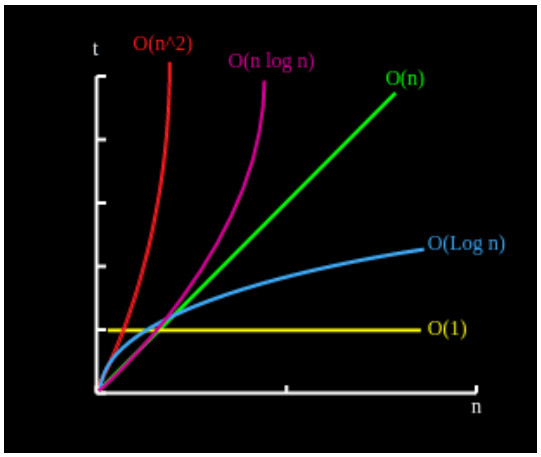
IN2010 innlevering 2 sameera, vetled, prebennc

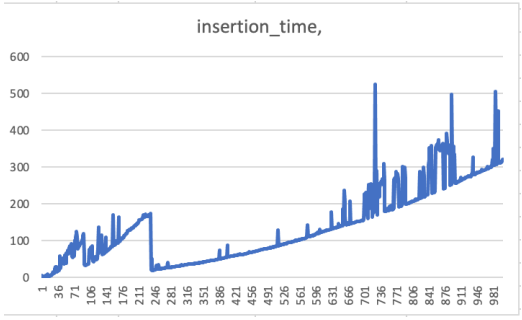
1) Vi har implementert quicksort, Insertion sort, heap sort, selection sort. Vi har testet disse med nearly-sorted_10 og 100. vi testet også med random filene opp til 1000.
2) Vi har målt sammenligninger og bytter ved at vi har brukt sorter.java sine metoder i algoritmene våre. Slik at vi kaller på metodene som gjør sammenligninger og bytter for oss.
3)

1)

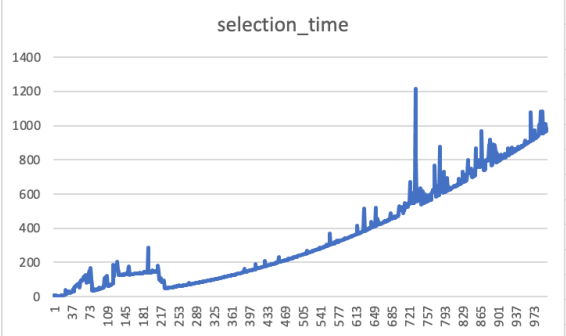
Dette bildet viser hva vi kan forvente de ulike kjøretidskompleksitetene til å se ut som i en graf.



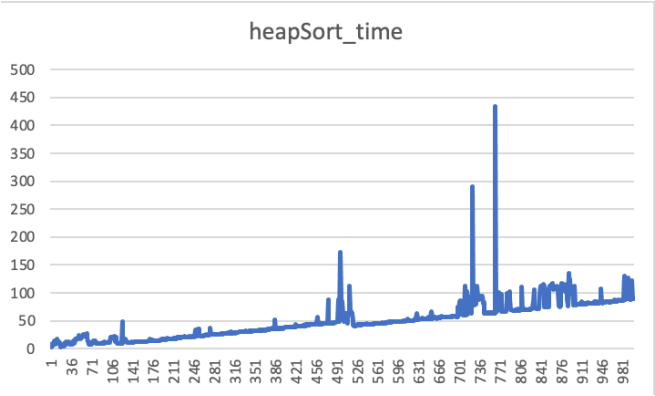
Insertion sort har kjøretidskompleksitet O(n^2). Vi plotter en graf over hvordan kjøretid stiger i forhold til inputstørrelse. Vi forventer en graf der, i verste fall, tid-aksen dobler seg i forhold til input aksen. Det vi ser er at grafen vokser raskere desto større input blir. Det er derfor tydelig at kjøretidskompleksiteten er større enn *linjær-tid*. Enkelte punkter på grafen går mye høyre enn den generelle veksten av grafen, det er input som er nærmere *verste tilfelle* enn hva vi typisk ser.



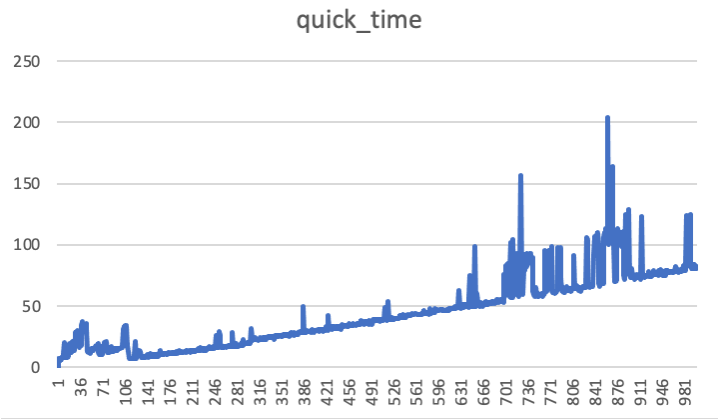
selection sort har også kjøretidskompleksitet O(n^2). Vi plotter en graf over hvordan kjøretid stiger i forhold til inputstørrelse. Vi forventer en graf der, i verste fall, tid-aksen dobler seg i forhold til input aksen. Det vi ser er at grafen vokser raskere desto større input blir. Det er derfor tydelig at kjøretidskompleksiteten er større enn *linjær-tid*. Enkelte punkter på grafen går mye høyre enn den generelle veksten av grafen, det er input som er nærmere *verste tilfelle* enn hva vi typisk ser.



Heap sort har kjøretidskompleksitet O(n log(n)). Vi plotter en graf, F(x), over hvordan kjøretid stiger i forhold til inputstørrelse. Vi forventer at F(x) vokser mellom linjær-tid og n^2. Det vi ser er at grafen, F(x), vokser raskere desto større input blir. Det er derfor tydelig at kjøretidskompleksiteten er større enn *linjær-tid*. Enkelte punkter på grafen går mye høyre enn den generelle veksten av grafen, det er input som er nærmere *verste tilfelle* enn hva vi typisk ser.

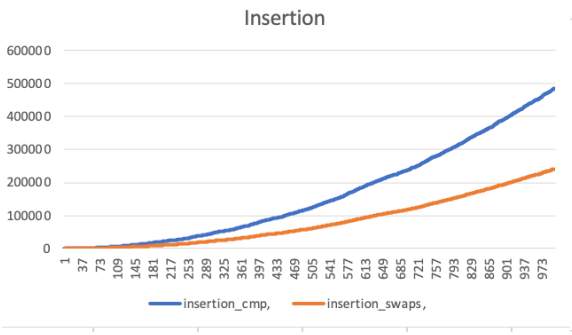


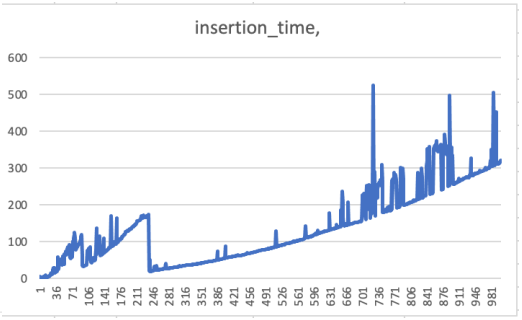
quick sort har også kjøretidskompleksitet O(n^2). Vi plotter en graf over hvordan kjøretid stiger i forhold til inputstørrelse. Vi forventer en graf der, i verste fall, tid-aksen dobler seg i forhold til input aksen. Det vi ser er at grafen vokser raskere desto større input blir. Det er derfor tydelig at kjøretidskompleksiteten er større enn *linjær-tid*. Enkelte punkter på grafen går mye høyre enn den generelle veksten av grafen, det er input som er nærmere *verste tilfelle* enn hva vi typisk ser.



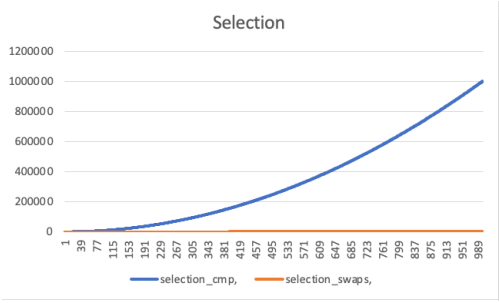
2)

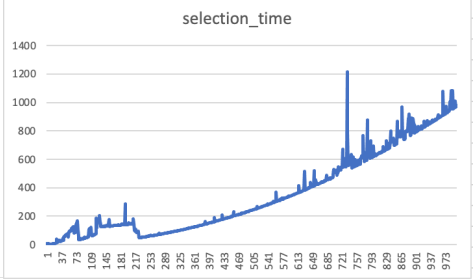
Vi har plottet to grafer, en som viser oss hvordan cmps and swaps stiger i insertion sort ettersom input blir større. Vi har også en graf som viser oss hvordan kjøretiden endrer seg ettersom input blir større. Det er en klar relasjon mellom disse, det syntes som om de vokser i takt.



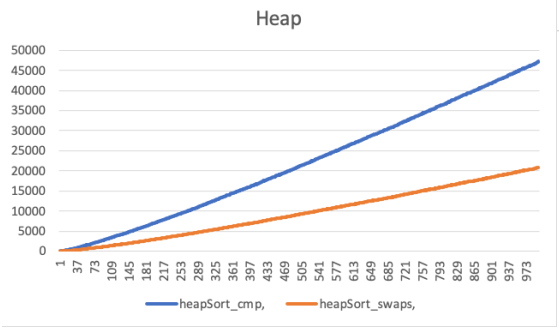


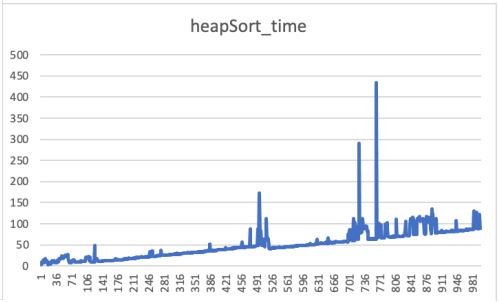
Her på selection sort ser vi at det er veldig mange sammenligninger i forhold til bytter. Vi kan se at sammenligninger vokser raskere når input blir større, det samme ser vi på tiden. Dermed kan vi se at det er en klar relasjon mellom sammenligninger og tidsbruk.



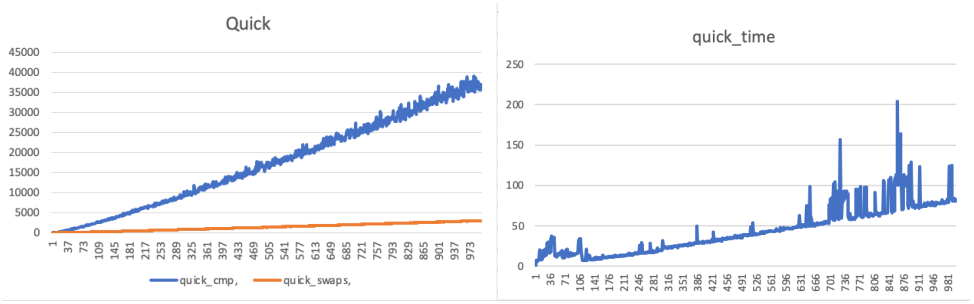


Vi ser at sammenligninger og bytter på heap sort vokser raskere enn linjær-tid, ettersom at vi ser en liten bue i grafen. Det er det samme vi ser i tidsbruken på algoritmen.



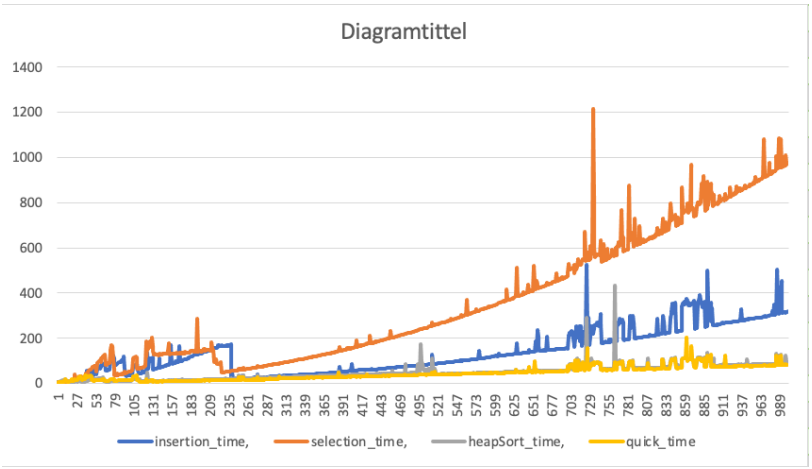


Vi ser at på quick sort at det er veldig mange sammenligninger i forhold til bytter. Vi kan se at sammenligninger vokser raskere når input blir større, det samme ser vi på tiden. Dermed kan vi se at det er en klar relasjon mellom sammenligninger og tidsbruk.



3)

Vi ser at SelectionSort utmerker seg som dårlig både ved store og små inputs, men spesielt ved store inputs. QuickSort og HeapSort utmerker seg som bra på små og store inputs. InsertionSort er et sted midt imellom på store inputs, men behandler små inputs mindre effektivt QuickSort og HeapSort.



4) Her har vi to grafer som viser tidsbruk på nearly_sorted_1000 og random_1000. Vi ser at selectionSort tar merkbart mye lengere tid på nearly sorted. Den er fremdeles algoritmen som tar lengst tid. Ettersom at grafen nearly_sorted_1000 blir forskjøvet av størrelsen på selectionSort får det de tre andre algoritmene til å se mye jevnere ut, men vi legger merke til at denne gangen tar quickSort legere tid enn hva HeapSort og insertionSort gjør.