

Oppgave 2 Effektivitetsanalyse

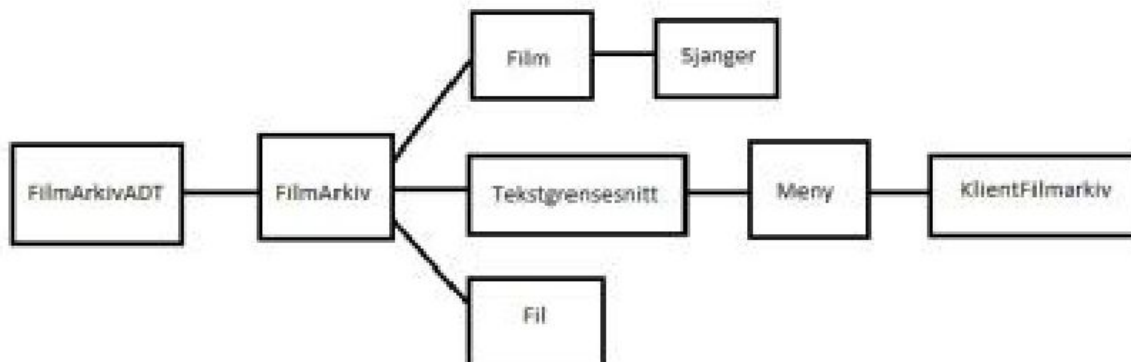
Rapport

Gruppearbeidet har vært utfordrende på grunn av situasjonen med covid som har gjort at vi ikke fikk jobbe så mye fysisk sammen som vi hadde håpet. Vi hadde også et gruppemedlem som droppet ut av kurset. Vi har gjort det beste ut av situasjonen og benyttet oss av Discord for å kunne dele skjerm, sende kode osv.

Vi brukte lang tid på å ta oppgaven inn over oss da det var veldig mye tekst og ganske uoversiktlig, men til slutt kom vi igang.

For best mulig læringsutbytte begynte vi hver for oss slik at alle fikk prøvd seg frem med å lage egne klasser, interface, tekstgrensesnitt osv. Det var masse nytt som vi ikke har hatt om tidligere og dermed tok dette veldig mye tid.

Det som var mest utfordrende med selve kodingen var å få til skriving til og fra fil, og å utvikle ment-klassen helt selv uten å ha jobbet med dette tidligere. Dette tok også enormt mye tid. Vi satt sammen som gruppe og løste oppgave 2 fra øving2 som var ganske utfordrende for oss, men det gikk seg til etter hvert.



Oppgave 2 a)

Hva er størrelsesorden uttrykt i O-notasjon (dvs. vi behøver ikke finne c og N) for algoritmen når vekstfunksjonen er gitt som:

i) $4n^2 + 50n - 10 = O(n^2)$

ii) $10n + 4 \log_2 n + 30 = O(n)$

iii) $13n^3 - 22n^2 + 50n + 20 = O(n^3)$

iv) $35 + 13 \log_2 n = O(\log_2 n)$

Oppgave 2 b)

Gitt følgende algoritme:

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

Finn antall tilordninger (=) for algoritmen og effektiviteten uttrykt i O-notasjon. Begrunn svaret. Vi ser kun på løkke kroppen når vi analyserer løkker!

$$n * n = n^2 = O(n^2)$$

Oppgave 2 c)

Gitt tre algoritmer A, B og C som alle beregner summen $1 + 2 + \dots + n = n(n+1)/2$, n heltall > 0 .

Algoritme A	Algoritme B	Algoritme C
<pre>sum = 0 for (int i = 1; i <= n; i++){ sum = sum + i; }</pre>	<pre>sum = 0 for (int i = 1; i <= n; i++){ for (int j = 1; j <= i; j++){ sum = sum + 1 } }</pre>	<pre>sum = n*(n + 1)/2</pre>


Finn antall operasjoner (tilordninger, addisjoner, multiplikasjoner, divisjoner) for hver av algoritmene og fyll ut tabellen under. Vi tar ikke med tilordning av løkkeindeks

	Algoritme A	Algoritme B	Algoritme C
Tilordninger =	$1 + n$	$1 + (n * n + 1)/2$	1
Addisjoner+	n	$(n * n + 1)/2$	1
Multiplikasjoner*	0	0	1
Divisjoner/	0	0	1
Totalt antall operasjoner	$2n + 1$	$1 + 2((n * n + 1)/2)$	4

Oppgave 2 d)

Vi antar at det foreligger 5 algoritmer av ulik orden. I tabellen er det gitt de tilhørende vekst-funksjonene. Vi er interessert i å finne ut tiden det tar å prosessere 10^6 elementer for de ulike algoritmene på en prosessor som kan utføre 10^6 operasjoner pr. sekund. $\log n$ betyr her $\log_2 n$; $\log_2 n = \ln n / \ln 2$. Fyll ut siste kolonne i tabellen under og angi med de enheter som er angitt.

$t(n)$	$t(10^6) / 10^6$
$\log_2 n$	0.00002 [sekunder]
n	1 [sekunder]
$n \log_2 n$	19.9 [sekunder]
n^2	11.6 [dager]
n^3	31709.8 [år]

$h = \frac{\log_2(10^6)}{10^6}$ $\rightarrow 0.0000199315686$	⋮
$i = \frac{10^6}{10^6}$ $\rightarrow 1$	⋮
$j = \frac{10^6 \log_2(10^6)}{10^6}$ $\rightarrow 19.9315685693242$	⋮
$k = \frac{(10^6)^2}{10^6 \cdot 60 \cdot 60 \cdot 24}$ ≈ 11.5740740740741	⋮ 
$\ell = \frac{(10^6)^3}{10^6 \cdot 60 \cdot 60 \cdot 24 \cdot 365}$ $\rightarrow 31709.7919837645859$	⋮

Oppgave 2 e)

Følgende metode avgjør om en tabell inneholder minst en duplikat: Finn antall sammenligninger i verste tilfelle for algoritmen og effektiviteten uttrykt i O-notasjon. Begrunn svaret.

```
boolean harDuplikat (int tabell[], int n){// n er antall elementer
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;
} //metode
```

Det verste tilfelle er når det ikke blir funnet duplikater og da vil koden kjøre $(n \cdot n - 1) / 2$ ganger.

Oppgave 2 f)

Vi ser på tidskompleksiteten for vekstfunksjoner til 4 ulike algoritmer (for en viktig operasjon) der n er antall elementer.

i) $t_1 = 8n + 4n^3 = O(n^3)$

ii) $t_2 = 10 \log_2 n + 20 = O(\log_2 n)$

iii) $t_3 = 20n + 2n \log_2 n + 11 = O(n \log_2 n)$

iv) $t_4 = 4 \log_2 n + 2n = O(n)$

Hva er O-notasjonen for de ulike vekstfunksjonene? Hvilken av algoritmene over er mest effektiv og hvilken er minst effektiv (i begge tilfeller for store n)? Grunngi svaret.

T2 er den mest effektive algoritmen da dette er en logaritmisk funksjon. Logaritmisk algoritme er den mest effektive algoritmen da det er en funksjon som minker for høye verdier, og er mest skalerbar.

T1 er den minst effektive algoritmen da dette er en eksponensial funksjon som vokser rask og egner seg dårlig for høye verdier, og er lite skalerbar.

Oppgave 2 g)

Gitt følgende metode:

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

Vi ønsker å måle tiden for n = 10 mill, 100 mill og 1000 mill. Bruk javametoden,

```
public static long currentTimeMillis()
```

Hvorfor er vekstfunksjonen her $T(n) = cn$, der c er en konstant?

Konstanten c er hvor lang tid det tar å utføre linjen "k = k + 5", n forteller hvor mange ganger linjen blir utført.

Bruk disse kallene: Kjør programmet noen ganger og se om du da får bedre resultater.

```
public static void main(String[] args) {  
    tid(100000000L);  
    tid(100000000L);  
    tid(1000000000L);  
}
```

Hvordan stemmer resultatene? Diskuter. Det kan ikke bli helt nøyaktige tider fordi `currentTimeMillis` er basert på systemklokken og ikke på prosessortiden. Det er flere kilder som kan forstyrre måling av tiden basert på systemklokken.

Det ser ut som resultatene blir dårligere og dårligere hver gang vi kjører programmet, og tallene blir større og større. Forskjellene blir også betydelig større når `n` går fra 10 mill til 100 mill og til 1000 mill.

Oppgave 3

i)

```
public int antall(Sjanger sjanger) {
    int antallSjanger = 0;

    for (int i = 0; i < filmTabell.length; i++) {
        if (filmTabell[i].getSjanger().equals(sjanger)) {
            antallSjanger++;
        }
    }
    return antallSjanger;
}
```

Tidskompleksiteten uttrykt i O-notasjon med `k` og `n` for metoden `antall` er: $O(kn)$

ii)

```
public void skrivUtStatistikk(FilmarkivADT filma) {

    System.out.println("Antall filmer totalt i arkivet " +
        filma.antall() + "\n");
    System.out.println("Antall filmer i sjanger action " +
        filma.antall(Sjanger.ACTION));
    System.out.println("Antall filmer i sjanger drama " +
        filma.antall(Sjanger.DRAMA));
    System.out.println("Antall filmer i sjanger history " +
        filma.antall(Sjanger.HISTORY));
    System.out.println("Antall filmer i sjanger scifi " +
        filma.antall(Sjanger.SCIFI));
}
```

Tidskompleksiteten uttrykt i O-notasjon med `k` og `n` for metoden `skrivUtStatistikk` er: $O(1)$