Oblig 3
IN2010 Høst 2020
siljejka

Oppgave 1

Deloppgave 1

Alle metodene fungerer for sorteringer opp til 10000.

Deloppgave 2

Alle metodene returnerer resultater for sorteringer av opptil 1000 verdier for tidsbegrensninger på 100 milisekunder.

Deloppgave 3

For Bubble Sort kan man se en stor forskjell mellom den største verdien for tid i resultaene og resten, dette kunne forbedres ved å avslutte algoritmen tidligere. Den er derfor ikke like rask som den kunne vært. Merge Sort ville også vært raskere hvis algoritmen istedenfor å lage subarrayer, holdt styr på indexene for laveste og høyeste verdi i sub-arrayene og byttet om på verdier i det orginale arrayet.

For store lister med tilfeldige verdier så er Quick Sort og Merge Sort raskere, som forventet med gjenomsnittlig kompleksitet O(n log(n)) mot $O(n^2)$ i de to andre algoritmene, men denne forskjellene ville nok blit sett tidligere og vært større hvis Merge Sort var mer effektiv. Fra antall sammenligninger og bytter så tror jeg fortsatt at Quick Sort ville vært litt raskere enn Merge Sort gjennomsnittlig, men dette kommer også til å variere på input og pivot, siden Quick Sort er $O(n^2)$ for verste tilfelle.

For nesten sorterte lister så er Insertion Sort raskere enn alle de andre algoritmene, også for store n. Denne forskjellen er noe overdrevent fordi pivotet til Quick Sort er en ytre verdi, som gjør den mindre effektiv når listen er nesten sortert. Dette gjør derfor også Quick sort dårligere enn både Merge Sort og Bubble Sort for større verdier av n i de nesten sorterte listene, til tross for at både Merge Sort og Bubble Sort er dårlig optimert.

Oppgave 2

1) Den dyreste måten å koble sammen alle signaltårn på er å bruke alle koblingene, som vil si 73 km.

2) Med Kruskal's algoritme så får du kantene T2-T5, T5-T6, T1-T4, T4-T2, T3-T5, og T7-T6. Dette treet har vekt på 35 km. Da er alle nodene koblet, og alle andre kanter som kunne koblet dem er større.

Prims og Boruvkas kunne også ha vært brukt.

Oppgave 3

1) Algoritmen returnerer n + 1 for alle n, det er ingen flere steg, og kjøretiden er alltid en samme for alle n. Derfor er algoritmen O(1).

2) Algoritment kjører en loop n ganger for hver m, derfor er kjøretiden $m \cdot n$, eller $O(n \cdot m)$.

3) For hvert steg så halveres verdien av n til den er 1 eller mindre. Dermed er kjøretiden $O(\log n)$.