Concurrency
Opdracht 1

Stijn van de Kroon
304774
Milan Langeler
150268

Inleiding
In dit verslag bespreken we de eerste opdracht van het 2e jaars vak Concurrency. Alle opdrachten zijn samen uitgewerkt en ook beschikbaar op GitHub.

De opdracht is gecentreerd rond het sorteren van lijsten in meerdere threads. Om consistente meetgegevens te produceren hebben we alle metingen gedaan op dezelfde laptop. De specificaties hiervan zijn:

Model	Dell XPS 13 Ultrabook
Processor	Intel Core i7 3537U @ 2.0 GHZ
Geheugen	8GB Ram
Opslag	Samsung SSD PM830 mSATA 256GB
Java versie	1.7.0 update 55

Opdracht 1.1:
de broncode van insertionsort komt van http://www.algolist.net/

Voor opdracht 1.1 moesten we een non-threaded applicatie bouwen die insertionsort doet op 25.000, 50.000, 100.000, 200.000, 400.0000 en 800.000 getallen gaat sorteren. We moesten meten hoe lang dit duurde voor elke lijst.
Elke lijst is 5 keer gesorteerd. Dit zijn de gemiddeldes:

25.000	107 milliseconden
50.000	559 milliseconden
100.000	2013 milliseconden
200.000	7886 milliseconden
400.000	32383 milliseconden
800.000	139225 milliseconden

Bij een verdubbeling van het aantal te sorteren getallen gaat de tijd die het kost niet keer twee. Het is dus geen recht evenredig verband. Elke verdubbeling van het aantal getallen kost het grofweg 4 keer zoveel tijd om te sorteren. Hieruit kunnen we een kwadratisch verband afleiden:

$$sort(n) = sort(n-1) * 4$$

Opdracht 1.2
de bron gebruikt voor deze opdracht is http://stackoverflow.com/questions/5958169/how-to-merge-two-sorted-arrays-into-a-sorted-array

Voor deze opdracht moest hetzelfde gebeuren alleen dan met 2 threads die beiden de helft van de lijsten sorteerden met insertionsort en daarna gemerged werden.
Elke lijst is 5 keer gesorteerd. Dit zijn de gemiddeldes:

25.000	38 milliseconden
50.000	145 milliseconden
100.000	587 milliseconden
200.000	2438 milliseconden
400.000	9624 milliseconden
800.000	39053 milliseconden

Ook bij deze opdracht is het verband uit opdracht 1.1 te zien: bij elke verdubbeling van het aantal getallen is de tijd 4 keer zo lang. Wat hier opvalt is dat de gemiddeldes meer dan twee keer zo snel zijn als in opdracht 1.1, terwijl je zou verwachten dat ze twee keer zo snel zouden zijn. Je sorteert immers in twee threads en dus verricht je twee keer zoveel werk in dezelfde tijd.

Wat hier gebeurd is interessant maar niet geheel verassend als je erover na gaat denken. Het sorteren van een twee keer zo kleine lijst is 4 keer zo snel. Omdat je de lijst in twee delen splits en tegelijk bewerkt zou hij dus 4 keer zo snel klaar moeten zijn. Het mergen van twee al gesorteerde lijsten is de extra tijd waardoor het niet precies 4 keer zo snel is. Wat hier ook mee telt is het “context switching” van de JVM, maar dit gaat te ver voor dit vak.

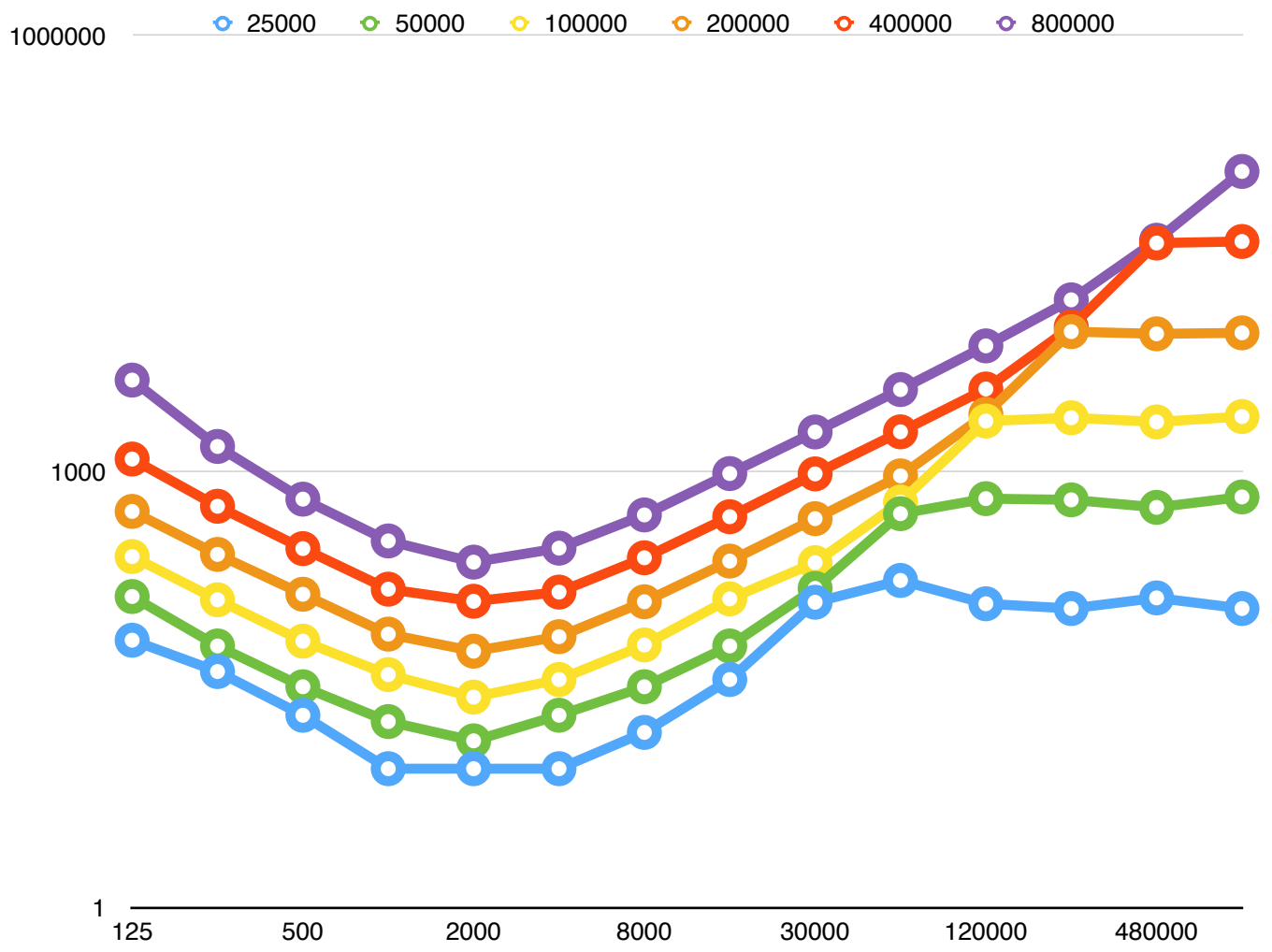
Opdracht 1.3 en 1.4
Opdracht 1.3 neemt het multithreaded sorteren nog een stapje verder. Hierin testen we in welke verhoudingen opsplitsen en sorteren het meest efficiënt zijn. Hiervoor nemen we weer dezelfde grote lijsten als in de vorige opdrachten. Bij het sorteren geven we een “threshold” mee aan elke thread: als het aantal te sorteren getallen lager ligt dan deze waarde dan wordt de lijst gesplitst en worden er twee nieuwe threads aangemaakt. In onderstaande grafiek staan de ruwe meetwaarden. Op de X as zijn de lijst grotes weergeven. Op de Y as staan de thresholds. Alle metingen zijn in milliseconden.

	25000	50000	100000	200000	400000	800000
125	69	138	260	532	1220	4265
	42	63	131	269	577	1489
500	21	33	68	143	295	646
1000	9	19	40	76	155	333
2000	9	14	28	58	128	238
4000	9	21	37	73	148	297
8000	16	33	64	127	255	503
16000	37	63	133	242	489	969
30000	126	157	236	474	964	1874
60000	178	513	617	935	1878	3671
120000	123	651	2228	2482	3705	7337
240000	114	639	2347	9194	9898	15226
480000	135	567	2198	8881	37466	39159
960000	114	669	2393	9021	38392	116595

Het eerste wat opvalt is dat de tijden het laagst zijn bij een threshold van 2000. Interessanter is om deze gegevens in een grafiek te bekijken. Hierdoor worden verbanden gevisualiseerd. Op de volgende pagina is deze grafiek weergeven. We hebben de Y as een logaritmische schaal gegeven om het verband beter te zien. Op de X as zijn de thersholds weergeven. Op de Y as zien we de tijd die het koste om het te sorteren.

De ideale threshold ligt bij 2000, hierna gaan de tijden weer stijgen. Op een gegeven moment rechten de lijnen uit, maar elk net een threshold later dan de vorige. Hierin zie je de zorgvuldig gekozen thresholds: de laatste 6 zijn telkens ieder net iets hoger dan een van de lijst lengtes. Hierdoor wordt er dus effectief met maar 1 thread gesorteerd.

Het verder opsplitsen dan 2000 zorgt ook voor stijgende tijden. In dit geval worden er zoveel threads aangemaakt dat het continu switchen tussen de threads te veel tijd in beslag neemt.



Conclusie
Het sorteren van lijsten in meerdere threads geeft soms andere resultaten dan je in eerste instantie zou verwachten. Het lijkt erop dat als je meer threads gebruikt het steeds sneller gaat, maar uiteindelijk is ook hier een limiet aan. Als we deze conclusie naar een ander abstractie niveau tillen is het met de huidige technologische ontwikkelingen in gedistribueerde systemen noodzakelijk om na te gaan of het wel nuttig is of dat het prima op een machine kan draaien.