Gard Sveipe Bahmanyar
IN2010
Oblig 3

Oblig 3 IN2010

Sorteringsalgoritmer brukt:

- **MergeSort: O(n log n)**
- **InsertionSort: BestCase O(n), WorstCase O(n^2)**
- **QuickSort: O(n^2)**
- **BubbleSort: O(n^2)**

Oppgave 1:

Hvordan testet jeg at algoritmene ga rimelige svar?:

Sjekket out-filene til de ulike sortertingalgortimene og så at alle ga den samme outputten.

Oppgave 2:

Kort om hvordan cmp og swaps har blitt telt:

Lagde egne funksjoner i sorter, swapped() og compared(). Disse foretar seg av operasjonene: cmp++ og swaps++.

Cmp:

For hver if-sjekk som slår til kalles det på compared().

Swaps:

For hvert bytte av to indekser i array, kalles det på swaps(). Eks: Dersom A[1] og A[2] bytter plass, kalles swaps().

Oppgave 3:

I denne oppgaven kommer jeg til å referere til grafer laget av resultats-dataene vi fikk av sorting-algoritmene. Kommer til å se på følgende input-filer:

-random_10000

-random_100

-nearly_sorted_10000

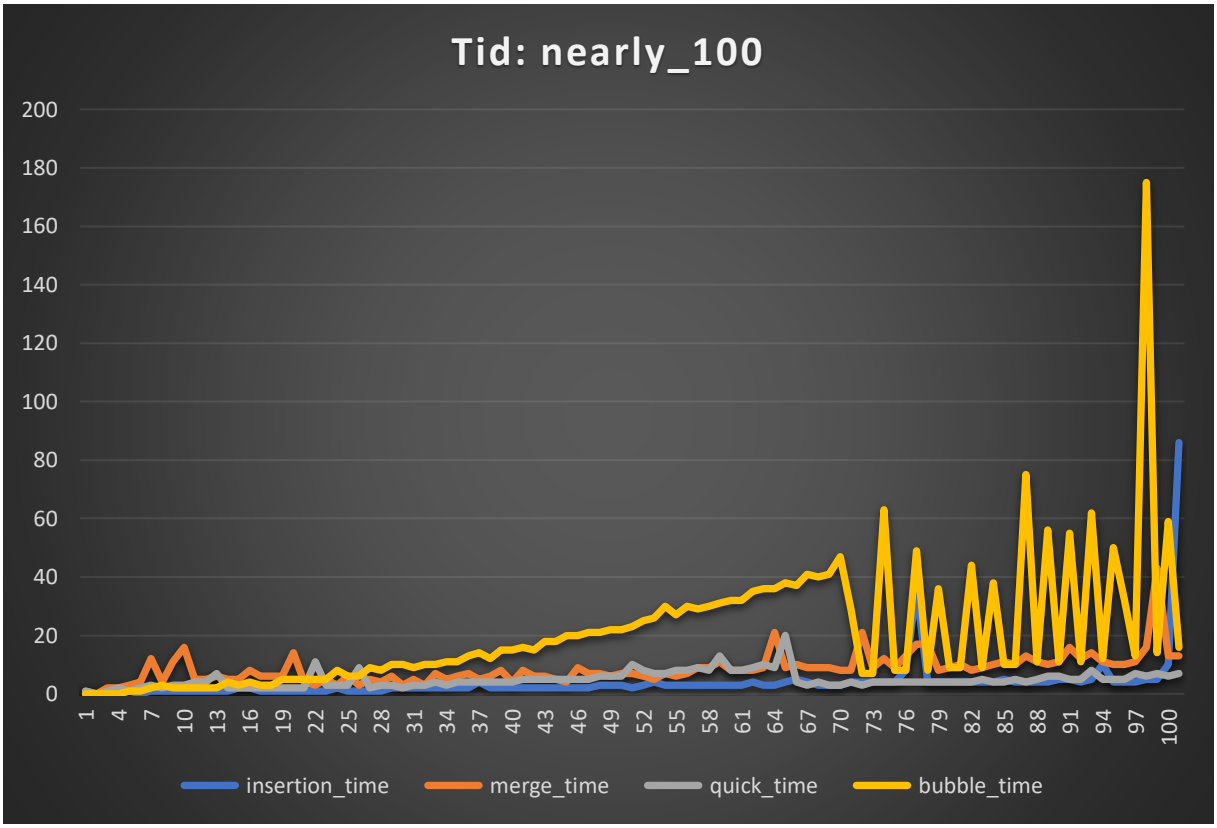
-nearly_sorted_100

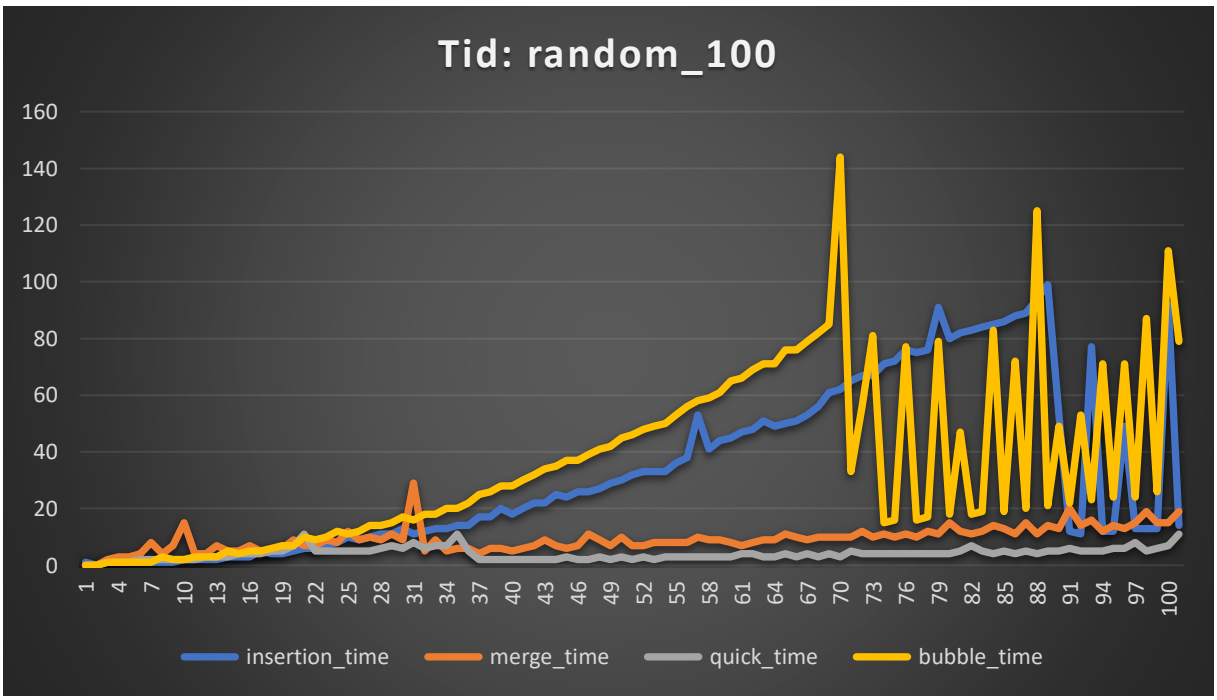
Forklaring på valg av input-filer:

Input-filene på 10000 gir oss en god oversikt over hvordan algoritmene fungerer på store data-sett. Input-filene på 100 gir oss en god oversikt over hvordan algoritmene fungerer på små data-sett. Grunnen til at jeg velger å se bort ifra inputfiler som er mindre enn 100 og større enn 10000 er to grunner. Input-filer på under 100 gir oss et dårlig estimat på algoritmenes effektivitet. Mens de på over 10000 bruker for lang tid å prosessere. I tillegg ser jeg på både random og nearly_sorted nettopp fordi effektiviteten til en algoritme kan ha med å gjøre hvor sortert listen vi skal sortere er fra før.

Gard Sveipe Bahmanyar
IN2010
Oblig 3

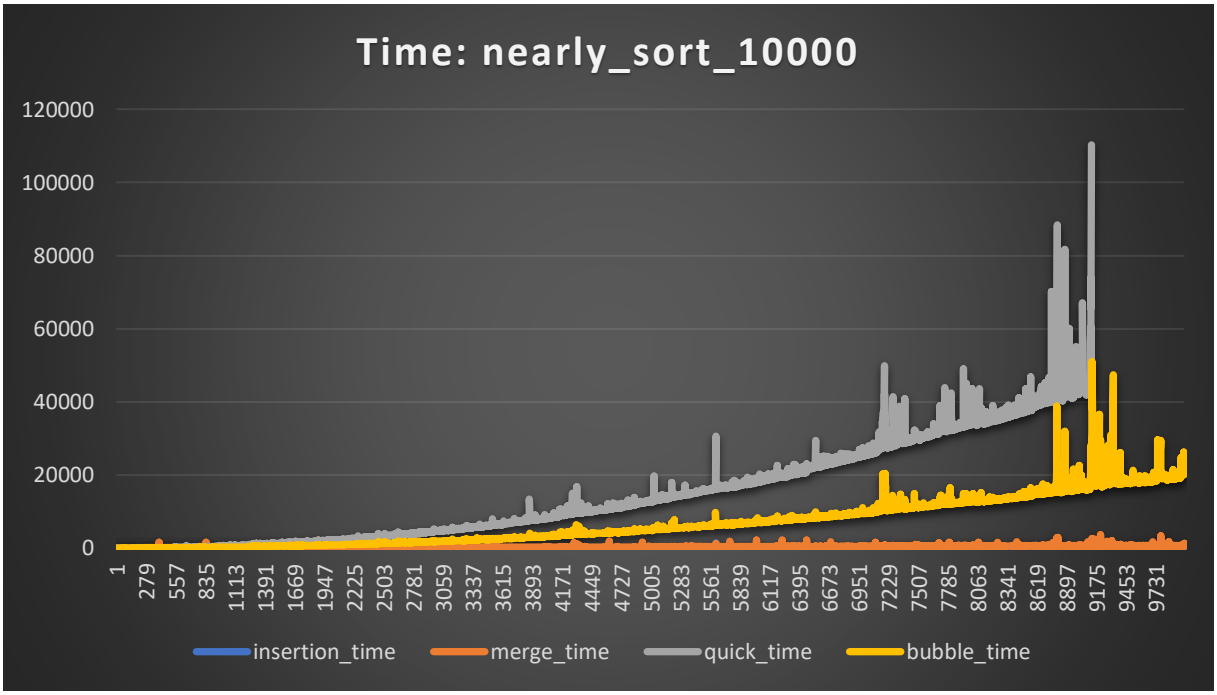
Resultat av 100-filene (tid):



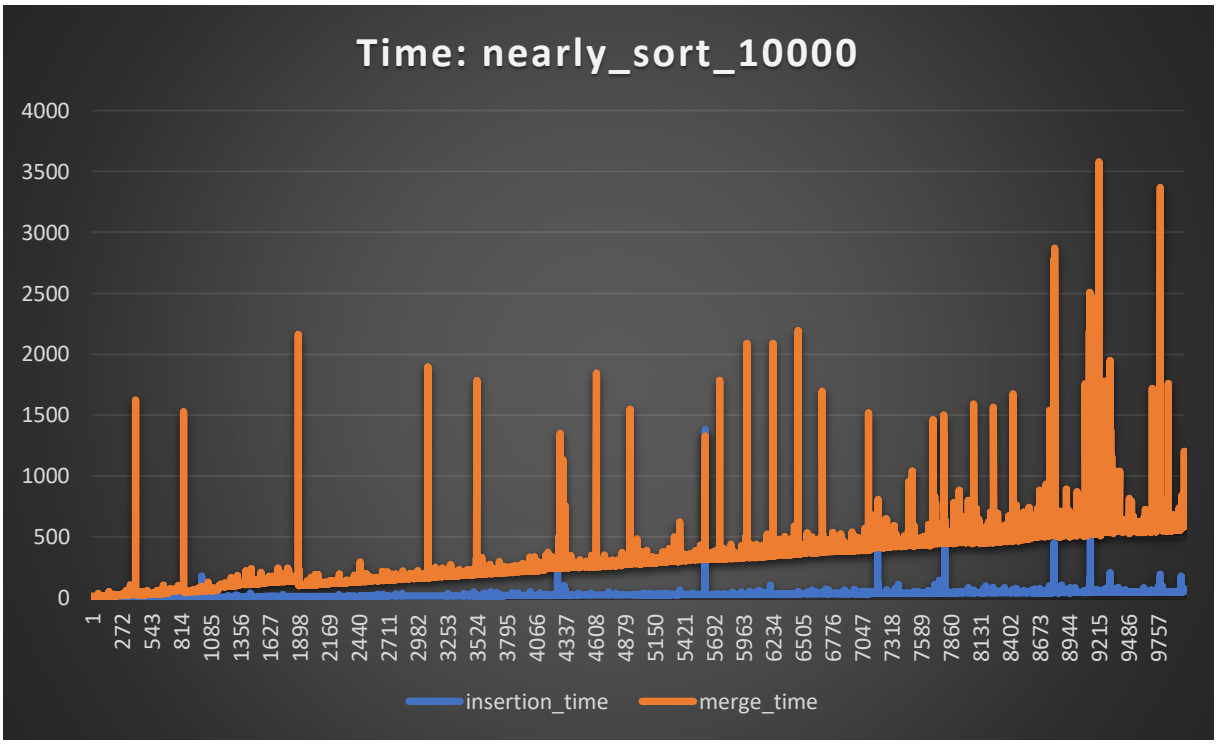


Gard Sveipe Bahmanyar
IN2010
Oblig 3

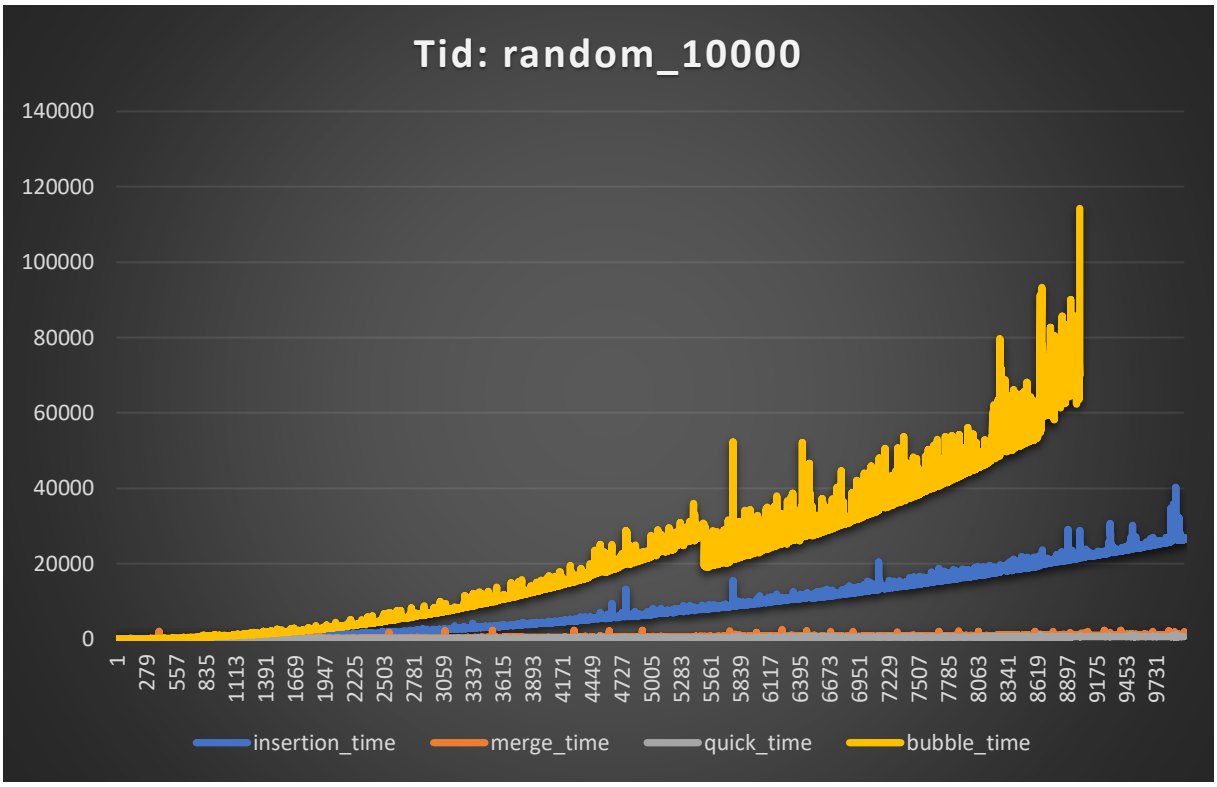
Resultat av 10000-filene:



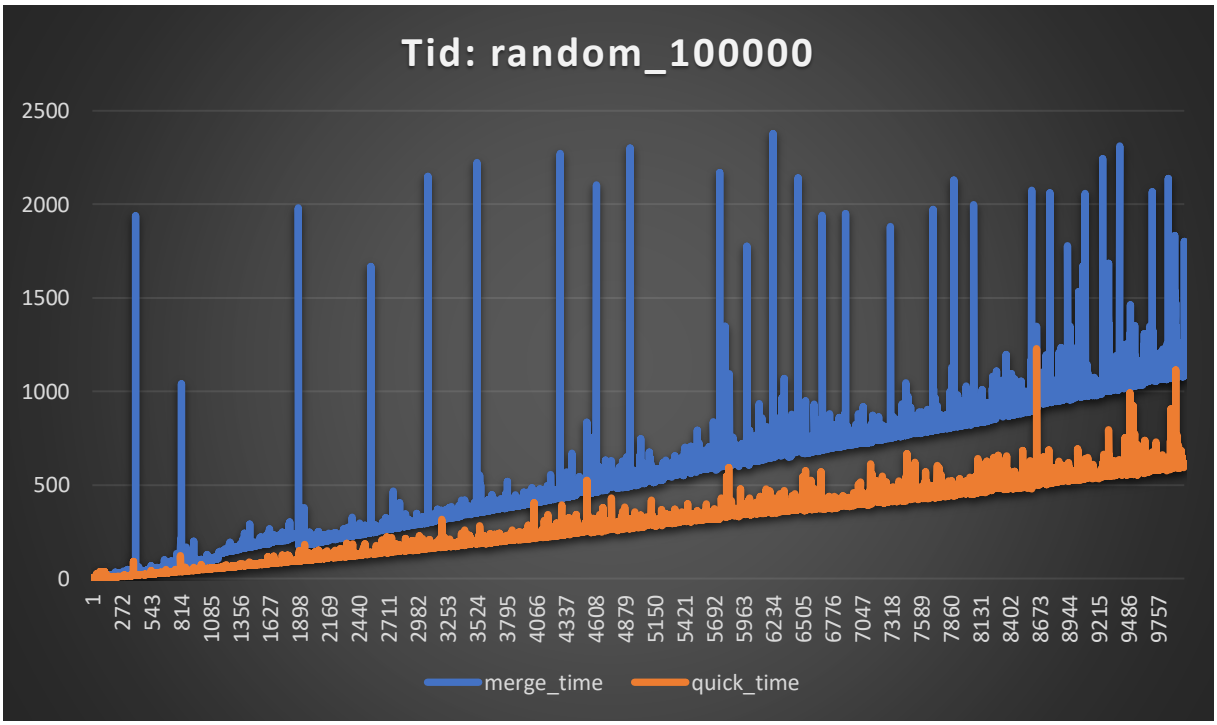
Ser ut til at bubble og quick dominerer. Ser derfor nærmere på nearly_sorted_10000:



Gard Sveipe Bahmanyar
IN2010
Oblig 3

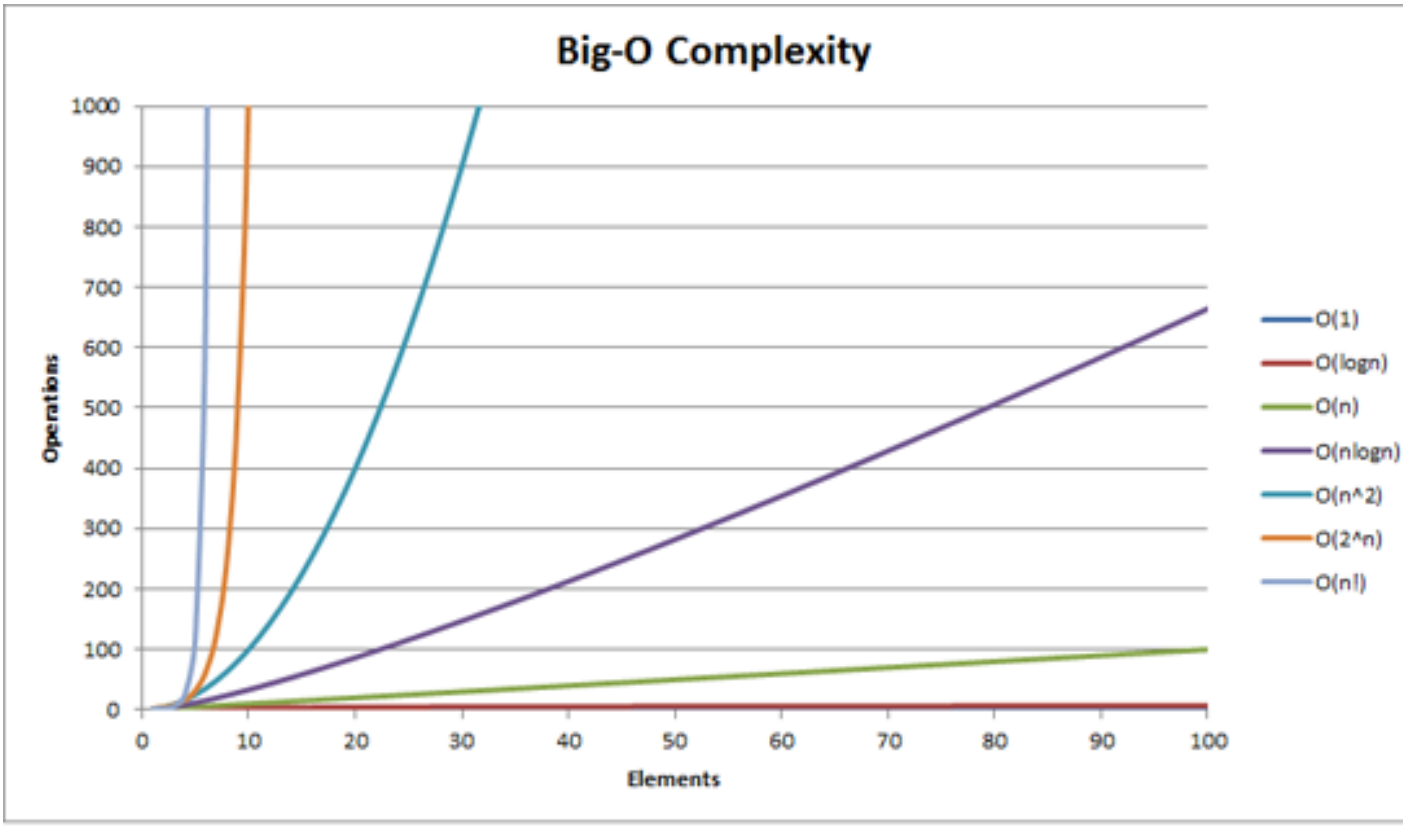


Ser ut til at grafene går inn i hverandre. Ser derfor nærmere på random_10000:



Gard Sveipe Bahmanyar
IN2010
Oblig 3

I hvilken grad stemmer kjøretiden overens med kjøretidsanalysene (store O) for de ulike algoritmene?:



For kjøretidsanalysene (store O) velger vi å se på dataene for de store input-filene (10000). Merk at det har mye å si på graden av «sorterthet» arrayene har fra før vi sorterer. Velger bare å se på de tildelte inputfilene. Vi har igjen:

Sorteringsalgoritmer brukt:

- **MergeSort: O(n log n):**
  Vi ser at formen n er tilnærmet lik den formen vi får på random_10000 og nearly_sorted_10000. Dermed stemmer mergesort big-O dårlig overens med sin big-O.

- **InsertionSort: BestCase O(n), WorstCase O(n^2):**
  Formen på insertionSort på nearly_sorted_10000 ligner log n, mens på random ligner den n log n. Insertion stemmer dermed også dårlig overens med kjøretidsanalysen.

- **QuickSort: O(n^2):**
  Formen på quickSort på nearly_sorted_10000 minner om n^2, noe som stemmer godt overens med big-O. Kjøretiden på random_10000 ligner på n, som ikke stemmer godt overens med big-O.

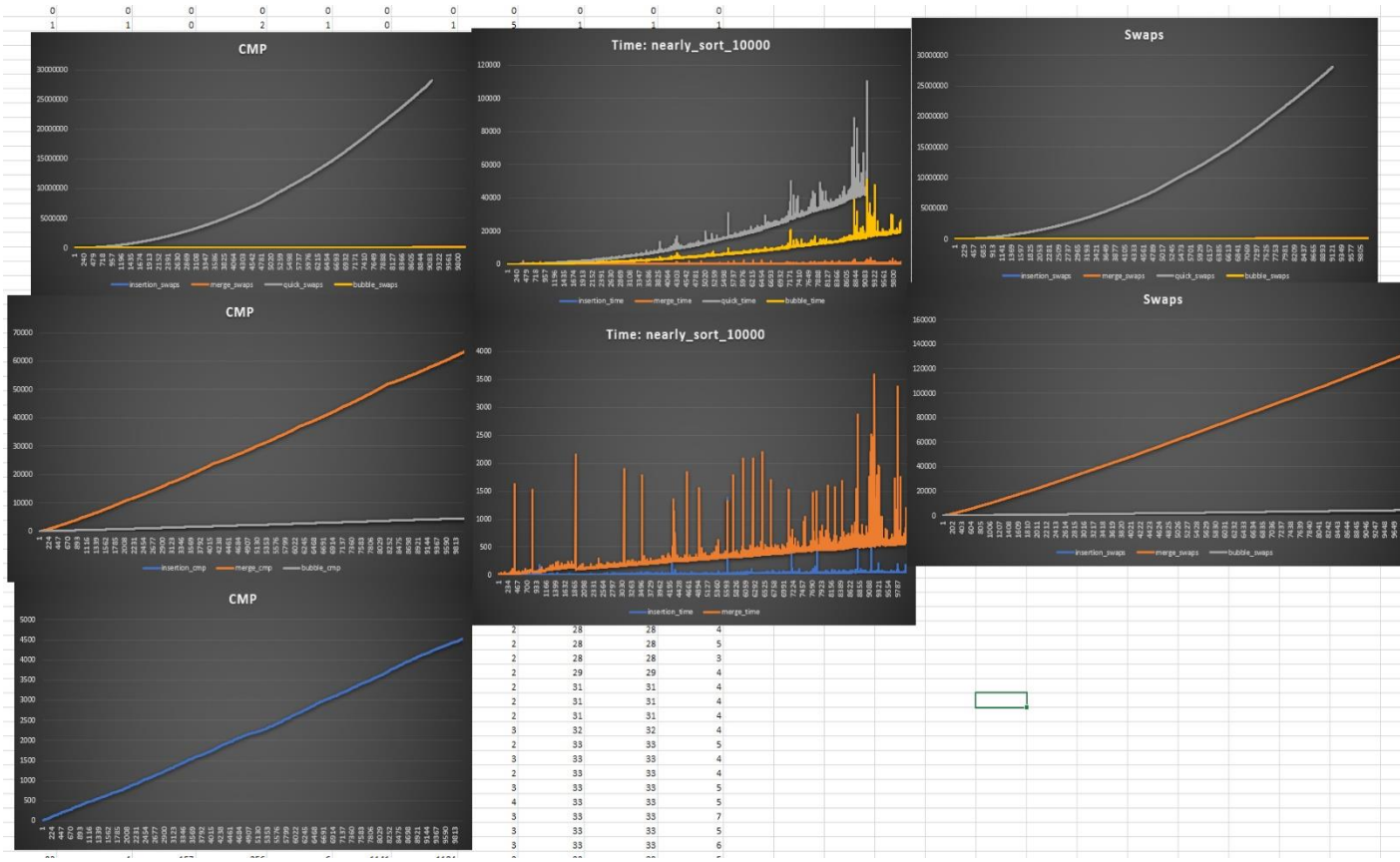
- **BubbleSort: O(n^2):**

Gard Sveipe Bahmanyar
IN2010
Oblig 3

BubbleSort sin kjøretid på nearly_sorted_10000 er på n, mens på random_10000 er den på n^2, altså stemmer kjøretiden på random_10000 godt overens med Big-O.

<u>Hvordan er antall sammenligninger og antall bytter korrelert med kjøre-tiden?:</u>

Nearly_sorted_10000:

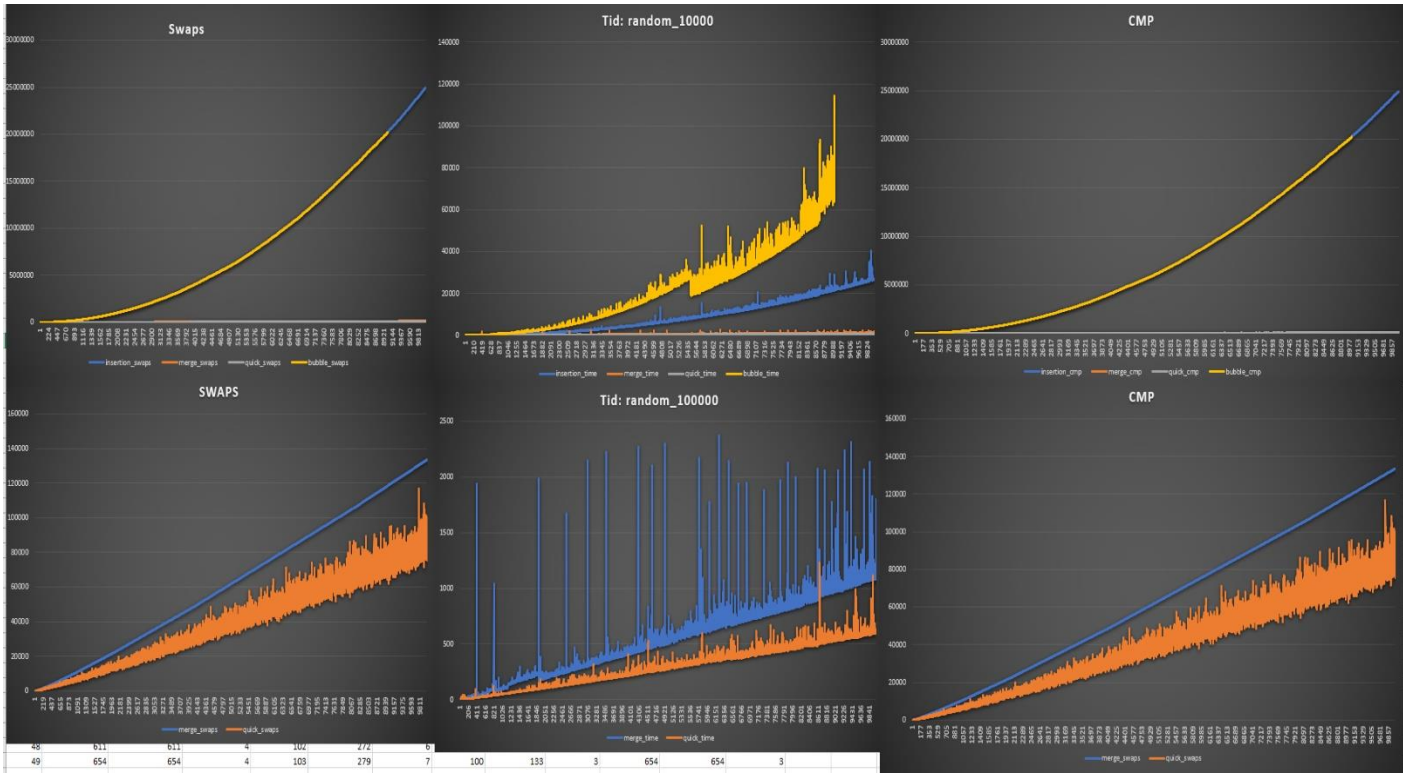


Korrelasjon mellom cmp, swaps og tid:

Sterk korrelasjon mellom cmp, swaps og time. Dette ser vi på formen på alle grafene. Kjøretiden til alle algoritmene gjenspeiler seg i cmp- og swapsgrafene.

Gard Sveipe Bahmanyar
IN2010
Oblig 3

random_1000:



Korrelasjon mellom cmp, swaps og tid:

Sterk korrelasjon mellom cmp, swaps og time. Dette ser vi på formen på alle grafene. Kjøretiden til quick, bubble og mergeSort gjenspeiler seg i kjøretiden. Der det er minst korrelasjon er på insertionSort som har sterkere korrelasjon på cmp og swaps med bubbles-kjøretid enn sin egen.

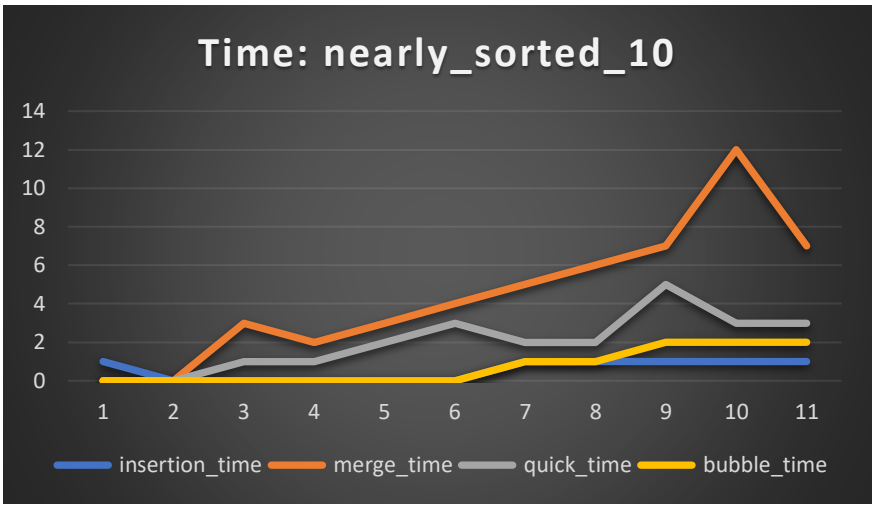
Konklusjon om korrelasjon:

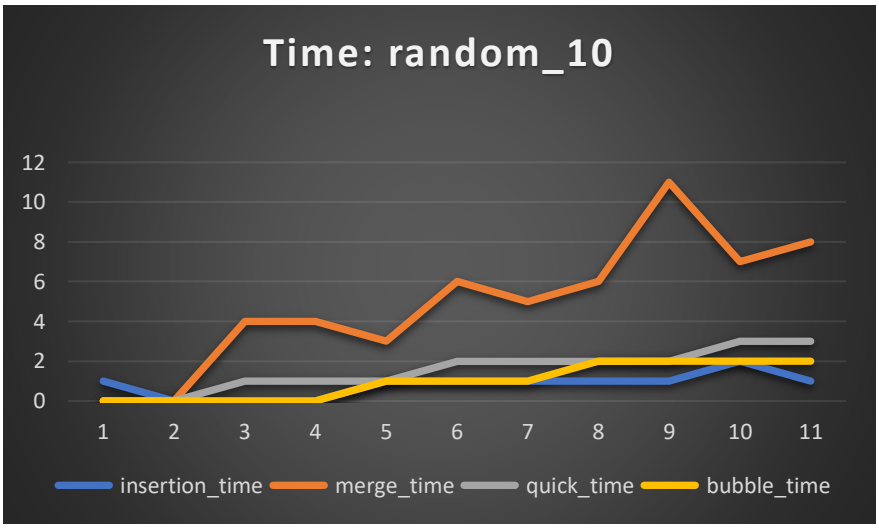
Det er sterk korrelasjon mellom cmp, swaps og time på disse algoritmene. En forklaring på dette er at operasjonene som tar tid utenom itereringen er sammenligninger i «if-tester» og bytte mellom elementer i arrayene, altså cmp og swaps. Spikes i kjøretid kan komme av vanskelige operasjoner eller av andre variabler som er vanskelig å påpeke. Alt i alt gir kjøretiden en form som ligner både cmp- og swapsgrafene, bortsett fra random_10000 som gir annen form på cmp og swaps enn dens kjøretid.

Gard Sveipe Bahmanyar
IN2010
Oblig 3

Hvilke sorteringsalgoritmer utmerker seg positivt når n er veldig liten? Og når n er veldig stor? Hvilke sorteringsalgoritmer utmerker seg positivt for de ulike inputfilene?:

(legger til resultat av 10-inputsfilene):





Algoritmer som utmerker seg på veldig liten n (ser på inputfiler med 10 elementer)

- På random_10 utmerker bubble og insertion seg.
- På nearly_sorted_10 utmerker også bubble og insertion seg.

Algoritmer som utmerker seg på liten n (ser på inputfiler med 100 elementer):

- På random_100 utmerket quick og merg seg mest.
- På nearly_sorted_100 utmerket quick, merge og insertion seg.

Algoritmer som utmerker seg på stor n (ser på inputfiler med 10000 elementer):

- På random_10000 utmerket merge- og quickSort seg.
- På nearly_10000 utmerket merge- og insertionSort seg veldig her også, men insertion var hakket bedre.

Gard Sveipe Bahmanyar
IN2010
Oblig 3

Konklusjon og tanker rundt eksprimentene:

- Spikes:
  Algoritmene spiker i tid på utrolig mange steder, noe som overrasket meg ettersom jeg trodde grafene skulle bli glattere.

- Er 2 datasett på 10000 elementer godt nok estimat?:
  Spørsmålet en kan stille seg når vi får at f.eks. mergeSort og InsertionSort stemmer dårlig overens med dens kjøretidsanalyse (big-O) kan komme av at flere grunner. To av dem er følgende: vi har 2 datasett vi ser på dataene fra. En nærme-sortert og en random. Dette er ganske lite data å trekke sterke konklusjoner med, ettersom vi kunne fått helt andre målinger på 2 datasett av samme type med samme antall elementer. Den andre grunnen kan komme av at antall elementer er for få. Dersom vi hadde hatt enda flere elementer ville vi fått et klarere bilde. I tillegg til fler datasett av samme type, som nevnt over, ville vi fått en enda bedre tanke om nettopp hvilken kjøretid sorteringsalgoritmene faktisk har.

- Konklusjon over de testede algoritmene. Vi har igjen:

Sorteringsalgoritmer brukt:

- **MergeSort:**
  MergeSort er et godt alternativ på arrays med 100+ elementer, uansett helt randome elementer eller nær-sorterte.

- **InsertionSort:**
  InsertionSort er et godt alternativ på arrays som er nær-sorterte uansett størrelse. Dette gir kort kjøretid.

- **QuickSort:**
  QuickSort passer bra på middels-store arrays i dette tilfellet; nær-sortert eller helt random.

- **BubbleSort:**
  BubbleSort er et godt alternativ for rask sorting på veldig små arrays.

- Avsluttende ord:
  Som nevnt er ikke dette nok data, heller ikke et godt nok estimat på akkurat hvor effektive disse algoritmene er på de ulike oppgavene. Det finnes også fler implementasjoner av de ulike algoritmene, som f.eks. annet valg av pivot i quickSort. I denne oppgaven har jeg sett på effektiviteten til MINE versjoner av sortingAlgortimene, og min tolkning av bruken av compared og swapped.

Gard Sveipe Bahmanyar
IN2010
Oblig 3