K	N	A1 (T)	A2 (T)	A3 (T)	S(2-3)
20	1000	0,559187	0,089517	4,802981	0,01863780015
20	10000	1,127757	0,278018	3,710304	0,07493132638
20	100000	13,507394	0,628975	2,894337	0,2173122895
20	1000000	119,327039	4,074283	3,042236	1,339239625
20	10000000	1382,717052	18,426075	4,826526	3,817668236
20	100000000	15927,50588	386,898511	26,449238	14,62796437
100	1000	0,915734	0,615896	4,505147	0,1367094126
100	10000	2,229502	0,159111	4,189056	0,03798254308
100	100000	14,912427	0,743781	2,556927	0,2908886331
100	1000000	118,724893	4,29421	4,077745	1,053084487
100	10000000	1365,292787	17,486422	4,705236	3,716375119
100	100000000	15750,57303	391,350786	23,26164	16,82386908

Tabell 1: Resultater for både A1, A2 og A3. I tillegg speed up fra A2 til A3.

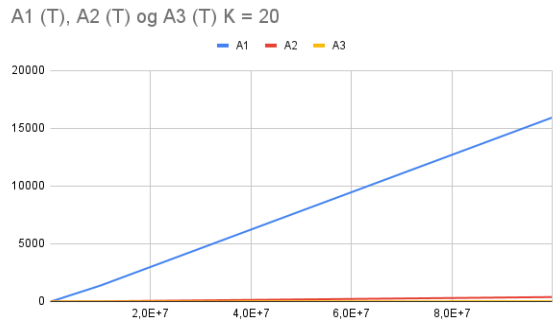

Diagram 1: Resultat fra K = 20.

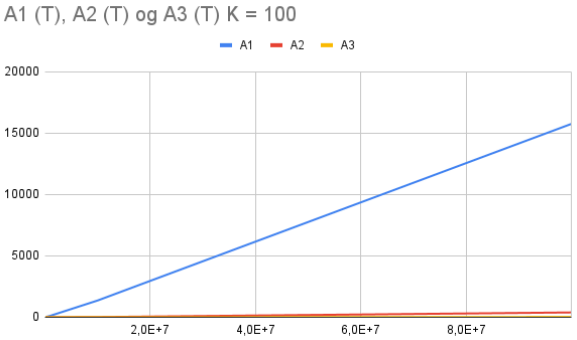

Diagram 2: Resultat fra K = 100.

Ut ifra resultatene så har valg av algoritme mye å si om hvor mye tid det tar å finne de K største tallene i en liste av N elementer. På mindre tall generelt så vil A1 og A2 slå ut raskest, men for større tall så vil A3 ha best tid, minst er best. Dette har nok med at det tar tid å synkronisere trådene i A3, men den tiden er irrelevant når man skal finne K største på lister større enn en million.

Jeg er usikker på hvorfor det er raskere med K = 100 enn K = 20, men kanskje det har noe med cache og hvordan Java håndterer de samme elementene.

Prosessor informasjon:
16 × 13th Gen Intel® Core™ i5-13500H
https://www.intel.com/content/www/us/en/products/sku/232147/intel-core-i513500h-processor-18m-cache-up-to-4-70-ghz/specifications.html