Oblig 2, IN2010

Severin Schirmer, Simon Halstensen, Iris Bore

September 2021

Oppgave 1

Vi implementerte sorteringsalgoritmene quicksort, insertion sort, heapsort og selection sort. For å sjekke at algoritmene gir rimelige svar legger vi inn en test i runner-fila for å sammenlikne output-fila med output til den forrige algoritmen vi kjørte. Vi kunne også sjekket at hvert element i outputet er større enn det forrige, men siden de produserer samme resultater og vi ser at de er korrekte for små arrays antar vi det stemmer for store. Dette er på linjer 37 til 42 i `oblig3runner.py`.

Oppgave 2

Bytter og sammenligninger målt ble målt med klassene countcompares og countswaps, som jobber i kulissene fordi algoritmene vi implementerer mottar en liste som er en instans av `countswaps` med `countcompares` som elementer. Antar bytter blir lagret i `countswaps` hver gang et bytte blir gjort. Antall sammenligninger inkrementeres for hvert element hver gang vi sammenlikner elementer i arrayet og summeres til slutt. Vi teller derimot ikke sammenligninger mellom ints som gjøres flere steder i algoritmen.

Oppgave 3

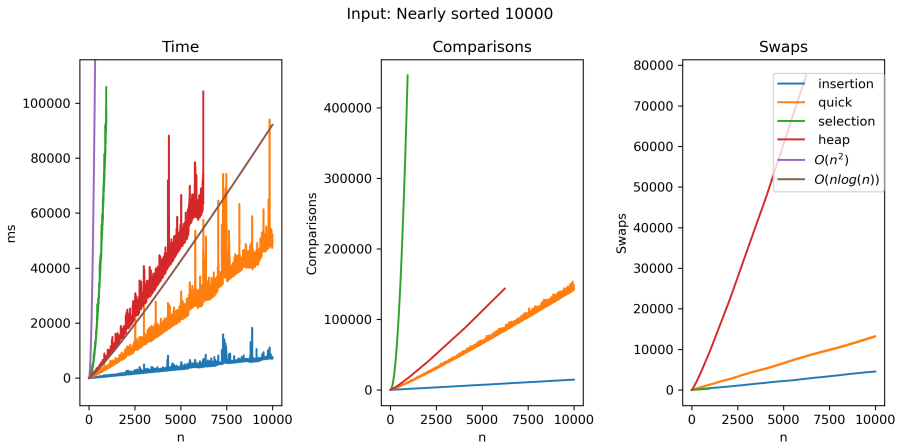

Figure 1: Resultater for nesten sorterte arrays.

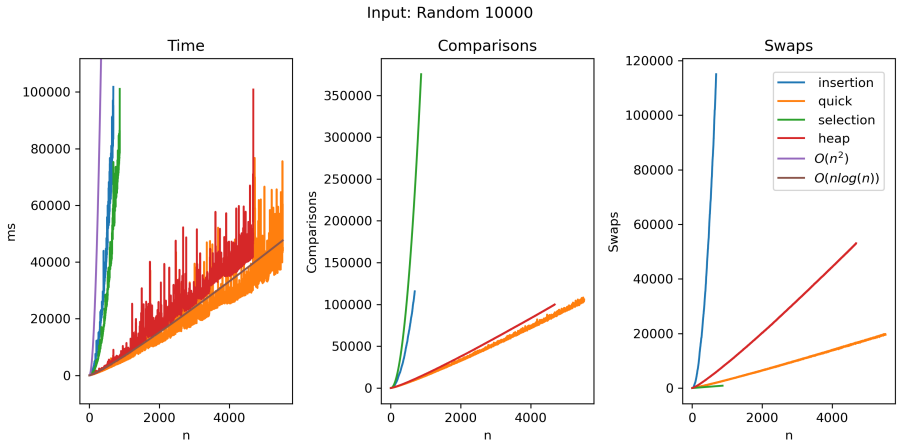

Figure 2: Resultater for tilfeldig sorterte arrays.

Analyse

Vi ser fra figur 2 at kjøretidsanalysen stemmer godt overens med kjøretiden når arrayet er tilfeldig sortert. Men når vi har nesten sorterte arrays utklasser insertion sort de andre og er lineær siden den bare går gjennom arrayet en håndful ganger

Ingen av algoritmene er ren $O(n^2)$ i våre forsøk, men dette er fordi O-notasjonen

Table 1: Kjøretidsanalyser

Algoritme	Kjøretidsanalyse
Insertion	$O\left(n^2\right)$
Selection	$O\left(n^2\right)$
Quicksort	$O\left(n^2\right)$*
Heapsort	$O(n\log(n))$

er worst-case og det er usannsynlig at input-arrayet skal være tilfeldig sortert på minst mulig effektive måte til å starte med. Quicksort er $O(n\log(n))$ i praksis, men $O\left(n^2\right)$ i worst-case. Vi ser fra våre analyser at $n\log(n)$ er en god tilnærming siden den ligger lavere enn den brune linjen i både figur 1 og 2.

Videre kan vi se at noen algoritmer er bytte-intensive og noen er sammenlignings-intensive hvor vi ser at selection sort er spesielt sammenlignings-intensiv hvor den grønne linjen i det andre plotet i hver figur skyter til værs i det som ligner en andregradsfunksjon, derav $O(n^2)$ for algoritmen. Insertion sort har det samme for bytter med tilfeldige sorterte arrays, men for nesten sorterte arrays minimerer selection sort både bytter og sammenligninger og den er derfor effektiv i dette tilfellet. Quicksort og heapsort har en balanse mellom de to hvor en ikke dominerer og totalsummen blir mindre og derfor er de raskere.

For veldig små n ($n < 25$) gjør insertion og selection sort det bedre enn heap og quicksort selv for tilfeldige arrays. Dette er nok fordi det er mer overhead i starten for heap og quicksort som jevnes ut for større n. Overhead består her av pivot elementer, oppdeling, og bubble down som gjøres i disse to.