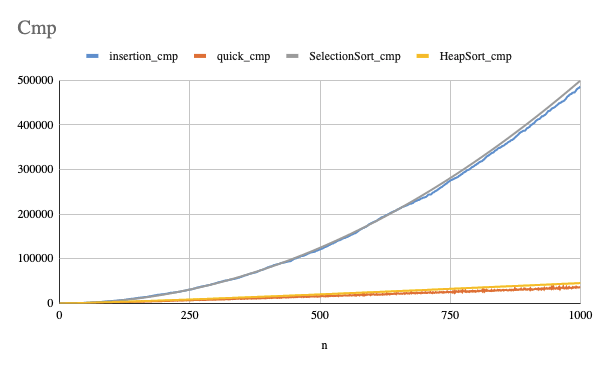
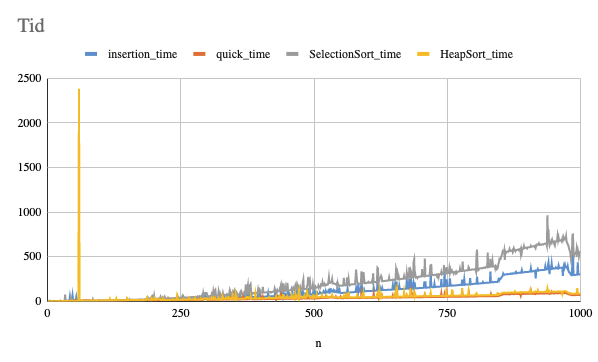
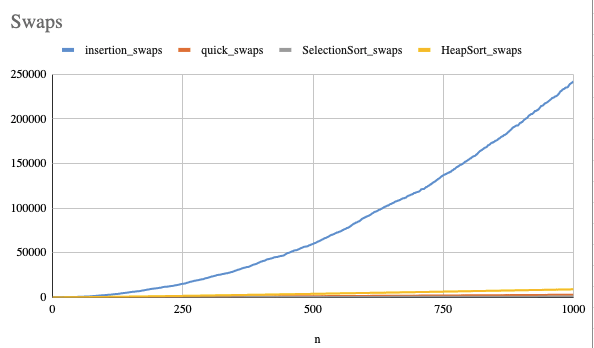
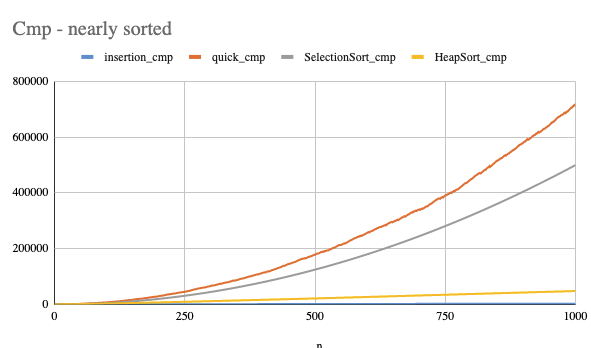
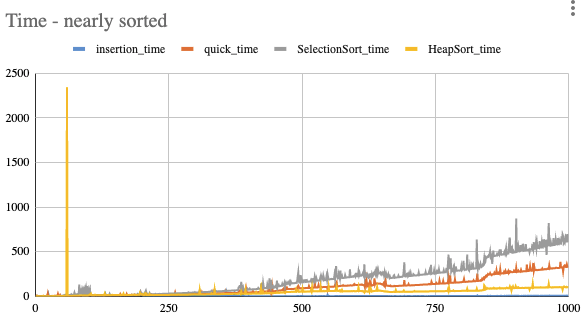
Deloppgave 3

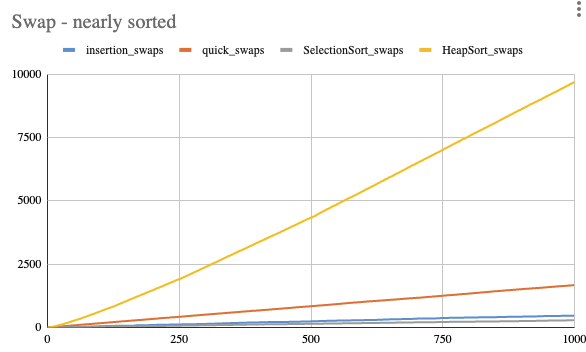
Grafene viser swaps, cmp og tiden til hver av algoritmene for random\_1000 og nearly\_random­\_1000



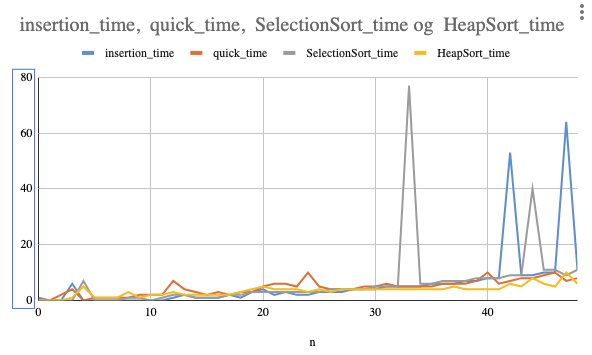
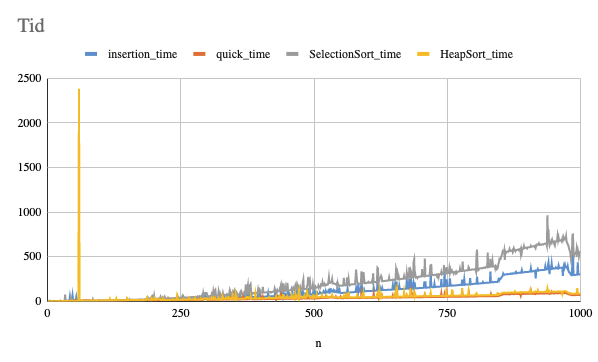








Hvis vi tar utgangspunkt i random\_1000 ser vi at kjøretiden stemmer godt overens med de forventningene vi får av kjøretidsanalysen. De algoritmene som har O(n^2) kjører en god del senere enn de som har O(n\*log(n)). Likevel ser vi forskjeller mellom de to algoritmene som har O(n^2), selv om de har samme worst case.



Diagrammet til venstre viser kjøretiden fra n = 0 til n = 50 for å se på hvilke algoritmer som presterer bra ved liten n. Vi ser at det ikke er noen som utmerker seg veldig, men at noen har noen spesielle verdier for n der de bruker veldig lang tid. Dette gjelder spesielt for heap sort som man ser bruker veldig lang tid på en spesiell verdi for n.

Ser man på nearly\_sorted\_1000 sitt kjøretidsdiagram ser man at insertion sort bruker utrolig liten tid. Selv om worst case er høyere for den en for heap sort og selection sort, er det den som kjører raskest. Man ser også at både swap og cmp er veldig lav for insertion sort om verdiene nesten er sorterte. Dette er fordi denne algoritmen bare flytter på de nødvendige verdiene, og i nearly sorted er det veldig få nødvendige bytter å gjøre.

Man kan også observere at heap sort har mange swaps når listen nesten er sortert. Dette er fordi algoritmen bygger en max heap, implementert som et tre. I en nesten sortert liste vil alle de høyeste verdiene komme langt nede i listen, og det må derfor gjøres mange bytter for å bygge dette treet.

Noe vi ikke hadde sett for oss er at heap sort bruker utrolig lang tid på sortering av listen når n = 59, både for random og nearly sorted.

Oppgave 3

O-notasjon

1)

Den første algoritmen endrer bare på et tall og er derfor konstant tid O(1). Den endrer ikke kjøretid avhengig av input.

2)

Den andre algoritmen looper gjennom m-elementer før den looper igjen igjennom n-ganger.

På grunn av loopen inni loopen er algoritmen O(m\*n). Kallet på Algor1 påvirker ikke o-notasjonen.

3)

Den 3. algoritmen kjører en while-loop der den kaller på algoritme 1, før den halverer «søkefeltet sitt». Kallet på algoritme 1 påvirker ikke kjøretiden noe særlig. Algoritme 3 er derfor O(log n).