Innleveringsoppgave 3

Trym H. Nyheim

Redgjør for hvilke sorteringsalgoritmer som er implementer og hva som er testet for å sjekke at alle algoritmene gir rimelige svar:

Sorteringsalgoritmene som er implementert er:

- Insertion sort
- Merge sort
- Heap sort
- Bubble sort

Jeg har testet algoritmene med «random_100.txt» og sjekket at alle algoritmene gir samme output, i tillegg til å se over at de faktisk returnerte en sortert output.

Skriv kort om hvordan bytter og sammenligninger er målt:

Programmet teller bytter i utgangspunktet når to indekser i den usorterte lista bytter plass. Unntaket er merge sort, som ikke utfører bytter på denne måten. Her er bytter telt når elementene flettes sammen fra listene A_1 og A_2 til A.

Sammenligningene telles når to indekser i den usorterte listen sammenliknes. Sammenlikninger for indeksen ved løkker som itererer over listen telles ikke.

Programmet håndterer verdiene for bytter og sammenligninger i statistikk objekter som gis som argumenter til sorteringsfunksjonene. Metoder for inkrementering av tellerne i disse objektene kalles i funksjonene, og i en sammenligningsfunksjon lessThan().

Framgangsmåte:

For å undersøke effektiviteten til algoritmene har jeg testet de med dataene fra:

Filer med randomiserte tallverdier:

- random_10.txt
- random_100.txt
- random_10000.txt
- random_100000.txt

Filer med nesten sorterte tallverdier:

- nearly_random_100.txt
- nearly_random_10000.txt
- nearly_random_100000.txt

For hver fil som er testet genereres det data fra antall sammenlikninger og bytter, i tillegg til kjøretid i nanosekunder. Disse dataene genereres for hver 10-del av størrelsen på input-et. Altså hvis input er en fil med 100 verdier, genererer programmet statistikk for størrelsene av inputdataene, 0, 10, 20, … , 90, 100.

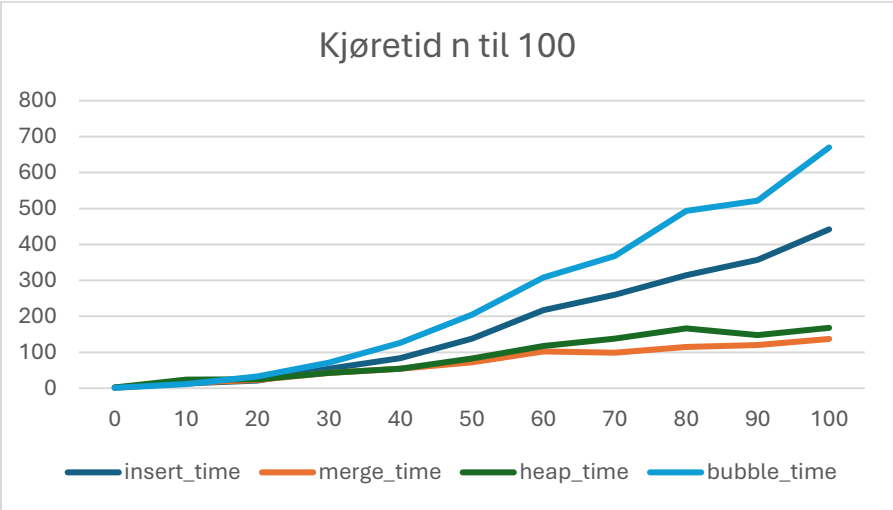
Med disse dataene kan man analyserer hvordan de forskjellige algoritmene oppfører seg når størrelsen på input-et øker, og jeg har brukt de til å svare på spørsmålene som følger.

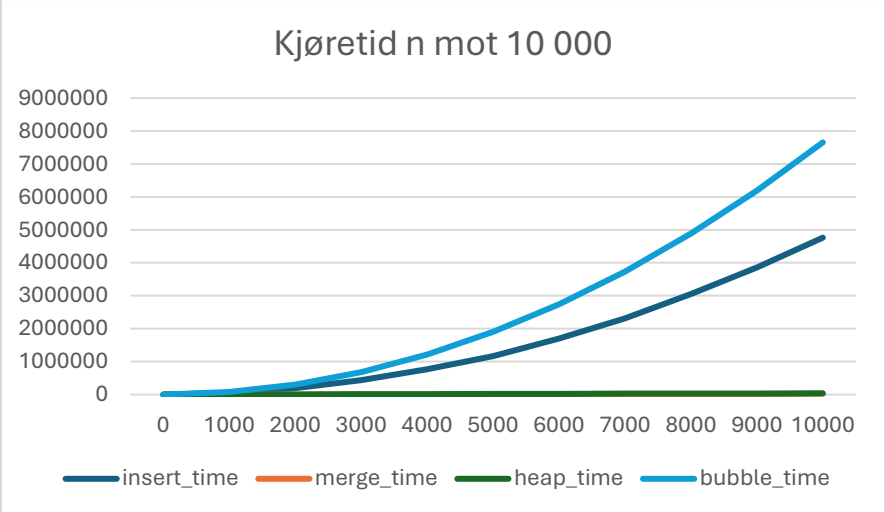
I hvilken grad stemmer kjøretiden overens med kjøretidsanalyse for de ulike algoritmene?

Insertion og bubble sort er i $O(n^2)$, og man kan forvente at kjøretiden deres vokser betydelig raskere enn heap og merge sort, som er i $O(n \log(n))$, ved store verdier av n.

Ved kjøring av testdataene med helt randomisert rekkefølge stemmer dette godt, og vi kan se at heap og merge sort er veldig mye raskere når størrelsen på n går over i 1000-tallet.

Allerede når n er 100, begynner algoritmene i $O(n \log(n))$ å bli flere ganger raskere enn de i $O(n^2)$.





Hvordan er antall sammenligninger og antall bytter korrelert med kjøretiden?

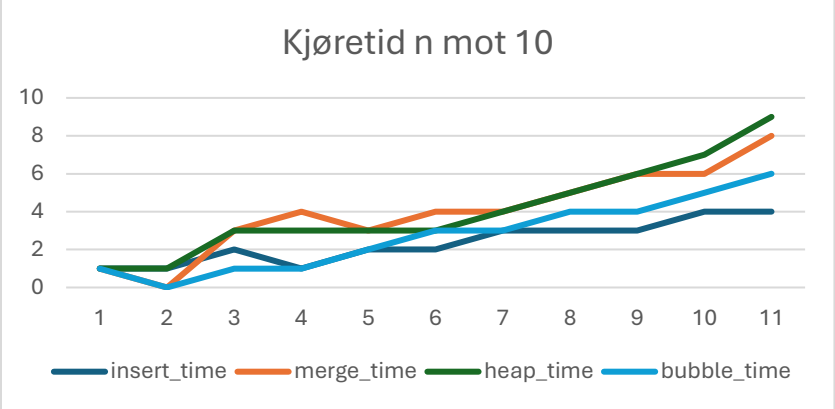
Her er en liste over hvor mange sammenligninger og bytter de forskjellige algoritmene gjorde ved n = 10 000.

	insert	merge	heap	bubble
Sammenlikninger	24935360	120488	235359	49977609
Bytter	24925372	128267	124183	24925372

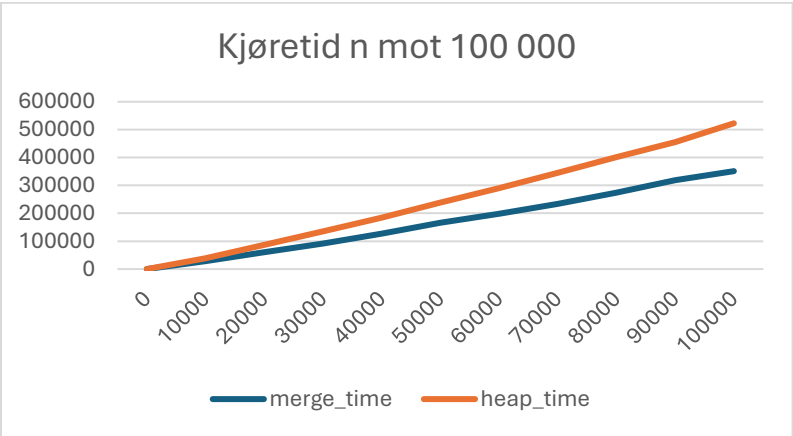
Tabellen viser altså at algoritmene som er i $O(n^2)$ gjør betydelig flere sammenligninger og bytter, og siden disse også hadde veldig mye treigere kjøretid, så det ser ut til at det er en korrelasjon mellom sammenlikning/bytter og kjøretiden.

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

Insertion sort og bubble sort ser ut til å være raskere enn merge sort og heap sort ved små størrelser av n, men dette varer bare til størrelser av n på ca. 20.



Ved store størrelser av n blir insertion sort og bubble sort upraktisk trege, og jeg gjorde derfor ikke analysen på disse ved så store verdier for n. Ved en sammenligning av heap sort og merge sort med n fra 10 000 til 100 000 ser det ut til at merge sort er raskere.



Hvilke sorteringsalgoritmer utmerker seg positivt for de ulike inputfilene?

Et unntak fra den forventede kjøretiden til algoritmene, er at insertion og bubble sort oppnår bedre kjøretid på testdataene som er nesten sorterte. Her gjorde de også veldig mye færre sammenlikninger og bytter.

Heap sort kommer dårlig ut, som jeg antar er fordi den i minst grad klarer å utnytte en nesten sortert input, og at det rett og slett rotes til når verdiene settes på en heap.

Insertion sort kommer best ut, da den bare vil løpe igjennom listen og fikse opp i de siste uryddighetene, og derfor få best utbytte av at listen nesten er sortert.