Innlevering 1 – IN2010

Teque

Mappe: oblig1zip

Fil: Teque.py, Psuedokode_Teque.txt

1c)

Kjøretidskompleksitet for push_back() = O(1)

Kjøretidskompleksitet for push_front() = O(1)

Kjøretidskompleksitet for push_middle() = O(1)

Kjøretidskompleksitet for get / __getitem__() = O(1)

Kjøretidskompleksitet for size / __len__() = O(1)

1d)

Hadde N vært begrenset kunne muligens kjøretiden blitt kortere, men det kommer med andre ulemper. Skalering blir et problem siden en input kan fort bli veldig stor. Det er heller ikke særlig robust hvis den ikke kan håndtere de fleste type input, både stor og ikke bare liten. Ytelse er en ting, men det å eksperimentere med noe som har begrenset N gir lite muligheter og er lite spennende.

Sortering

Mappe: oblig1zip

Fil: insertion.py, merge.py

Sorteringsalgoritmene som er implementert er merge og insertion. Disse algoritmene gjør ting på forskjellige måter og som resultat får ulike resultater.

Merge sort er en sorteringsalgoritme som bruker rekursjon for å sortere en gitt mengde. Den gjør ikke bytter på en helt typisk måte man skulle tenkt seg. Istedenfor splitter den lister opp i mindre sublister og gjør dette helt til det er et element i listen som da skal sammenlignes med en annen liste med et element, dette er basistilfellet der rekursjonen returnerer. På vei opp fra basistilfellet legges elementer på riktig plass til den kommer til toppen og man har en sortert mengde.

Insertion sort er en sorteringsalgoritme med en enkel ide om at man tar hvert element fra den usorterte siden av listen og plasserer det på riktig plass i den sorterte siden av listen mens man itererer seg gjennom. Ideen for algoritmen er grei, men i praksis kan den være veldig treg for sine worst-case inputs.

For å sjekke at disse to algoritmene gir rimelige svar har jeg både sjekket at de faktisk gir riktig output, så et sortert array hvis det er gitt et usortert array. Og hvordan algoritmene håndterer input av forskjellige størrelser.

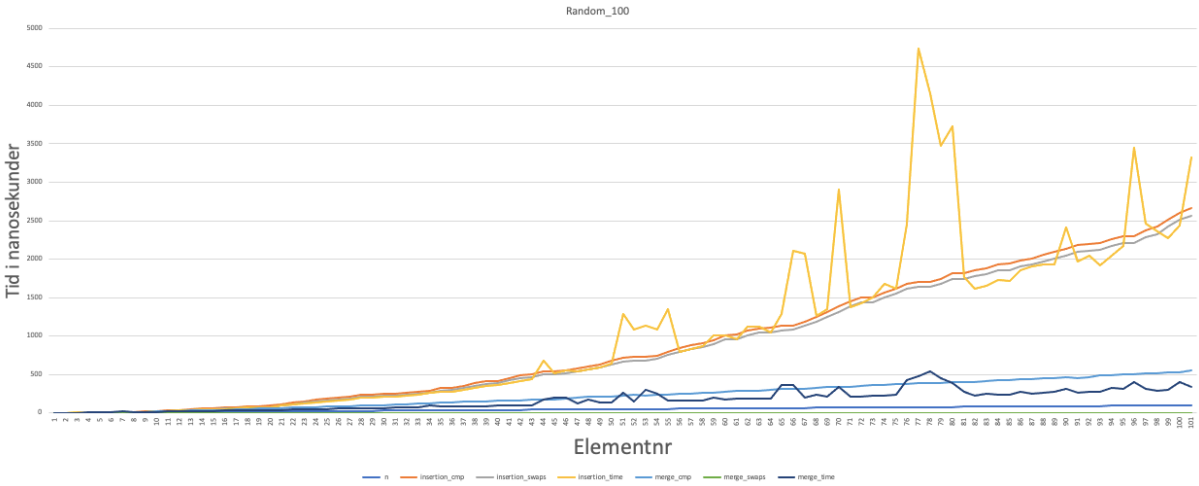
Sammenligninger, bytter og tid

Mappe: Oblig1zip

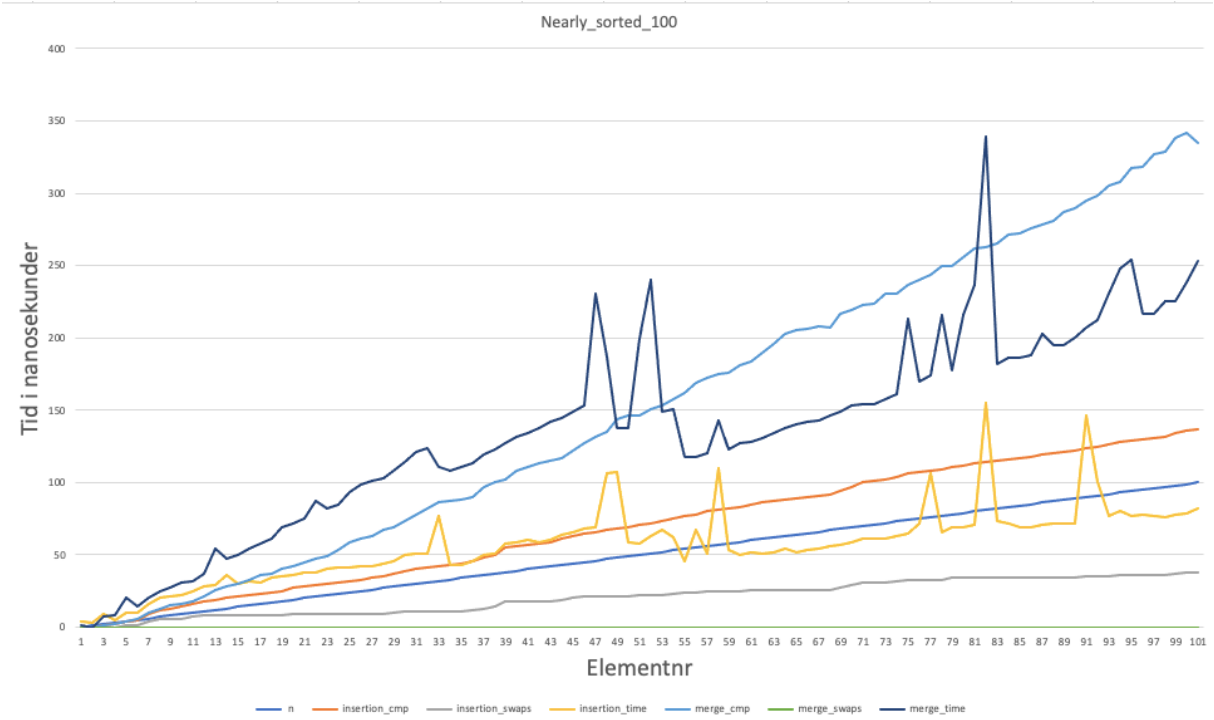
Filer: sort_runner.py, countcompares.py, countswaps.py, main.py

Eksperimentér

Random input med hundre elementer: (Merge: blå) (Insertion: gul)



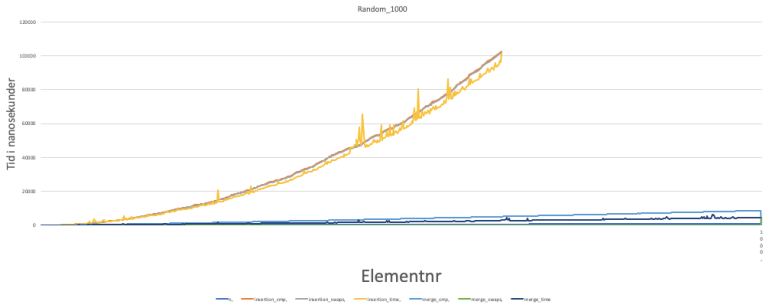
Nesten sortert input med hundre elementer: (Merge: blå) (Insertion: gul)



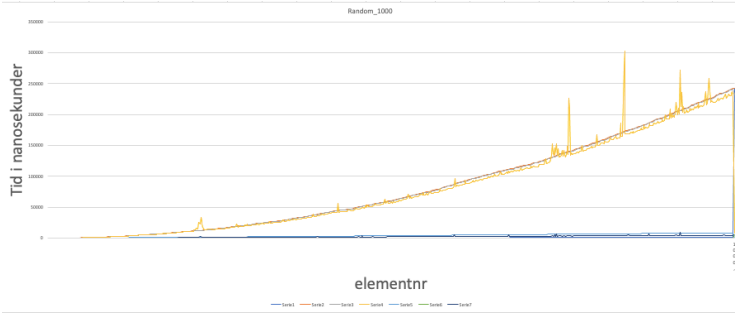
Random input med tusen elementer:

(Merge: blå) (Insertion: gul)

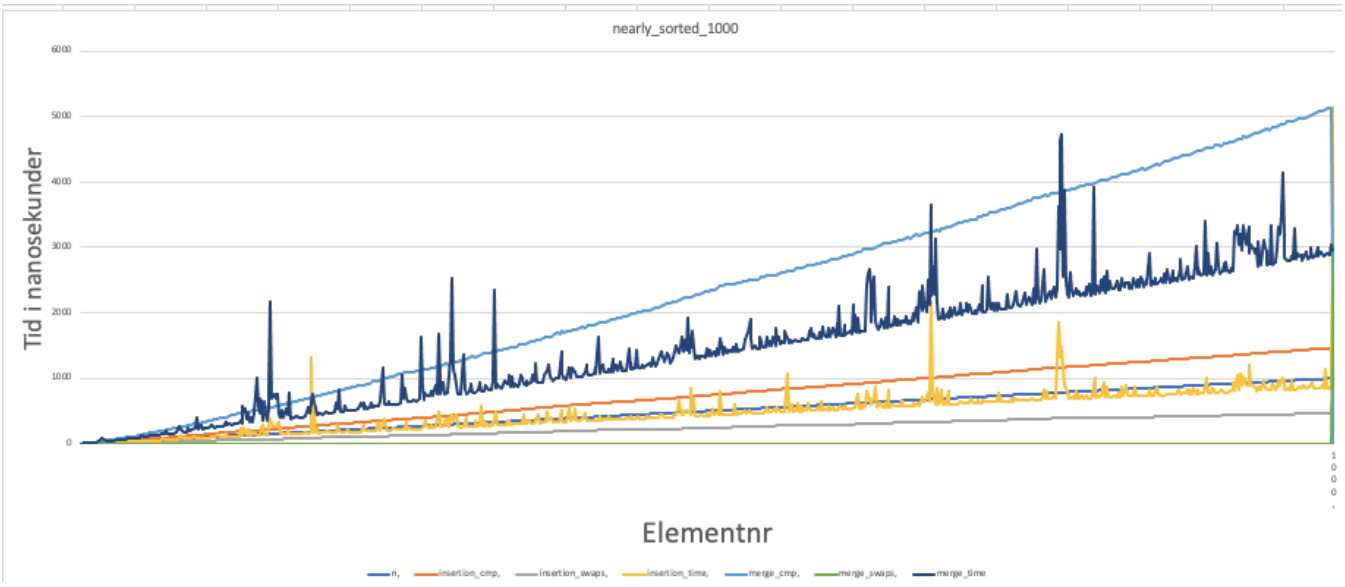
Programmet ga opp på insertion



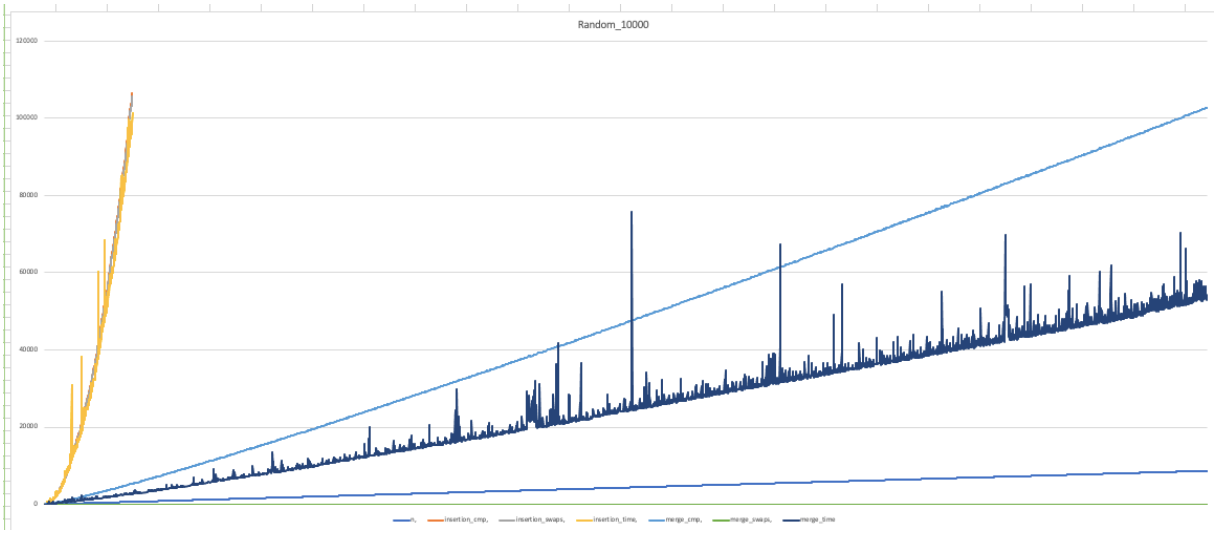
Random input med tusen elementer, men programmet ble tvunget til å ikke gi opp på algoritmen



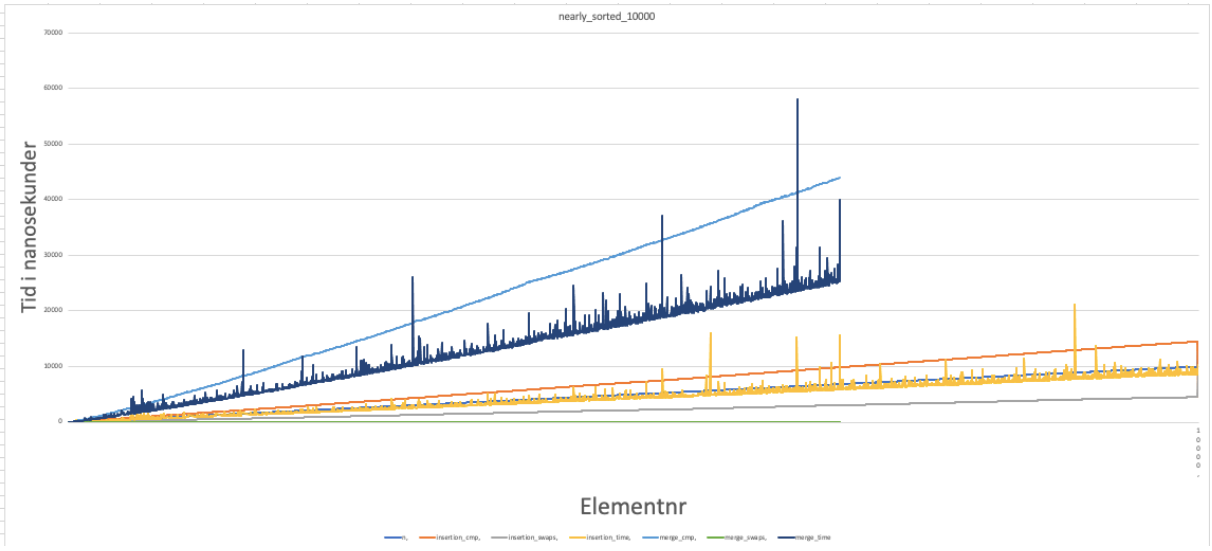
Nesten sortert input med tusen elementer: (Merge: blå) (Insertion: gul)



Random input med ti tusen elementer: (Merge: blå) (Insertion: gul)



Nesten sortert input med ti tusen elementer: (Merge: blå) (Insertion: gul)



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

Kjøretidsanalysene til algoritmene stemte ganske godt overens med hvordan resultatet ble, særlig med random input med ti tusen elementer. Her ser man tydelig at insertion viser en kurve som tilsvarer n^2 siden random input er nesten worst-case for algoritmen. For merge viste den en mer stabil kurve som ser veldig lineær ut.

Hvordan er antall sammenligninger og antall bytter korrelert med kjøretiden?

Generelt er sammenligninger korrelert med kjøretiden i at desto mer sammenligninger desto lengere kjøretid. Hvor man ser at det kan påvirke kraftig er for eksempel med en algoritme der worst-case scenario leder til langt flere sammenligninger. Det kan man blant annet se i insertion, som må iterere mye mer dersom input er lite gunstig.

Antall bytter er på samme måte korrelert med at flere bytter fører til lenger kjøretid. Det som og er dyrt for en algoritme å gjøre er det å flytte på elementer i minnet, som kan påvirke kjøretiden hvis det er snakk om veldig mye data.

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

Av algoritmene jeg har implementert synes jeg begge utmerker seg positivt når N er veldig liten, men mest insertion fordi den er veldig rask hvis input er både best mulig og liten. Merge er den som presterer best på worst-case input.

For når N er veldig stor er det klart at merge presterer best.

Hvilke sorteringsalgoritmer utmerker seg positivt for de ulike inputfilene?

Det som var felles med de forskjellige outputene utenom N størrelse var at merge preserte best med random_n, mens insertion presterte bedre på nearly_sorted_n.

For random_n inputene kan man sammenligne O(n^2) og O(n log n) og se at O(n log n) blir å ta kortere tid. Men når det gjelder nearly_sorted_n, det som skjer er at insertion sin kjøretid blir nesten O(n), som er raskere enn O(n log n). Siden nearly_sorted_n er veldig nært best-case for en algoritme som insertion.

Har du noen overraskende funn å rapportere?

I utgangpunktet har jeg ingen særlig overraskende funn. For meg var det mest spennende funnet hvordan insertion oppførte seg desto større input ble. Hvis man ser på hvordan det forventes at kjøretiden vokser fra kjøretidsanalysen burde man ikke bli overrasket, men å få det visualisert med en så kraftig kurve var jeg ikke forberedt på.