Analyse av algoritmers effektivitet i O-notasjon

Birk Tangen, Gunnar Salbu, Glenn Boine, Thobias Høivik

31.01.25

Oppgave 3a

Vi analyserer vekstfunksjonene og uttrykker dem i O-notasjon:

- $4n^2 + 50n - 10 \quad \Rightarrow O(n^2)$
- $10n + 4\log_2 n + 30 \quad \Rightarrow O(n)$
- $13n^3 + 22n^2 + 50n + 20 \quad \Rightarrow O(n^3)$
- $35 + 13\log_2 n \quad \Rightarrow O(\log n)$

Oppgave 3b

Gitt algoritmen:

```
sum = 0;
for (int i = n; i > 1; i = i / 2) {
    sum = sum + i;
}
```

Løkken starter med $i = n$ og halveres for hver iterasjon. Antall ganger løkken kjører er $O(\log n)$.

Antall tilordninger:

- Initialisering av sum: 1
- Initialisering av i: 1
- i er satt til ny verdi ca. $\log_2 n$ ganger
- Tilordning i løkken: $O(\log_2 n)$ ganger

Total kompleksitet: $O(\log n)$.

Oppgave 3c

Gitt algoritmen:

```
sum = 0;
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= n; j = j * 2) {
        sum += i * j;
    }
}
```

Den ytre løkken kjører $O(n)$ ganger, og den indre løkken kjører $O(\log n)$ ganger per iterasjon av den ytre løkken.

Totalt antall tilordninger: $O(n \log n)$,

- Initialisering av sum: 1
- Initialisering av i: 1
- Initialisering av j: 1
- $i = i + 1$ n ganger
- $j = j * 2$ rundt $\log_2 n$ ganger

Tidskompleksitet: $O(n \log n)$.

Oppgave 3d

For en sirkel med radius r:

- Areal: $A = 2\pi r^2 \Rightarrow O(r^2)$
- Omkrets: $C = 2\pi r \Rightarrow O(r)$

Oppgave 3e

Gitt metoden:

```
boolean harDuplikat(int tabell[], int n) {
for (int indeks = 0; indeks <= n - 2; indeks++) {
    for (int igjen = indeks + 1; igjen <= n - 1; igjen++) {
        if (tabell[indeks] == tabell[igjen]){
            return true;
        }
    }
}
return false;
}
```

Den ytre løkken kjører $n-1$ ganger, og den indre løkken kjører $O(n)$ ganger per iterasjon av den ytre løkken. Tidskompleksitet: $O(n^2)$. Viss vi ser på den yterste løkken er det rent teknisk $n-1$ sammenligninger når betingelsen til løkken sjekkes. Den indre løkken kjører da $S = \sum_{i=0}^{n-2}(n-1-i) = \frac{n^2-n}{2}$.

Som betyr at det er i verste tilfeller $2*S+n-1 = n^2-1$ sammenligninger når vi teller med betingelse-sjekkene i kver løkke og sammenligningen i den indre løkken.

Oppgave 3f

Vi analyserer de gitte funksjonene:

- $t_1(n) = 8n + 4n^3 \Rightarrow O(n^3)$
- $t_2(n) = 10\log_2 n + 20 \Rightarrow O(\log n)$
- $t_3(n) = 20n + 2n\log_2 n + 11 \Rightarrow O(n\log n)$
- $t_4(n) = 4\log_2 n + 2n \Rightarrow O(n)$

Rangering av effektivitet fra best til verst: $t_2 : \log n < t_4 : n < t_4 : n\log n < t_1 : n^3$

Oppgave 3g

Analyse av tid()-metoden **Kode:**

```
public static void tid(long n) {
    long k = 0;
    for (long i = 1; i <= n; i++) {
        k = k + 5;
    }
}
```

Metoden har en enkel løkke som itererer n ganger og utfører en konstant operasjon per iterasjon. Når vi måler tid for $n = 10^7, 10^8, 10^9$, vil vi observere at kjøretiden øker proporsjonalt med n. Variasjoner kan oppstå på grunn av systemets belastning, CPU-cache, og andre prosesser, men i snitt vil tiden være proporsjonal med n, som bekrefter $O(n)$.