.add()

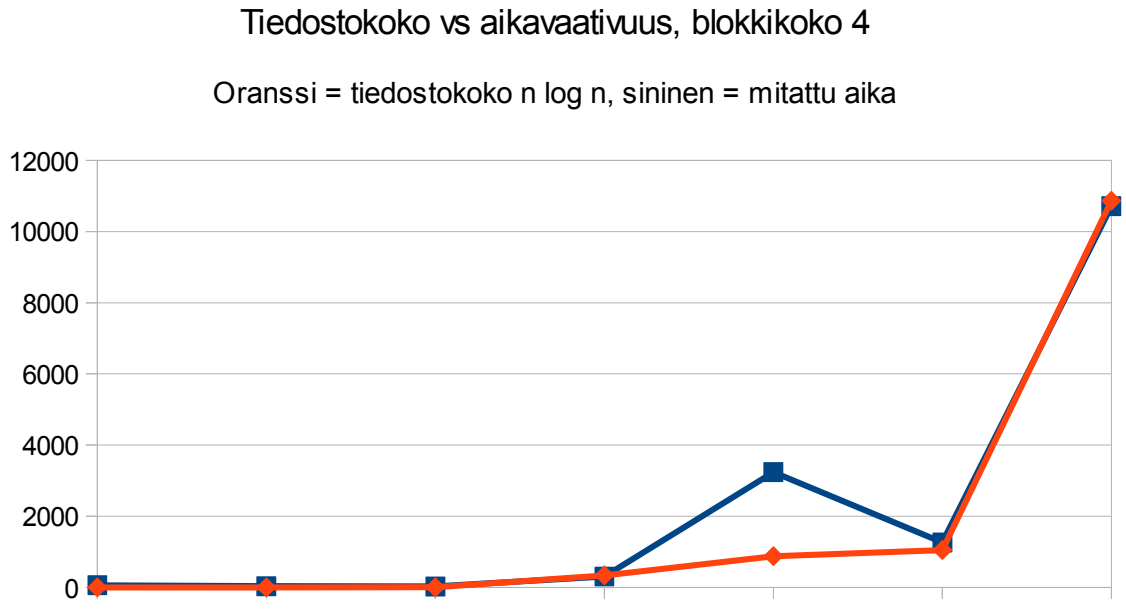

Priority queue 1000x1000 000 poll-operaatiota
ms
30000
25000
20000
15000
10000
5000
0
Java: PriorityQueue.poll()
Oma: PriorityQueue.poll()

Tiedostokoko vs aikavaativuudet

Käytetyt tiedostot:
tekstitiedosto – 1,6kt
tekstitiedosto – 18kt
tekstitiedosto – 74kt
xml-tiedosto – 2945kt
bmp-tiedosto – 6891kt
bmp-tiedosto – 8100kt
rakenteinen tiedosto, tekstiä – 67710kt

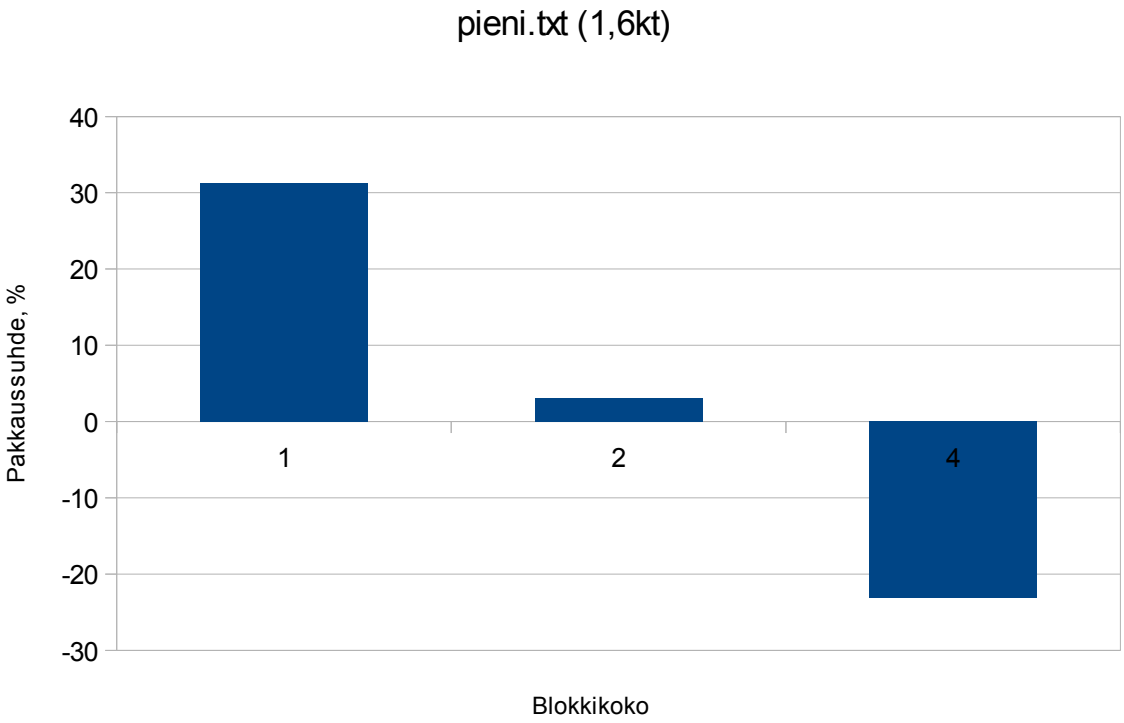


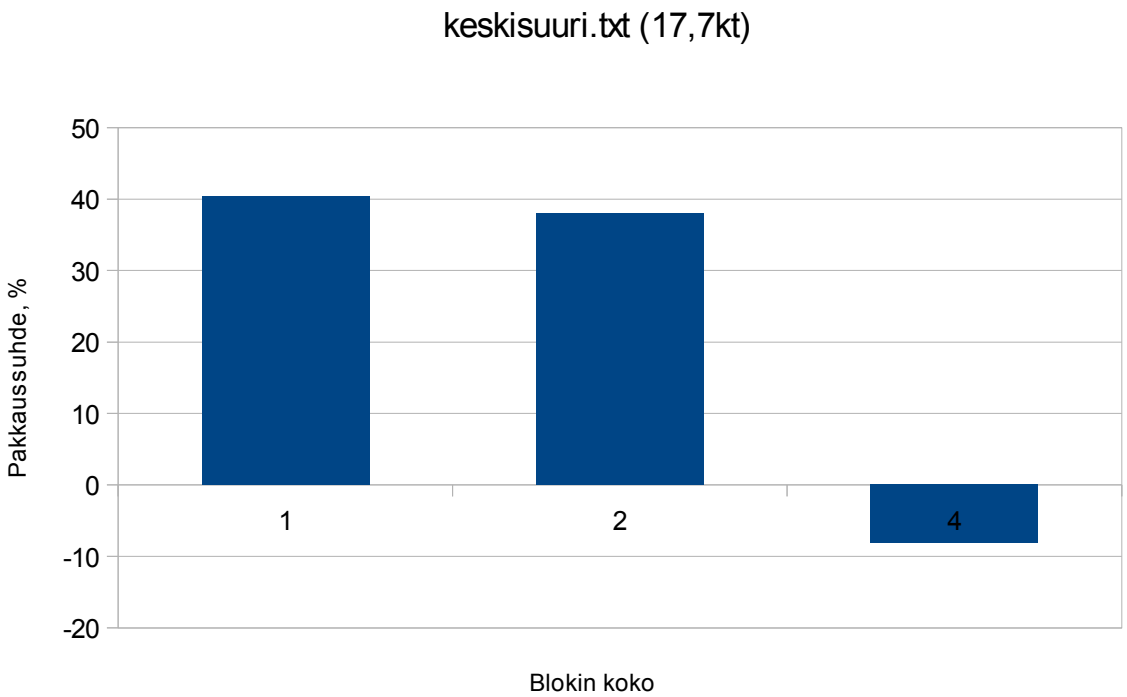


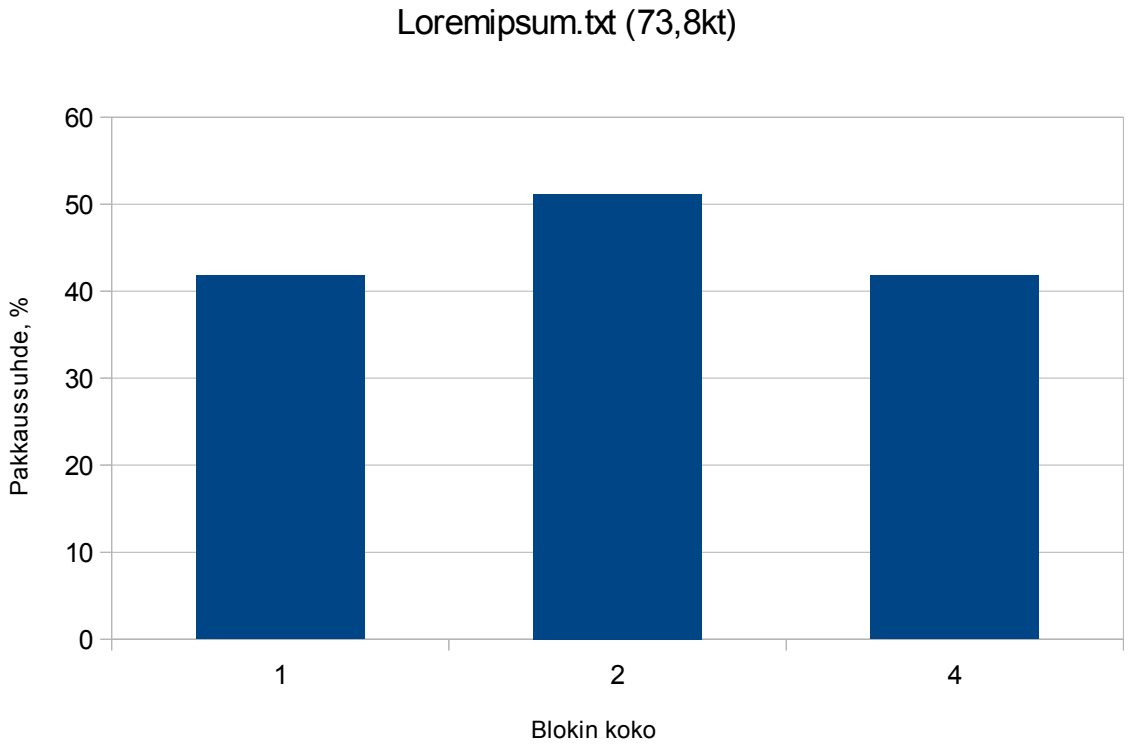


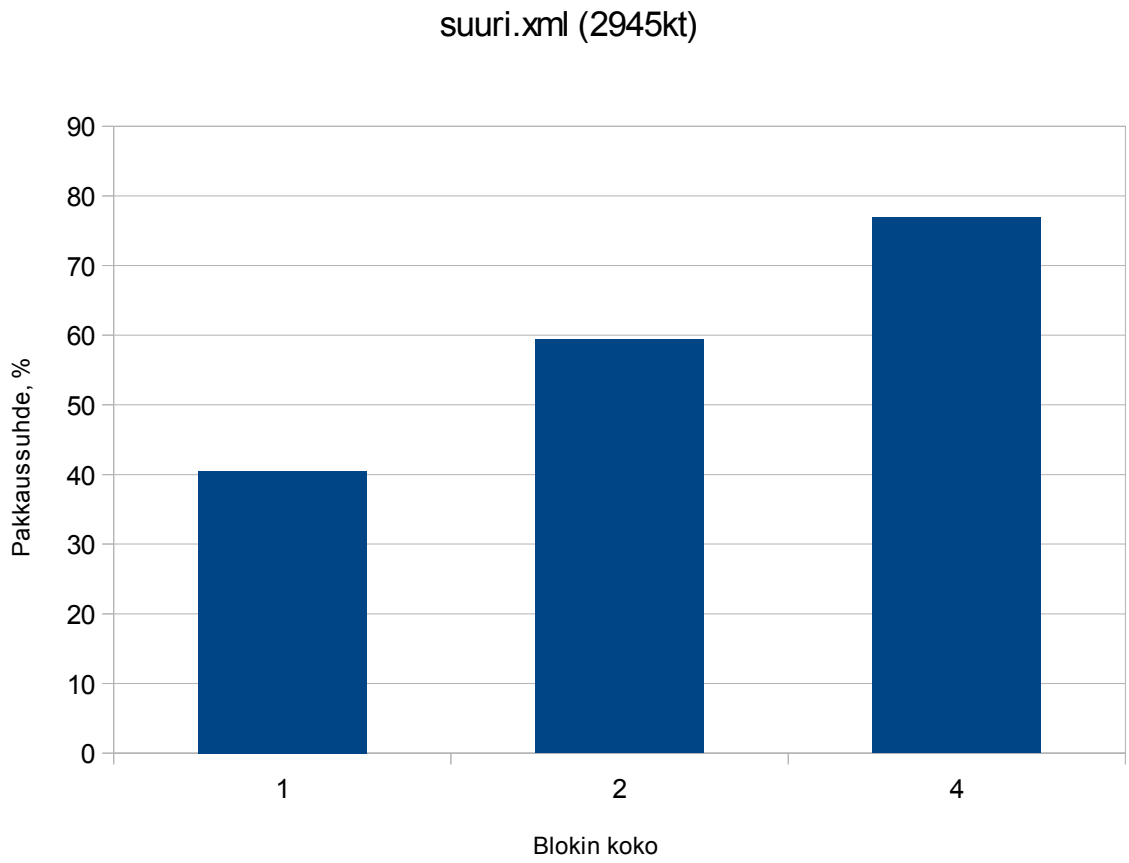
Enimmäkseen laskettu aikavaativuus ja tiedostokoko seuraavat toisiaan – pseudokoodin pohjalta arvioitu että aikavaativuus on ~O(n log n) missä n on tiedoston koko tavuissa. Poikkeuksena yksi bmp-tiedosto jossa luetut blokit pitkälti uniikkeja -> vaativampi kuin suurempi tiedosto koska O (n log n) tarkalleen ottaen verrannollinen uniikkien blokkien määrään.

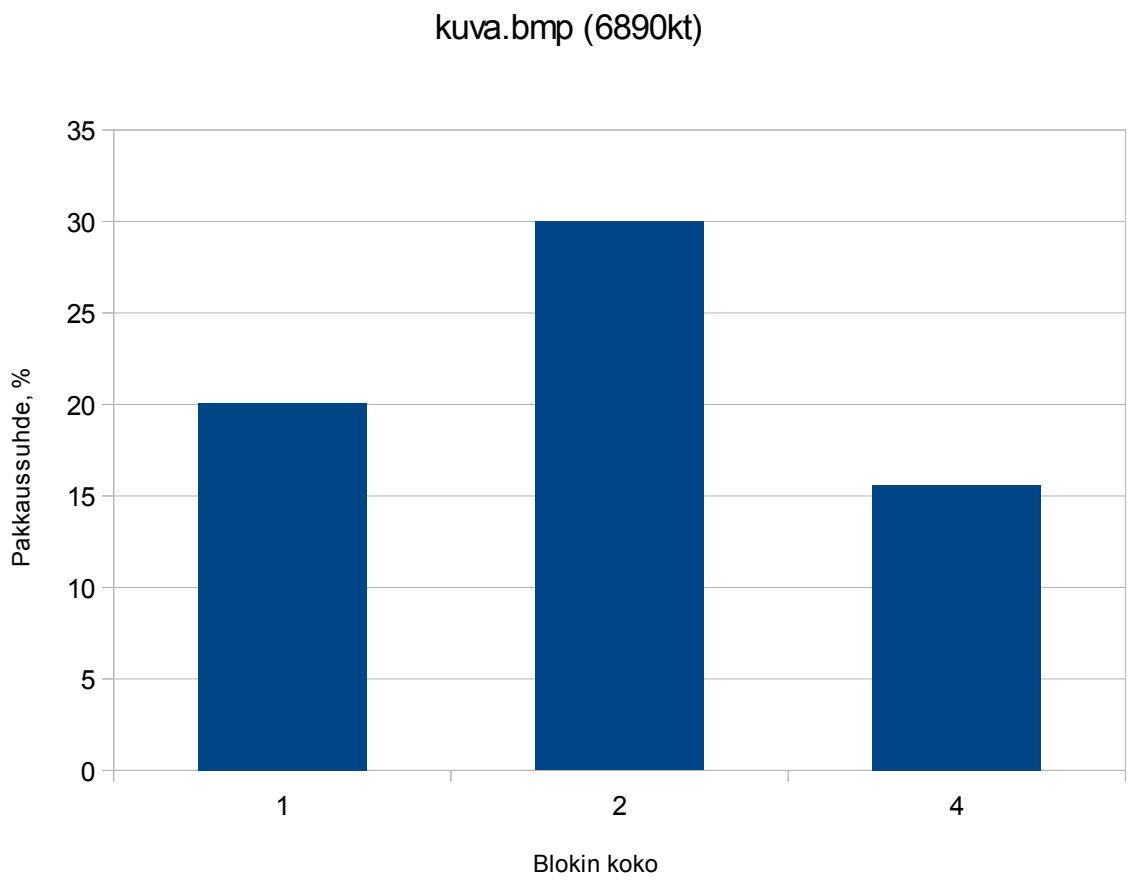
Pakkausprosentit vs blokkikoko

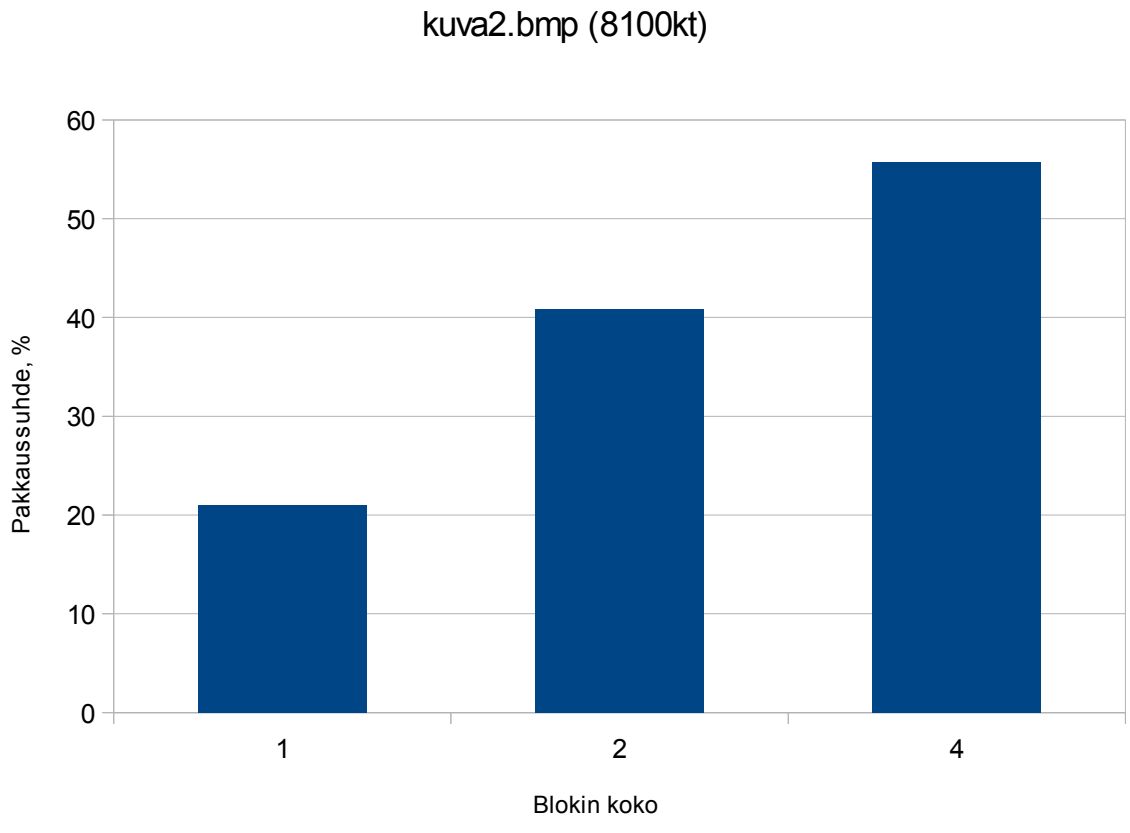


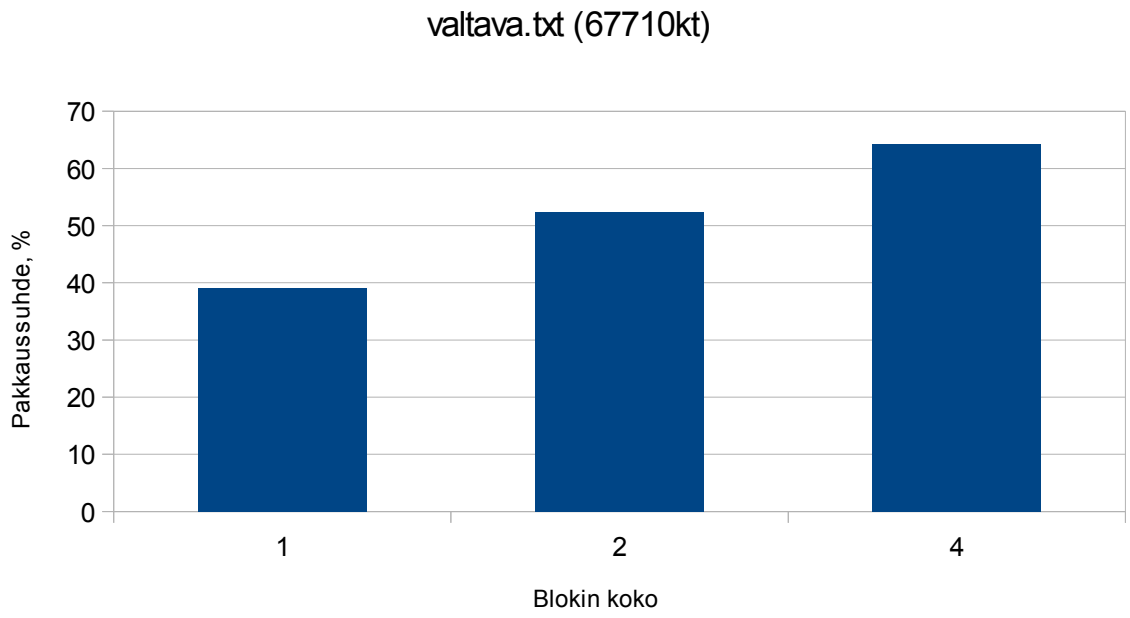



Loremipsum.txt (73,8kt)
Pakkaussuhde, %
60
50
40
30
20
10
0
1
2
4
Blokin koko


suuri.xml (2945kt)
Pakkaussuhde, %
90
80
70
60
50
40
30
20
10
0
1
2
4
Blokin koko


kuva.bmp (6890kt)
35
30
25
20
15
10
5
0
Pakkaussuhde, %
1
2
4
Blokin koko


kuva2.bmp (8100kt)
60
50
40
30
20
10
0
Pakkaussuhde, %
1
2
4
Blokin koko



Valmiiksi pakatun tiedon pakkaus