Rapport Oblig 1 INF2440

Brukernavn: eeaarset

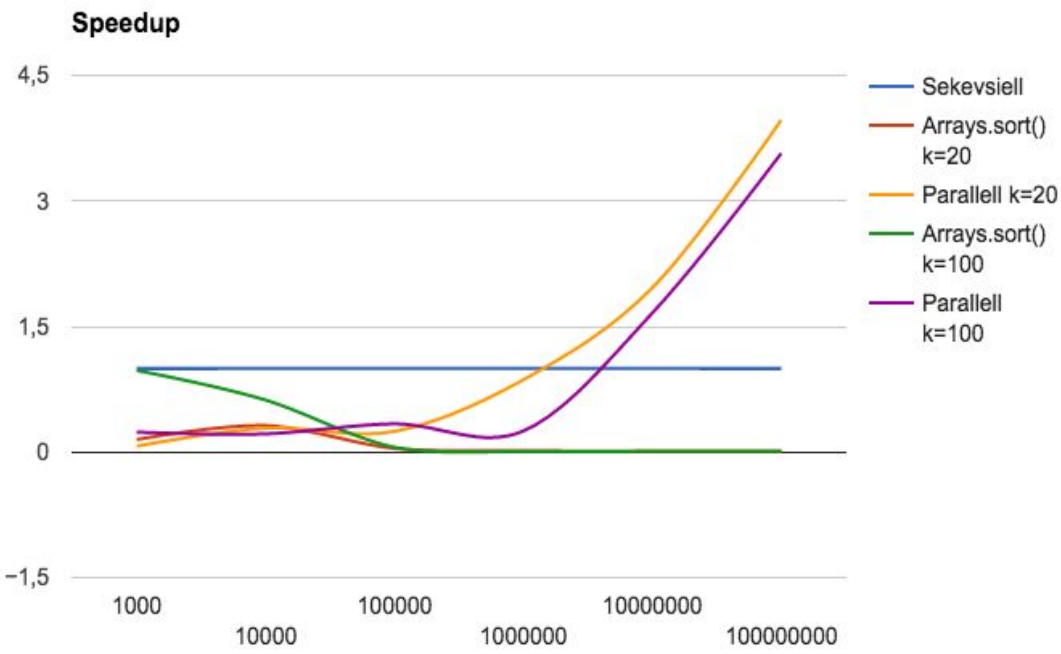
Kjøretider.

k = 20

n	Sekvensiell	Arrays.sort()	Parallell
1000	0,06	0,40	0,86
10000	0,30	0,95	1,05
100000	0,56	13,05	2,27
1000000	1,14	77,61	1,32
10000000	7,61	968,22	3,88
100000000	109,18	10762,16	27,53

k=100

	Sekvensiell	Arrays.sort()	Parallell
1000	0,39	0,40	1,64
10000	0,59	0,95	2,72
100000	0,73	13,11	2,16
1000000	0,58	82,72	2,31
10000000	7,92	970,42	4,77
100000000	116,58	10587,45	32,67



Alle tider er i millisekund, medianen på 7 kall på usorterte arrayer, og beregnet fra et Random objekt initialisert med:

```
new Random(500);
```

Prosessor: Intel(R) Core(™) i7-2720QM CPU @ 2.20GHz (4 fysiske kjerner, som hver blir til 2 virtuelle)

Kommentarer:
Parallelliseringen av A2 gir speedup > 1 fra et sted mellom 1 000 000 og 10 000 000. Den sekvensielle delen av den parallelle implementasjonen blir mindre betydlig, detso større n blir, hvis k forblir konstant, og det er derfor at en høyere n favoriserer en parallell løsning. Samtidig kan vi se at høyere k forskyver speedupen mot en høyere n. Målingene virker noe tilfeldige, ettersom man medianen kan variere +-50%, ved flere kjøringer. Men det er likevel ikke tvil om at den parallelle løsningen vil lønne seg når n blir tilstrekkelig stor. Begge implementasjonene er også mange ganger så rask som java egen Arrays.sort() som sorterer hele arrayet, og da spesielt ved høy n.