Oblig 2 Svar

Group ?

February 18, 2025

Oppgave 1

Test Metodikk

Vi implementerte alle 4 algoritmene for å så teste de i main metoden i U7O1/Oppgave1.java. Vi fikk tiden ved å bruke System.nanoTime() ved kver metodekall.

Resultater

Her er et eksempel på et resultat vi fikk når vi testet for $100,000$ elementer:

- Vanlig InsertionSort: $9,828$ms
- MinFirst InsertionSort: $9,406$ms
- TwoAtATime InsertionSort: $7,959$ms
- Combined InsertionSort: $7,026$ms

Det er viktig å nevne at vi fikk enorm forskjell i resultat fra forsøk til forsøk, noe som kan tyde på en feil i metodikken vår. Det var et tilfelle der MinFirst metoden brukte $2x$ så mye tid som alle andre algoritmer. Ved veldig småe arrays var det vanligst at den vanlige InsertionSort-en gjorde det best, noe som gir mening med tanke på at det er den som gjør minst operasjoner. Vi observerer tydelig hvordan InsertionSort ikke er den beste algoritmen for store mengder data ettersom ved $100,000$ elementer tar det gjerne over 10s å sortere listen i vårt test-miljø.

Oppgave 2

a)

Insertion Sort

N	Antall målinger	Målt tid (gjennomsnitt)	Teoretisk tid $c \cdot f(n)$
$32,000$	3	$428,923,900ns \approx 428ms$	$428ms$
$64,000$	3	$3,514,316,013ns \approx 3,514ms$	$1,713ms$
$128,000$	3	$9,623,484,732ns \approx 9,623ms$	$6,853ms$

Table 1: Insertion Sort

For å finne $c \cdot f(n)$ antar vi $T(n) = c \cdot n^2$ som betyr $c = T(n) \div n^2$.
Gitt $T(n) = 428ms$ har vi $c = \dfrac{428ms}{(32,000)^2}$ som vi kan bruke til
å finne teoretisk tid for andre verdier av N.

Selection Sort

N	Antall målinger	Målt tid (gjennomsnitt)	Teoretisk tid $c \cdot f(n)$
$32,000$	3	$380,996,609ns \approx 380ms$	$380ms$
$64,000$	3	$1,221,438,068ns \approx 1,221ms$	$1,517ms$
$128,000$	3	$7,192,890,455ns \approx 7,192ms$	$6,070ms$

Table 2: Selection Sort

$$380ms = c \cdot (32,000)^2 \leftrightarrow c = \frac{380ms}{(32,000)^2} \approx 3.71 \cdot 10^{-7}ms$$

Quick Sort

N	Antall målinger	Målt tid (gjennomsnitt)	Teoretisk tid $c \cdot f(n)$
$32,000$	1000	$2,400,351ns \approx 2ms$	$2ms$
$64,000$	1000	$5,200,779ns \approx 5ms$	$4.27ms$
$128,000$	1000	$15,935,803ns \approx 16ms$	$9.07ms$

Table 3: Quick Sort

$O(n) = n \cdot \log n \rightarrow 2ms = c \cdot (32,000 \cdot \log_2 32,200) \leftrightarrow c \approx \dfrac{1ms}{240,000}$
$\approx 4.17 \cdot 10^{-6}ms$
Vi bruker $\log_2$ siden vi deler array opp $\dfrac{n}{2} \rightarrow \dfrac{n}{4} \rightarrow \dfrac{n}{8} \rightarrow ....$
Vi får $\log_2 n$ "nivåer i treet" og ved hvert nivå gjøres det
en mengde arbeid proporsjonellt til n, så vi sier $f(n) = n \cdot \log_2 n$.
Det samme gjelder for `Merge Sort`.

Merge Sort

N	Antall målinger	Målt tid (gjennomsnitt)	Teoretisk tid $c \cdot f(n)$
$32,000$	500	$4,005,655ns \approx 4ms$	$4ms$
$64,000$	500	$8,592,282ns \approx 8ms$	$8.53ms$
$128,000$	500	$24,233,961ns \approx 24ms$	$17.70ms$

Table 4: Merge Sort

$$c \approx \frac{2}{240,000} \approx 8.33 \cdot 10^{-6}$$

b)

Et array med $10,000$ identiske elementer tar rundt $114ms$ i gjennomsnitt. Dette er fordi alle elementer i array-et er lik pivot-en. Dette fører til at Quick Sort ikke kan partisjonere problemet effektivt, og tidskompleksiteten blir nærmere $O(n) = n^2$.