IN2010 – oblig 3

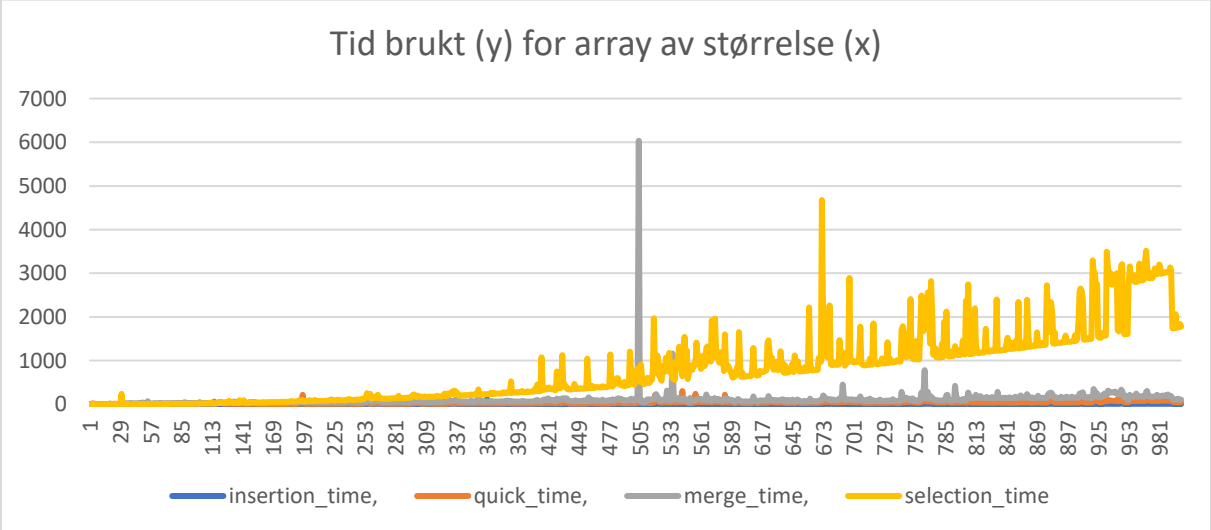
Del 1 – korrekthet

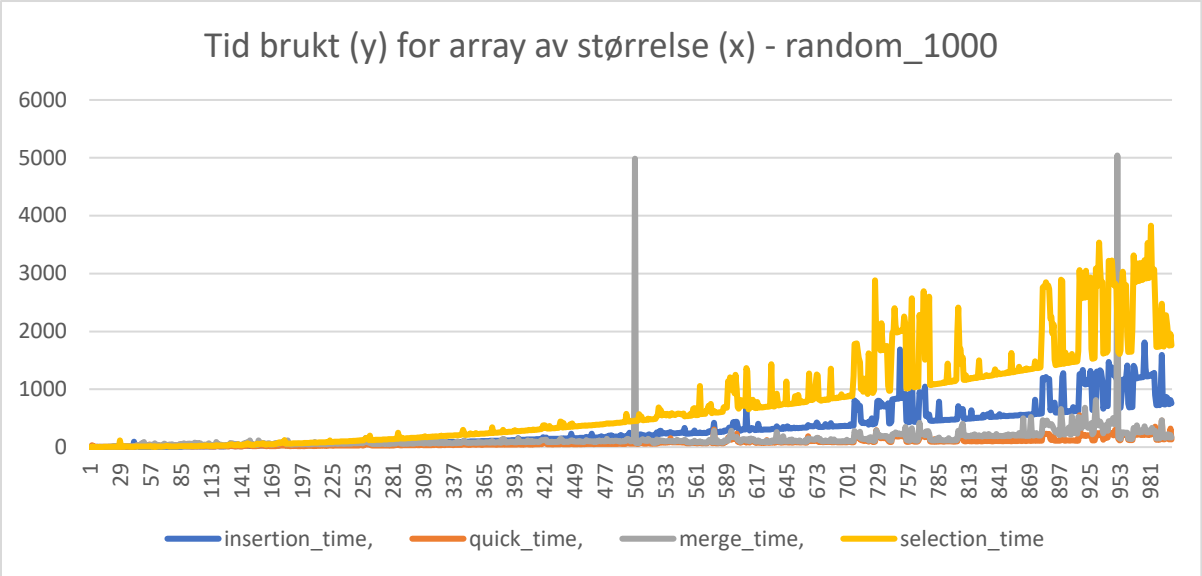
I oppgaven er det implementert fire ulike sorteringsalgoritmer. Først ble Insertion sort og deretter Quicksort implementert, som var en del av kravet. Deretter ble Merge- og Selection sort implementert for *O(nlog(n)* og *O(n^2)* kravene. Algoritmene har blitt testet ved å bruke inputfiler med størrelse n = 10, 100, 1000 og 100000. Deretter har output filene som koden produserer blitt manuelt gått igjennom for å kontrollere at sorteringen er riktig.

Del 2 – Sammenligninger, bytter og tid

For implementering av de ulike algoritmene, blir bytter og sammenligner målt ved hjelp av hjelpemetoder som utfører byttet eller sammenligningen samtidig som den øker en teller variabel. I denne implementasjonen brukes hjelpemetoder for sammenligninger hyppig, da den ikke bare finnes i if-sjekker, men også i for-loops og while-loops.

Del 3





Insertion – $O(n^2)$

Quick – $O(n^2)$ worst case, skjer sjeldent, O(nlog(n)) best case.

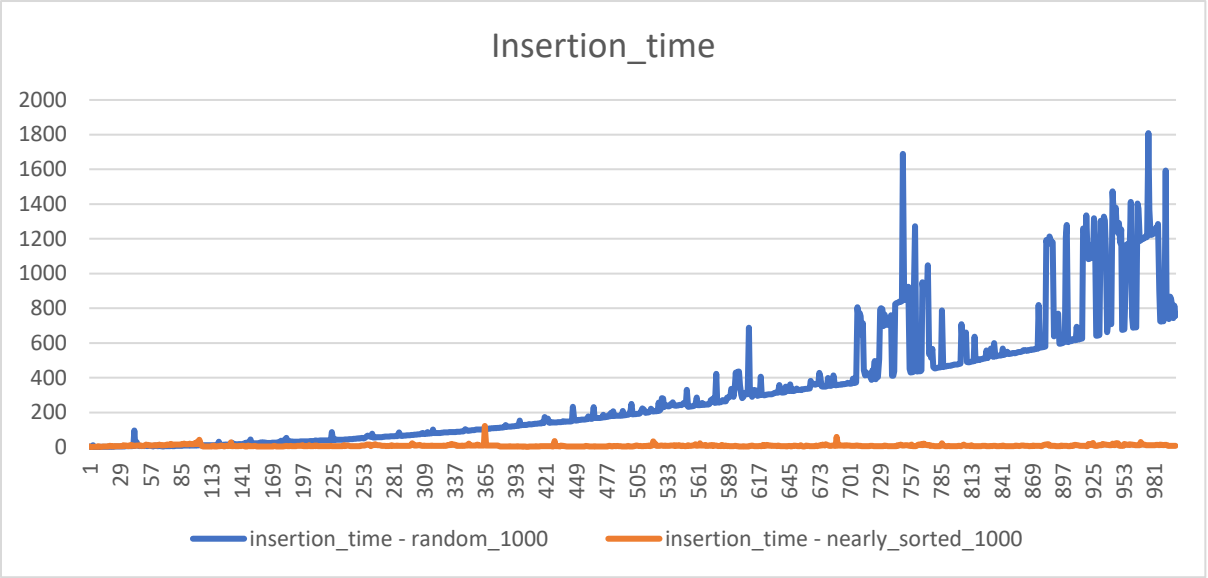
Merge – O(nlog(n))

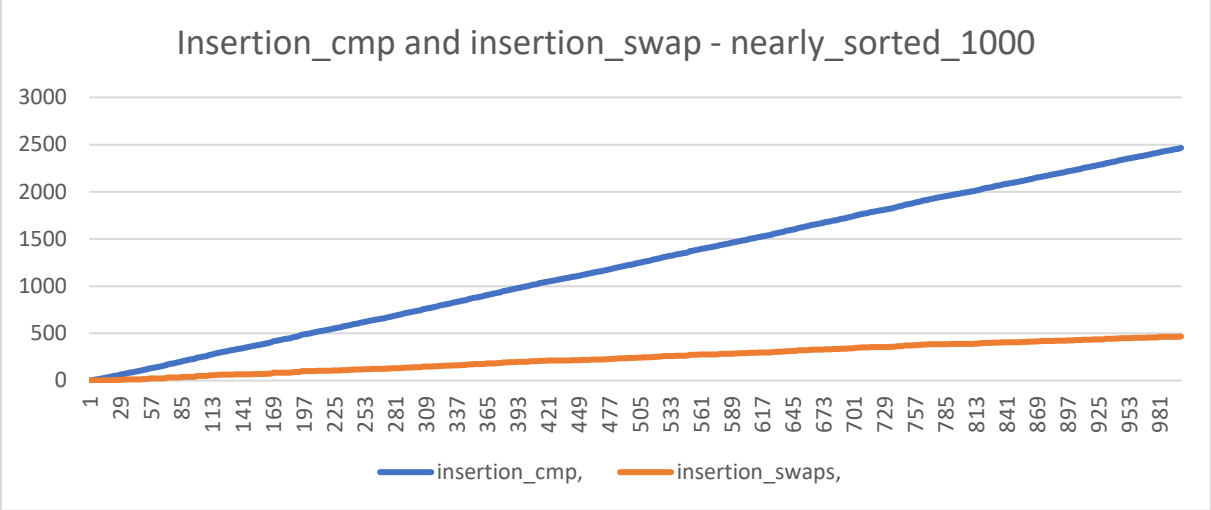
Selection – $O(n^2)$

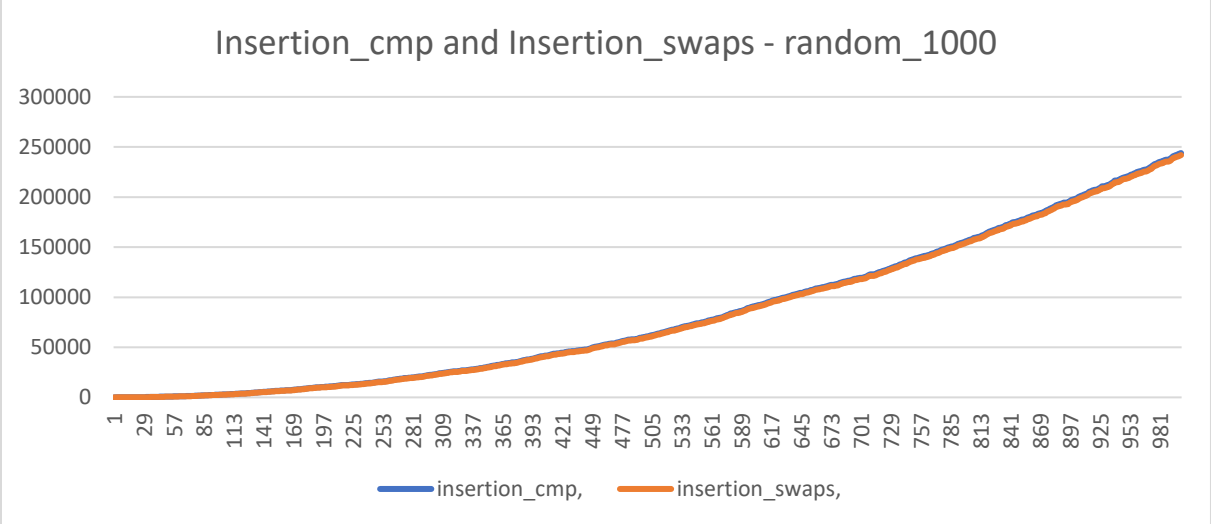
Insertion sort har O notasjon lik $O(n^2)$, men bemerker seg som den mest effektive når arrayet har størrelse lik 1000 og nesten er sortert. Merge sort derimot, som burde være den mest effektive algoritmen med sin O(nlog(n)) kjøretidskompleksitet, havner på en tredjeplass etter Quick sort, som i sitt worst case kan havne på $O(n^2)$ kjøretid. Til slutt kommer Selection sort, som helt klart bruker lengst tid. Så for et array med størrelse n = 1000, stemmer ikke den faktiske kjøretiden med hva man skal kunne forvente av de ulike algoritmene.

For et array et tilfeldig array med størrelse lik 1000, utmerker merge- og quick sort seg som de beste algoritmene. Detter stemmer med forventingen før kjøring, ettersom merge- og quick kan oppnå O(nlog(n)) kjøretidskompleksitet, mens insertion og selection bare oppnår $O(n^2)$.

Insertion

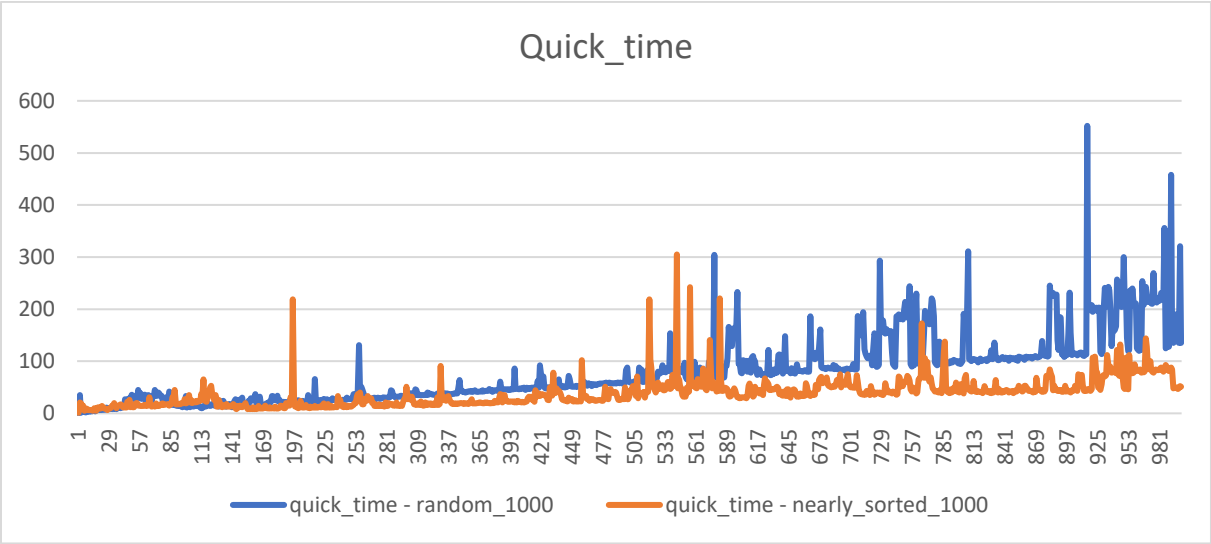

Insertion_time
2000
1800
1600
1400
1200
1000
800
600
400
200
0
1 29 57 85 113 141 169 197 225 253 281 309 337 365 393 421 449 477 505 533 561 589 617 645 673 701 729 757 785 813 841 869 897 925 953 981
insertion_time - random_1000
insertion_time - nearly_sorted_1000

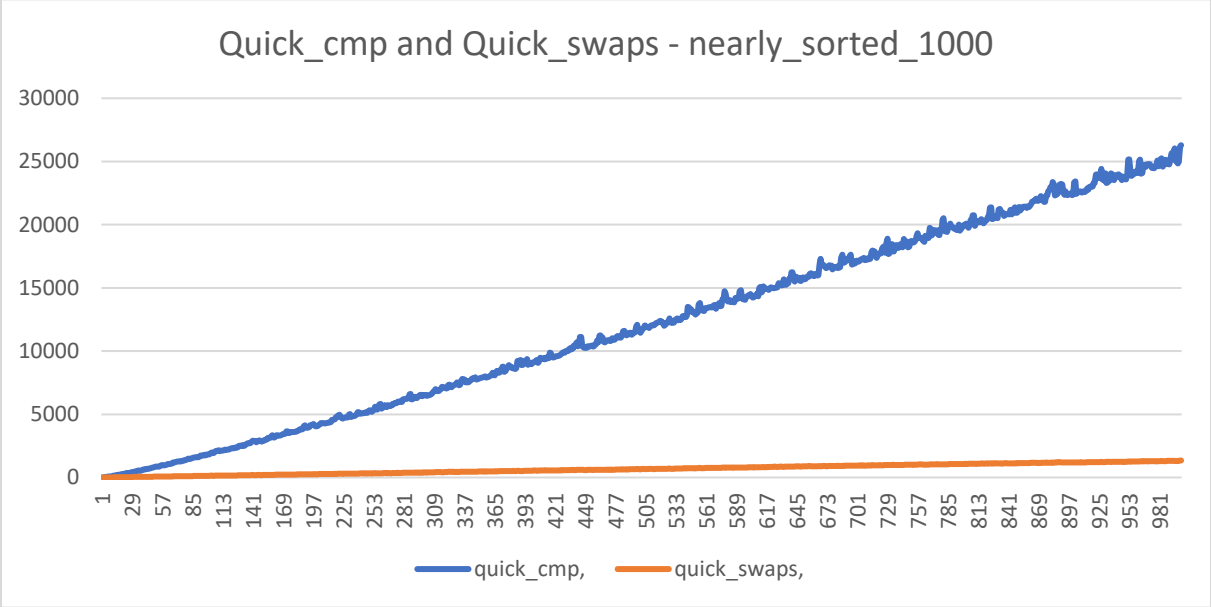

Insertion_cmp and insertion_swap - nearly_sorted_1000
3000
2500
2000
1500
1000
500
0
1 29 57 85 113 141 169 197 225 253 281 309 337 365 393 421 449 477 505 533 561 589 617 645 673 701 729 757 785 813 841 869 897 925 953 981
insertion_cmp,
insertion_swaps,

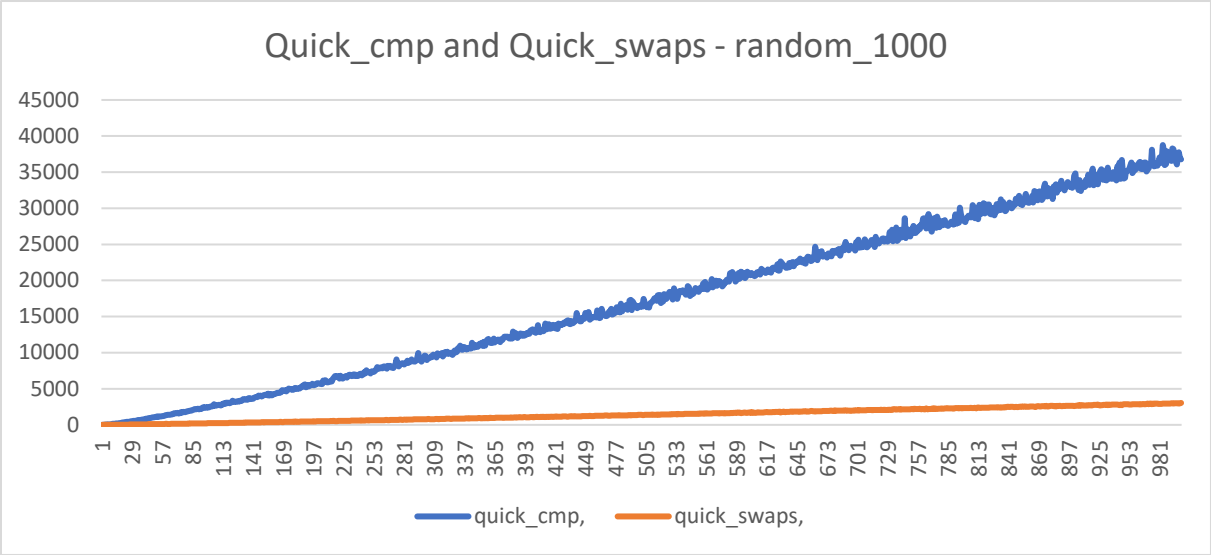

Insertion_cmp and Insertion_swaps - random_1000
300000
250000
200000
150000
100000
50000
0
1 29 57 85 113 141 169 197 225 253 281 309 337 365 393 421 449 477 505 533 561 589 617 645 673 701 729 757 785 813 841 869 897 925 953 981
insertion_cmp,
insertion_swaps,

For Insertion sort gir flere sammenligninger og bytter lengre kjøretid. Når arrayet nesten er sortert, stiger kjøretiden minimalt i forhold til hvor mye antallet sammenligninger og bytter stiger. For et tilfeldig array derimot, gjør insertion sort mange omtrent 100 ganger flere sammenligninger. Antallet bytter er også omtrent det samme som antallet sammenligninger, og algoritmen bruker naturligvis lengre tid.

Quick

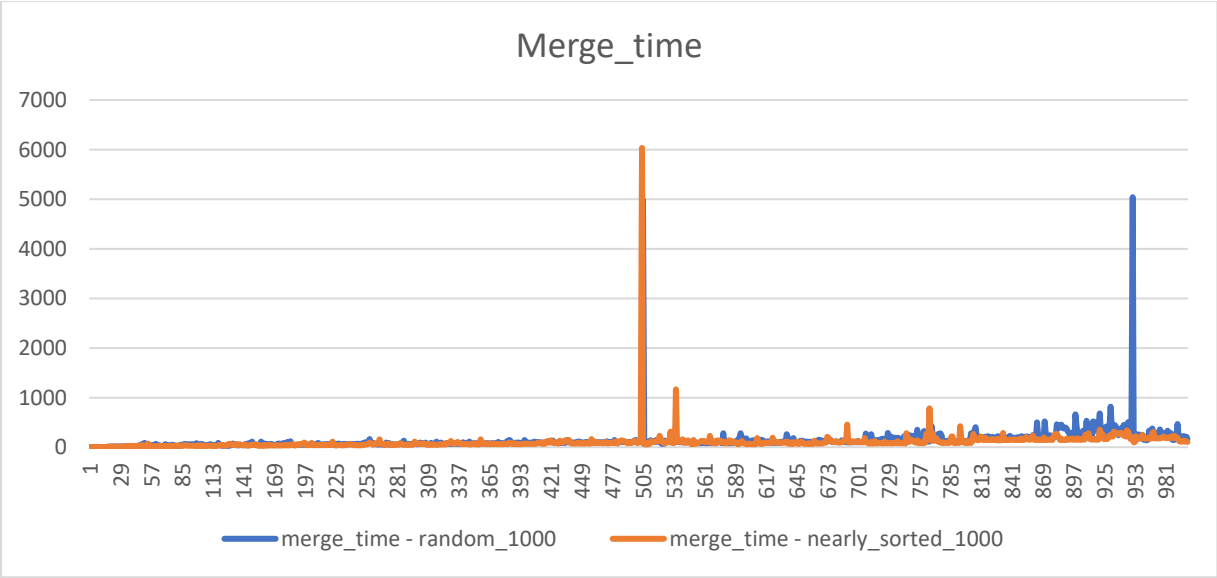


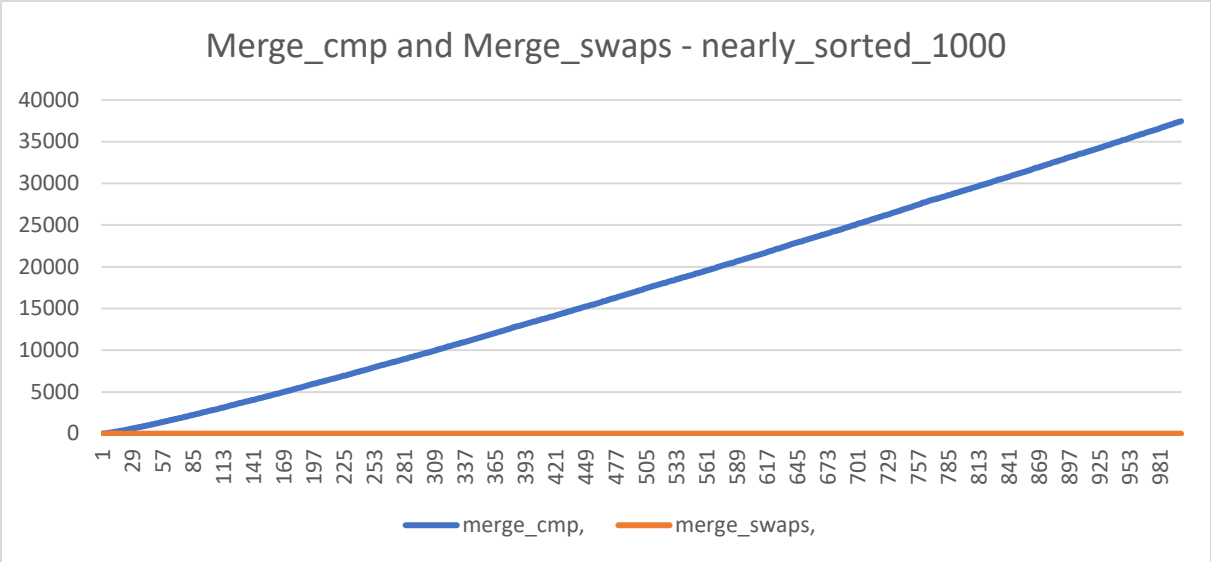


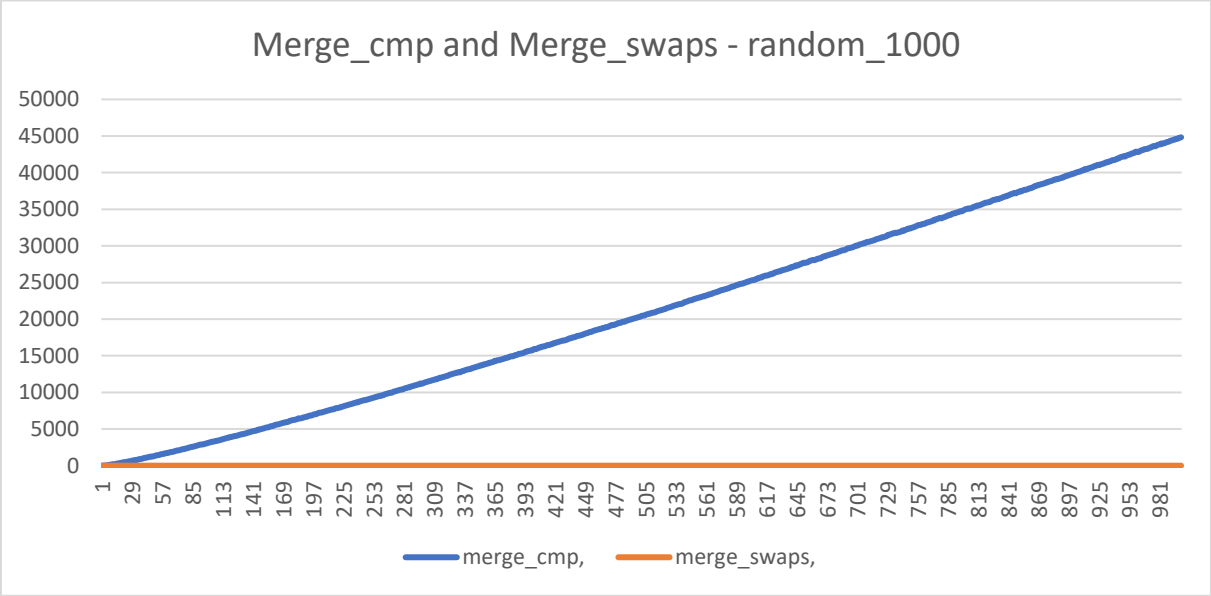


Quick sort gir de samme konklusjonene som insertion. Flere bytter og sammenligninger fører til lengre kjøretid, og det tilfeldige arrayet bruker da lengst tid på å bli sortert, som følge av flere bytter og sammenligninger. En forskjell å legge merke til for quick sort, er at forskjellen på antallet bytter og sammenligninger ikke er like stort for de ulike arrayene som for insertion.

Merge

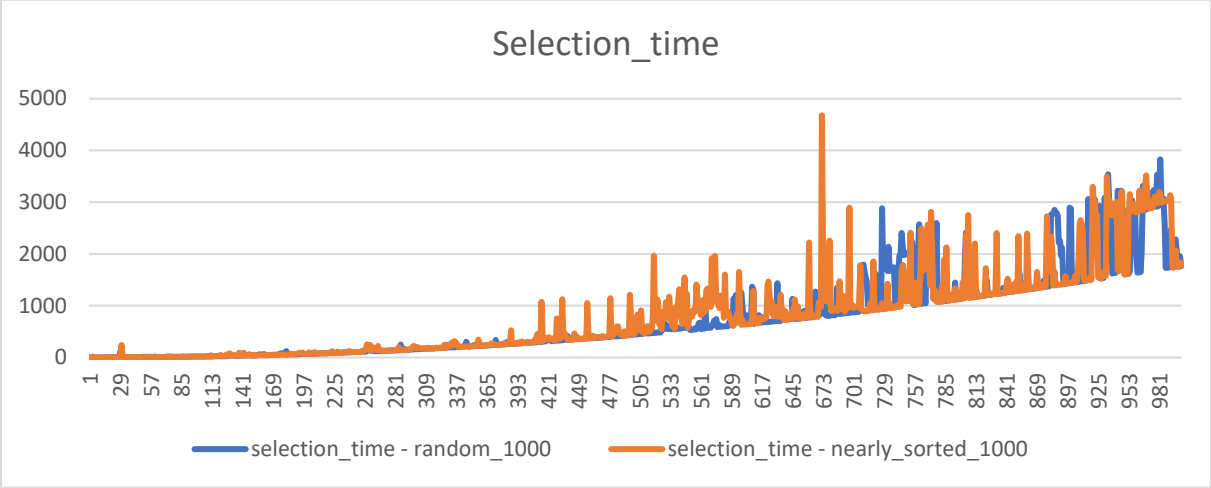


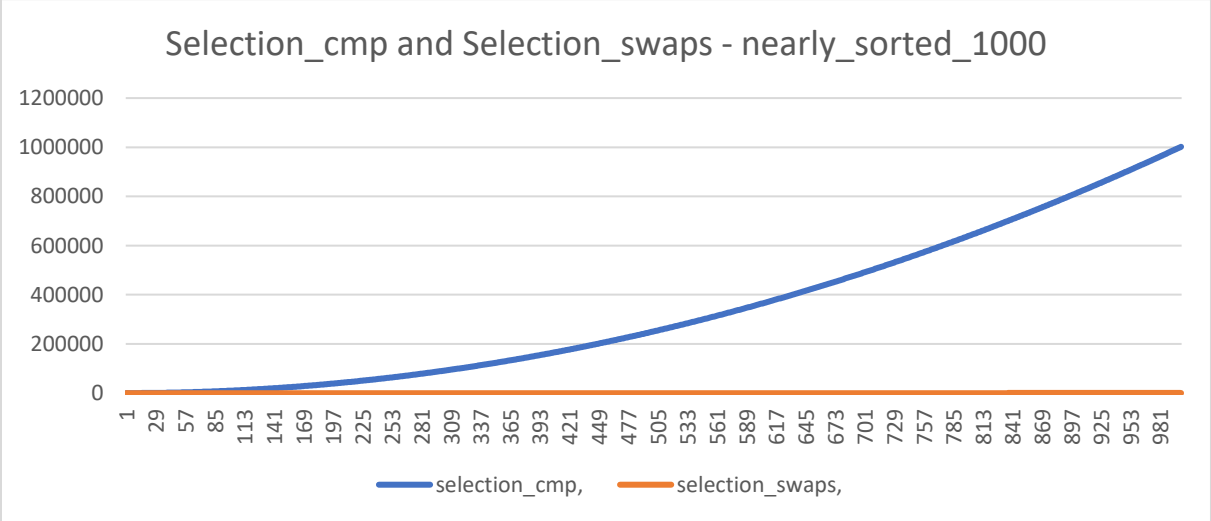


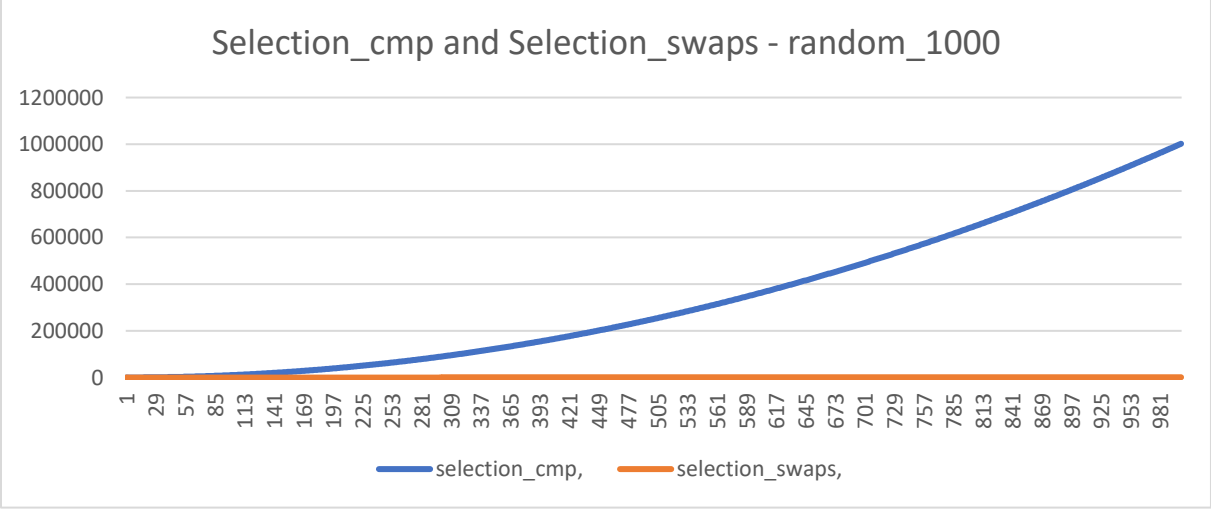


For merge sort er det kun antallet sammenligninger som påvirker kjøretiden, da det i merge algoritmen ikke forekommer noen bytter. Algoritmen følger en lineær trend med unntak for noen verdier av n, der kjøretiden plutselig får en stor økning i verdi. Her bruker også det tilfeldige arrayet lengre tid på å bli sortert, siden det forekommer flere bytter.

Selection

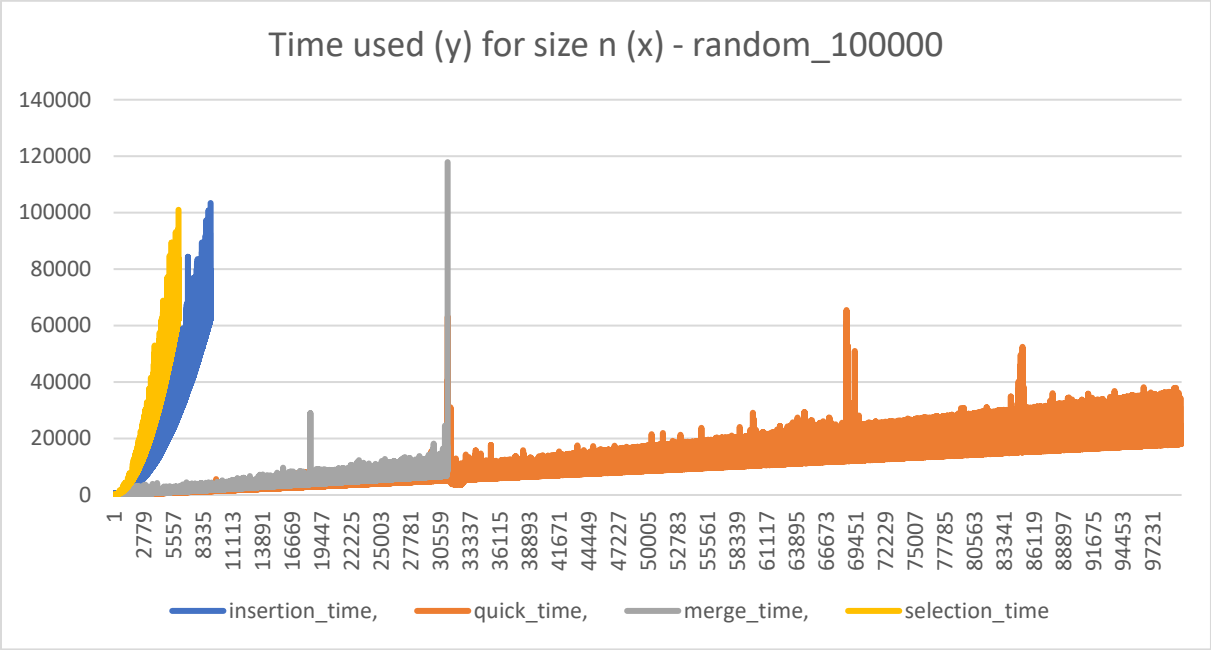


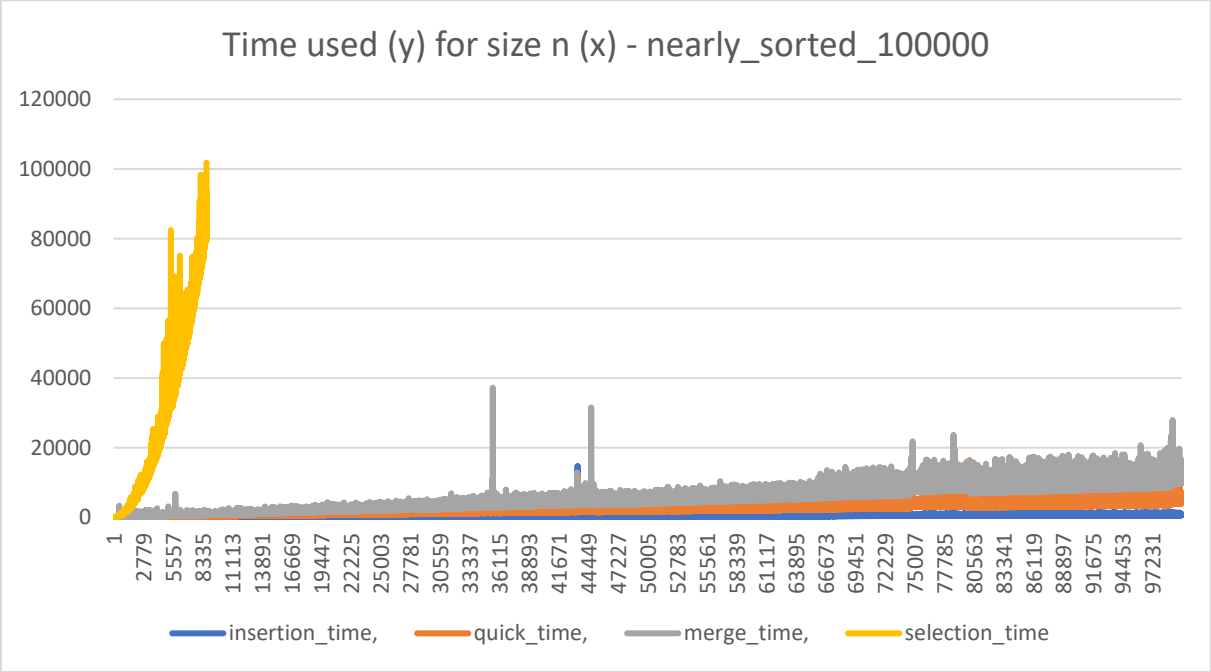




Selection sort har relativt lik kjøretid for de to forskjellige arrayene. Her er også antallet sammenligninger veldig likt, og forskjellen ligger i antallet

bytter. Forskjellen er like vel såpass liten at det har veldig lite å si for kjøretiden til algoritmen.





Når n blir veldig stor, fungerer quick sort best for tilfeldige array, og er den eneste algoritmen som ikke overgår tidsbegrensningen. For et nesten sortert array, er insertion den mest effektive algoritmen når n blir stor.