IN2010 – Oblig 3

Oppgave 1

Sorteringsalgoritmer som er implementert:

- InsertionSort
  - $O(n^2)$
- QuickSort
  - Worst-case $O(n^2)$
  - Best-case O(nlog(n)), kommer an på valg pivotelementet
- SelectionSort
  - $O(n^2)$
- HeapSort
  - O(nlog(n))

Har testet alle algoritmene med de minste filene (example, nearly_sorted_10,100,1000 og random_10,100,1000) og gått gjennom .out-filene og ser at de er sortert. Ser spesielt på bunnen av de nye sorterte listene for å sammenligne om alle er like.

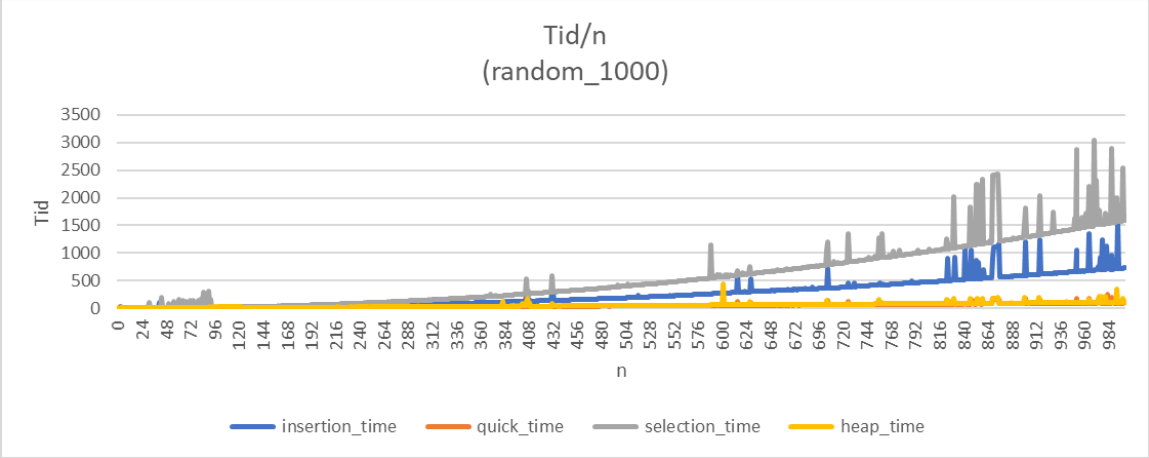
Oppgave 2

Sammenligninger og bytter:

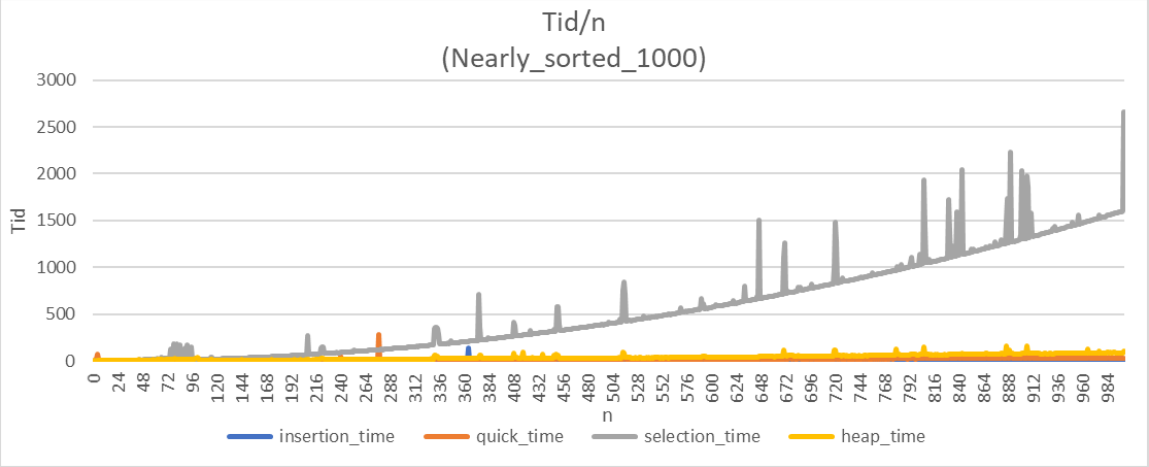
- Sammenligninger blir telt hver gang man gjør en sammenligning. Både i for-løkker og while-løkker samt utenfor løkker.
- Bytter blir telt hver gang jeg bytter to verdier i arrayen, men ikke når jeg bare bytter to tall alene.

Oppgave 3

I hvilken grad stemmer kjøretiden overens med kjøretidsanalysene for de ulike algoritmene?

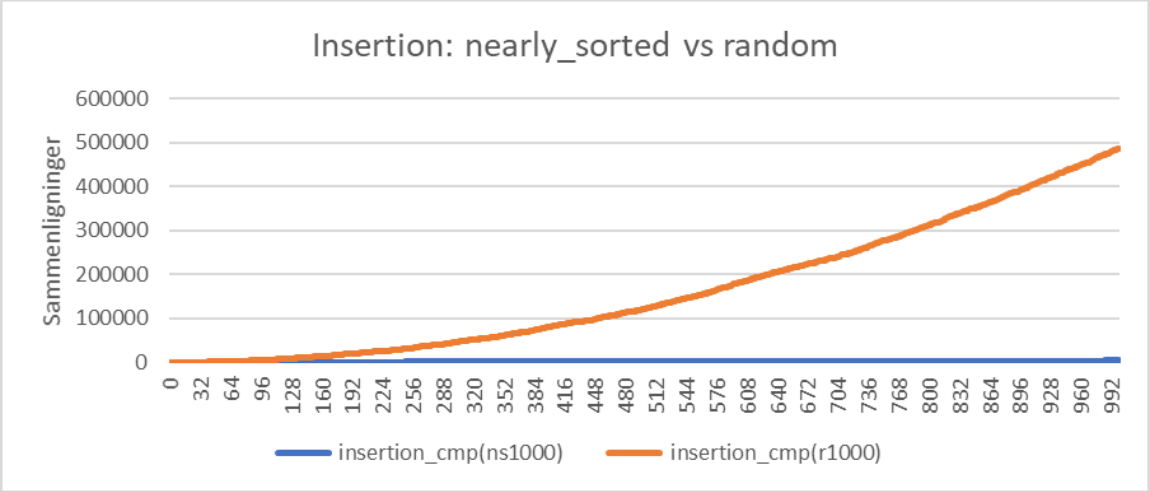


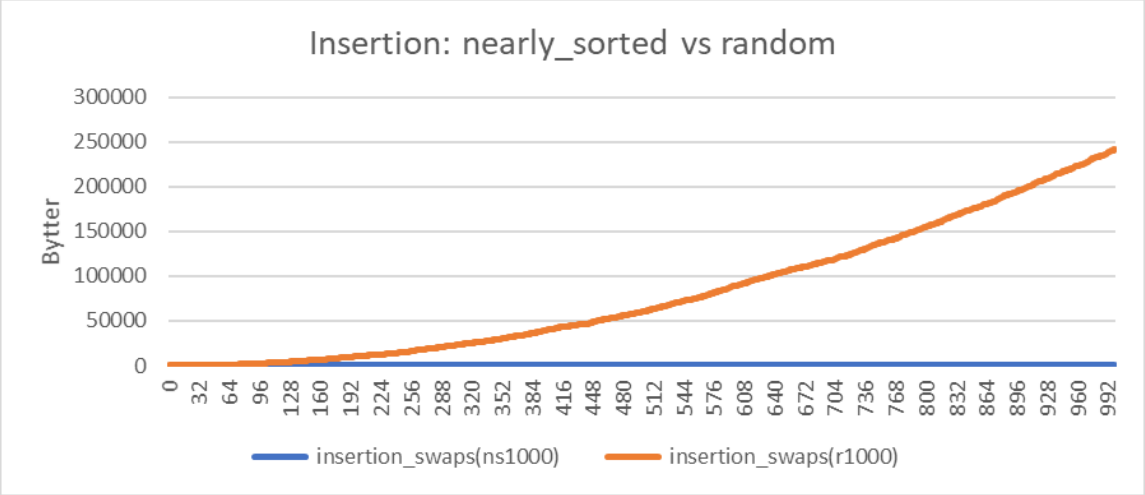
Som man ser på grafen, bruker SelectionSort(grå) og InsertionSort(blå) lengst tid på å sortere 1000 tilfeldige tall. Det stemmer fint overens med $O(n^2)$. Man ser også at QuickSort bruker ganske kort tid som tilsier at algoritmen valgte et godt pivot-element. HeapSort bruker også veldig kort tid som stemmer overens med O(nlog(n)).

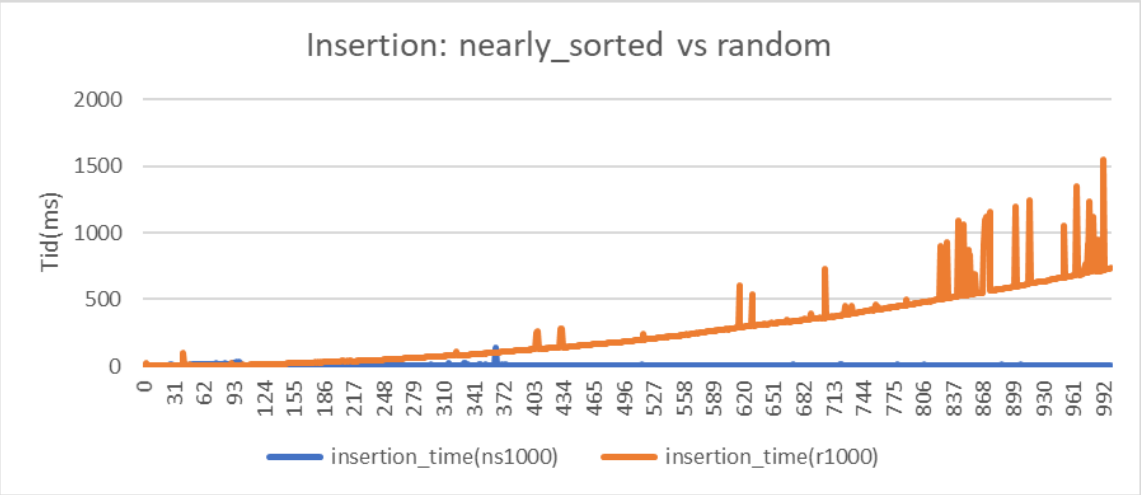


Denne grafen viser tiden algoritmene bruker på å sortere et nesten sortert array. Den ser ganske lik ut som forrige graf, men man ser tydelig at InsertionSort(blå), nå bruker mye kortere tid. Dette gir mening da InsertionSort ofte bryter ut av den indre loopen som gjør den mye raskere på nesten sorterte arrayer.

Hvordan er antall sammenligninger og antall bytter korrelert med kjøretiden?







De fleste sorteringsalgoritmene bruker ikke så veldig mange bytter eller sammenligninger, utenom InsertionSort. InsertionSort bruker ekstremt mange bytter og sammenligninger når den skal sortere tilfeldige tall. Dette kan man også se på tiden den bruker på sortere de tilfeldige tallene.

Antall sammenligninger for InsertionSort:

n	Nearly_sorted_1000	random_1000	Økning (%)
1000	3 931	486 775	12 383%

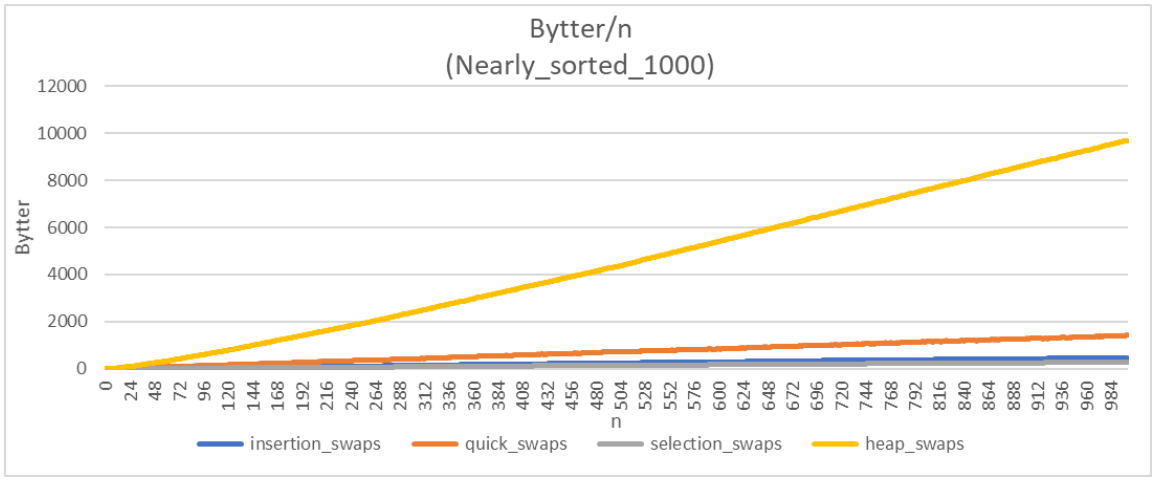
Antall bytter for InsertionSort:

n	Nearly_sorted_1000	random_1000	Økning (%)
1000	467	241 892	51 797%

Tid for InsertionSort:

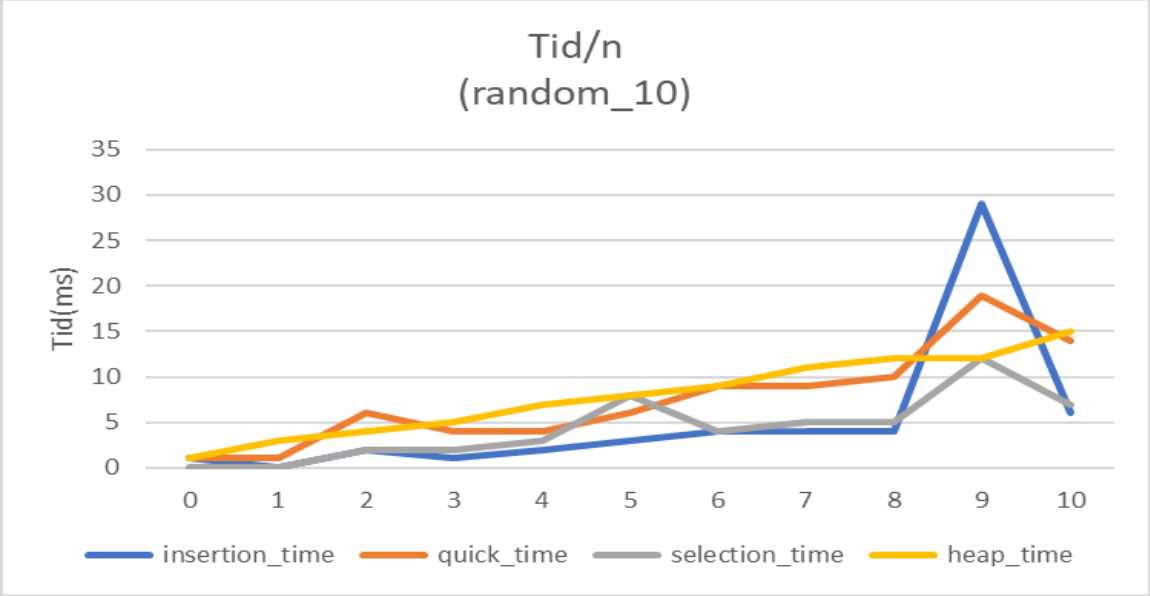
n	Nearly_sorted_1000	random_1000	Økning (%)
1000	7 ms	735 ms	10 500%

Man ser at det er en tydelig forskjell på kjøretiden når det er mange bytter og sammenligninger vs få sammenligninger og bytter. For de andre sorteringsalgoritmene syns ikke dette like godt, siden de nesten sammenligner og bytter like mye for nesten sortert og tilfeldige tall.

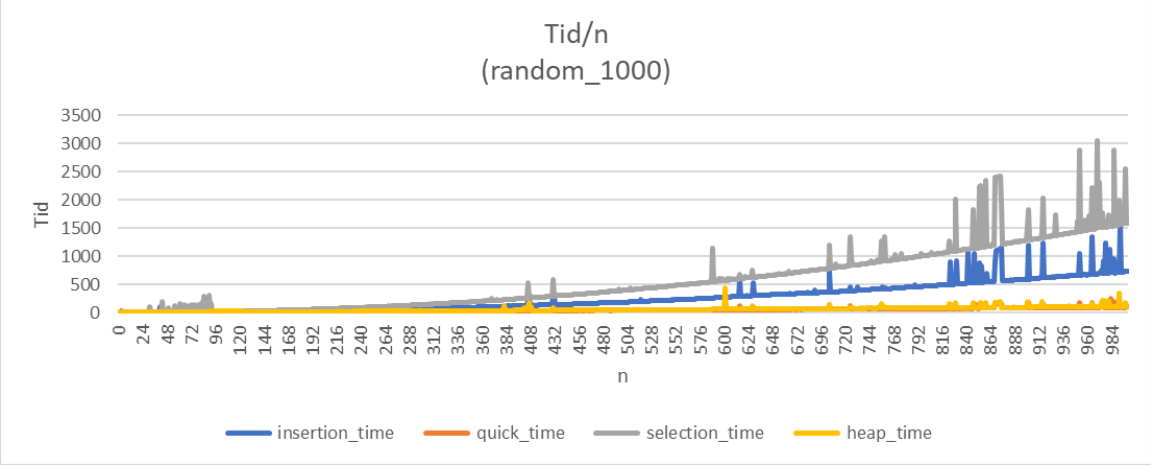


På grafen over ser man at HeapSort bytter lineært. Men som vi ser på kjøretidsgrafene, bruker ikke HeapSort noe vesentlig mye mer tid ved stor n selv om den bytter mere.

Hvilke sorteringsalgoritmer utmerker seg positivt når n er veldig liten? Og når n er veldig stor?



Når n er liten (0-10), ser vi tydelig at InsertionSort og SelectionSort gjør det bedre enn QuickSort og HeapSort, utenom en uteligger når n = 9. Det er ikke en stor forskjell, vi snakker om en forskjell på 2-5 ms, men som vi ser på grafen så er InsertionSort og SelectionSort faktisk dobbelt så raske (utenom n=9 av en eller annen grunn) når n er lav.



Når n blir høy derimot så ser vi lett at QuickSort og HeapSort gjør det mye bedre enn InsertionSort og SelectionSort.

Hvilke sorteringsalgoritmer utmerker seg positivt for de ulike inputfilene?

QuickSort og HeapSort gjør det ganske bra for alle inputfilene og spesielt de store. InsertionSort og SelectionSort derimot gjør det bedre på de små filene.

Har du noen overraskende funn å rapportere?

Det er nok ikke så veldig overraskende, men ser at InsertionSort er veldig mye raskere på nesten sorterte arrayer enn på helt tilfeldige arrayer.