Report in3030 Oblig 5
By: Finn Eivind Aasen (finneaa)

For kjøring av programmet:

Kommando for å kompilere programmet:

```
javac –d bin Oblig5.java
```

Kjør programmet slik:

```
java –cp Oblig5 <Tråder> <Tester>
```

Hvor <Tråder> er antall tråder og <Tester> (et positivt oddetall) er antall ganger testen får kjøre. Du trenger ikke å spesifisere noen for de to feltene hvis du ønsker at de skal bli valgt automatisk. I tillegg, for å gjøre det enklere å teste, har jeg laget en makefile som kan kjøre programmet med 8 tråder og 3 tester.

Paralleliseringen
Beskrivelse av algoritmen (relevant for begge Sekv og Para versjoner):
Hentet fra forklaring skrevet i toppen av Oblig5.java
* Algorithm Description:
 * ++ Step 1: Find minx and maxx.
 * ++ Step 2: Divide all the points into 2 lists (those on the left, and those
 * on the right of line maxx->minx). At the same time, find 2 points that
 * are furthest away from both sides of the line.
 * ++ Step 3: Recursion (which consists of 3 sub-steps):
 * -- Step 3A: Check for the base case
 * -- Step 3B: With p3 being the furthest point to the right of line p1->p2,
 * this step will put all the points to the right of the line p1->p3 and
 * the line p3->p2 into 2 separate lists. At the same time, find 2 points
 * (on the right side) that are furthest away from the 2 lines above.
 * -- Step 3C: Next recursive calls

Parallellisering av Steg 2 og 3:
- For steg 2, deler jeg hele punktmengden slik at hver tråd kan jobbe med n/k punkter uavhengige fra hverandre. Hver tråd skal dele punkter i sin del inn i 2 lokale mengder (en som inneholder punkter på den venstre side av maxx->minx linje, og en som inneholder punkter på den høyre side). Alle tråder skal bruke en monitor som har left og right som globale variabler. Begge to er en IntList array som har plass til å lagre alle trådenes lokale mengder. Det betyr at vi kan asynkront oppdatere de to IntList arrayer når trådene fra steg 2 er ferdig.

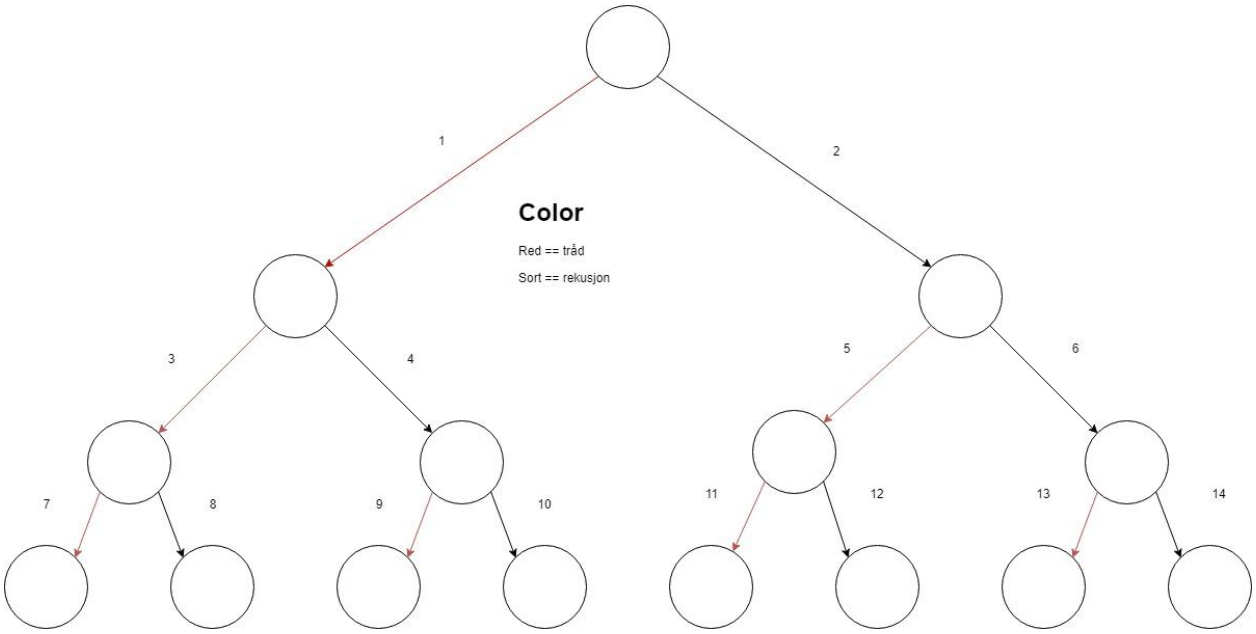
- Siden output som vi får fra steg 2 er left og right arrayer, må vi da endre signaturen til parRec() metoden i steg 3, slik at den kan håndtere en IntList array istedenfor bare et vanlig IntList objekt. Signaturen til parRec() ble endret fra:

```
public void parRec(int level, int p1, int p2, int p3, IntList ptSets, IntList koHyll);
```

Til:

```
public void parRec(int level, int p1, int p2, int p3, IntList[] ptSets, IntList koHyll);
```

- Parallellisering av rekursjonen i steg 3 er veldig enkel. Jeg har valgt å parallellisere dette steget på akkurat samme måte som i kvikksort: Hver node i rekursjons-treet representerer en tråd. Og hver tråd skal starte en ny tråd (som skal jobbe på venstre gren), mens denne tråden kan jobbe på høyre gren. Resultatene (dvs. koHyll listene) vil blir slått sammen etterpå ved å bruke metoden merge() i InnList klassen.



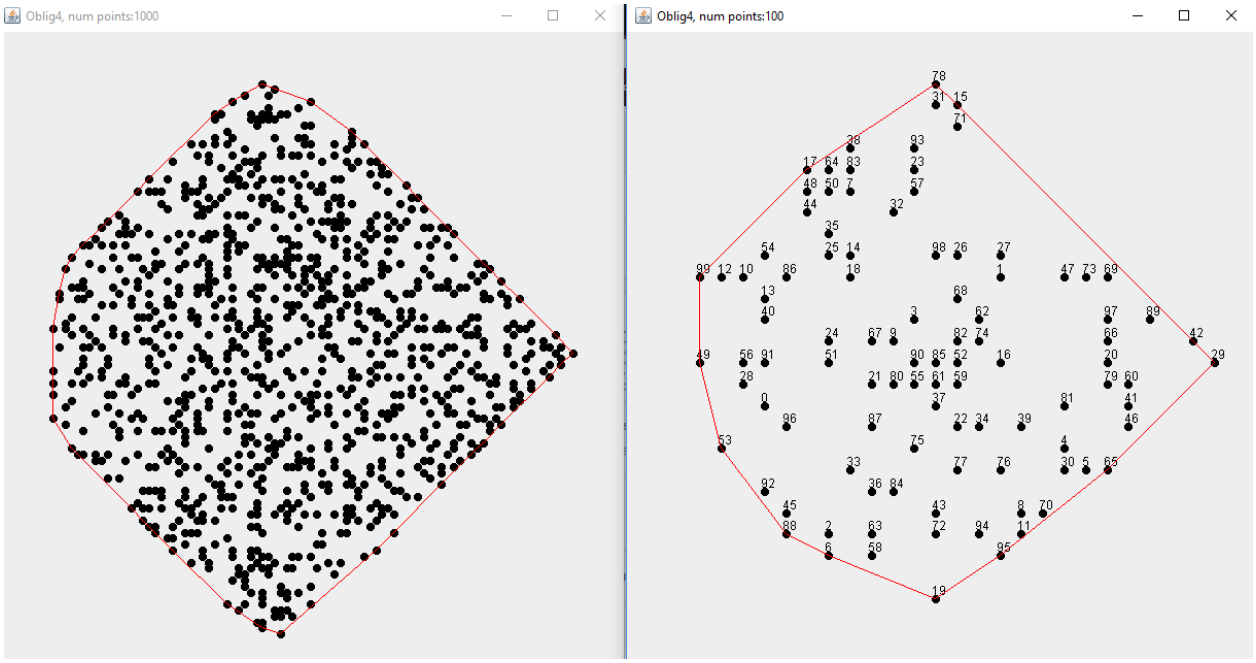
Resultat og Speedup

Programet var testet på min windows pc med 8 Intel(R) Core(™) i7-6700HQ CPU @ 2.60GHz. Programmet ble kjørt 7 ganger med 8 tråder. Tider i tabelen er medianen av dette:

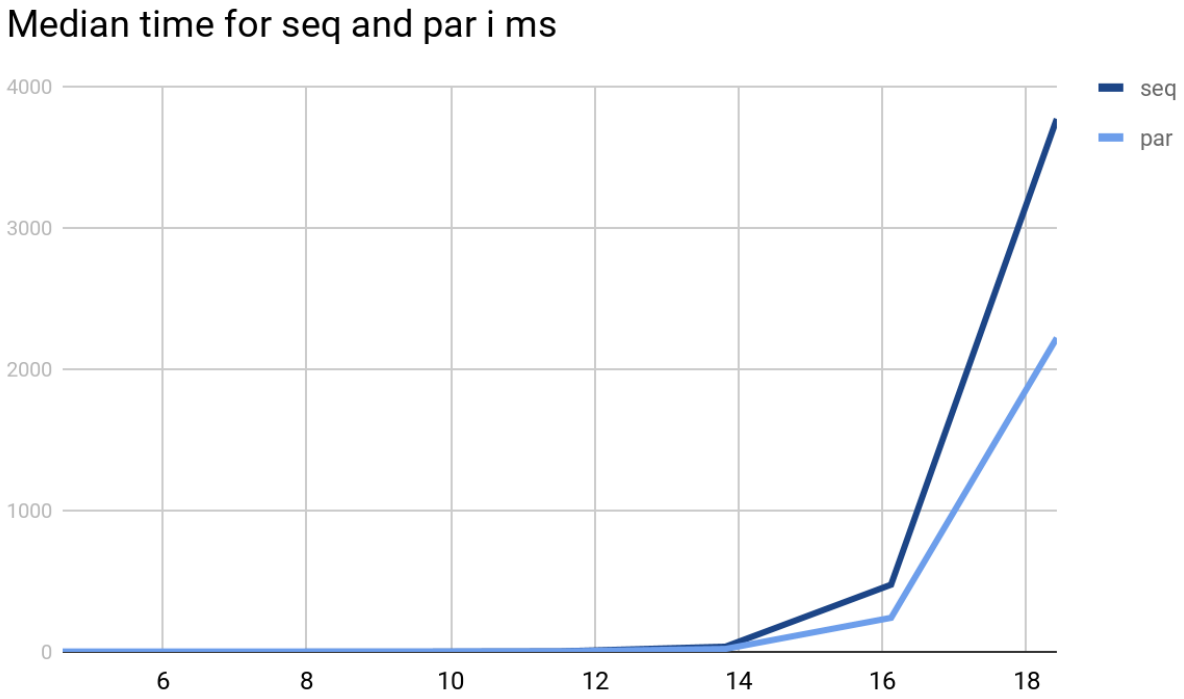
N verdi	Seq tid	Par tid	speedup
100 000 000	3771.099169 ms	2221.461026 ms	1,70
10 000 000	475.292632 ms	241.036342 ms	1,97
1 000 000	36.647126	20.950527	1,75
100 000	3.505384	3.247408	1,08
10 000	0.367012	2.761483	0,13

1000	0.026864	2.082371	0,01
100	0.004741	2.235261	0,00

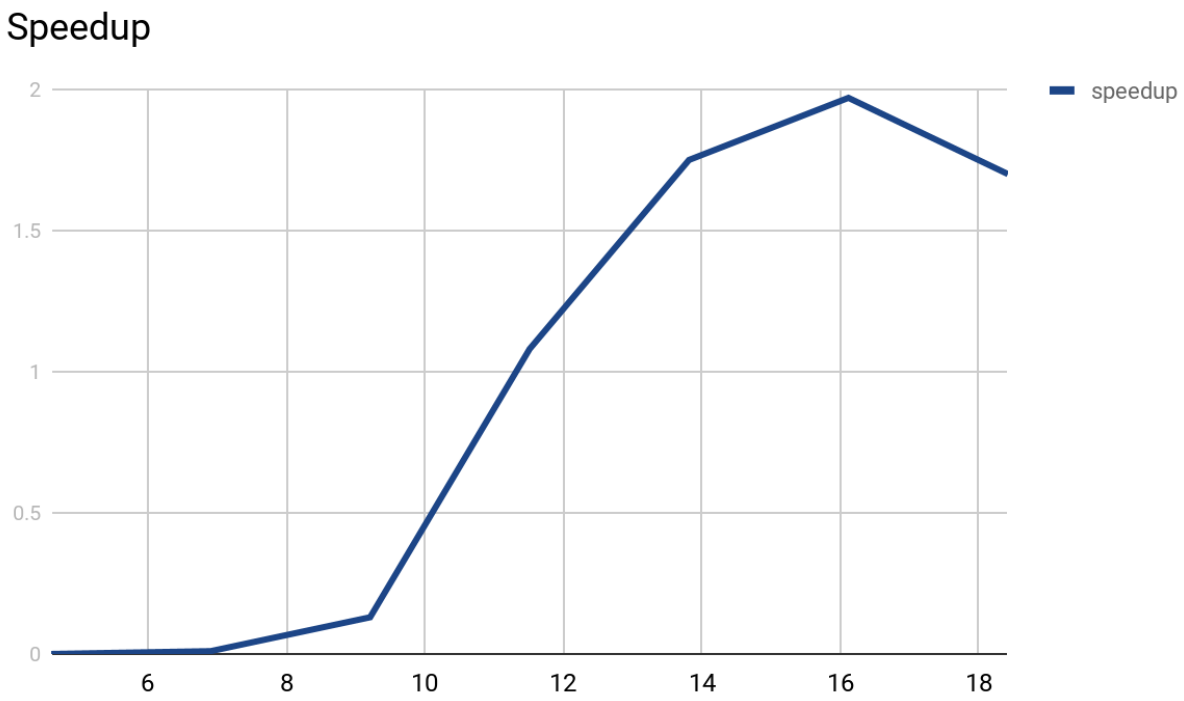
Bilde fra kjøring:



Graf for tid:



Graf for speedup:



Oppsummering:
Obligen var vanskelig og jeg er usikker på om jeg har parallellisert operasjonen på en god måte, hvis jeg får et andre forsøk på oppgaven ønsker jeg sterkt en god tilbakemelding så jeg bedre kan skjønne hva som er galt og hva som kan gjøres bedre.