

- Oppgave 2 a)

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

i) $4n^2 + 50n - 10$

i uttrykk) $O(n^2)$

ii) $10n + 4 \log_2 n + 30$

ii uttrykk) $O(n)$

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

iii uttrykk) $O(n^3)$

iv) $35 + 13 \log_2 n$

iv uttrykk) $O(\log n)$

- 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 løkkekroppen når vi analyserer løkker!

Effektiviteten uttrykt: $O(n^2)$

Antall tilordninger: n^2

Begrunnelse: Antall tilordninger i denne løkken vil være det samme som notasjonens uttrykk, fordi hver gang ytre løkke kjører, vil indre løkke kjøre n ganger.

- Oppgave 2 c)

Algoritme A:

Uttrykk: $O(n)$

Tilordninger: $1+n$

Operasjoner: 2 (utgangspunkt i at vi ikke tar med løkkeindeks)

Addisjoner: 1

Multiplikasjoner: 0

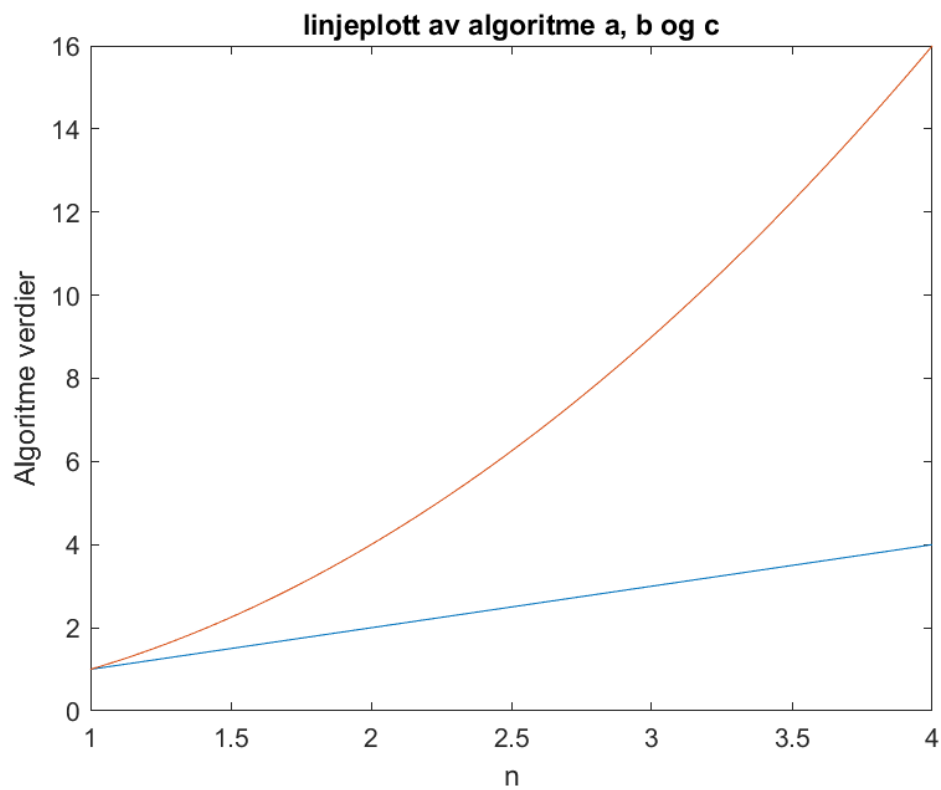
Divisjoner: 0

Algoritme B:

Uttrykk: $O(n^2)$
Tilordninger: $(n*(n+1)/2)+1$
Operasjoner: 2 (utgangspunkt i at vi ikke tar med løkkeindeks)
Addisjoner: 1
Multiplikasjoner: 0
Divisjoner: 0

Algoritme C:

Uttrykk: $O(1)$
Tilordninger: 1
Operasjoner: 1
Addisjoner: 1
Multiplikasjoner: 1
Divisjoner: 1



Oppg 2 d)

$t(n)$	$t(10^6)/10^6$
$\log_2 n$	0.00001993 sekunder
n	1 sekund
$n \log_2 n$	19.93156857 sekunder
n^2	11.57 dager
n^3	31688.76 år

Oppg 2 e)

Effektivitet: $O(n^2)$

Begrunnelse:

Den ytterste forløkka itererer fra indeks 0 til $n-2$, og for hver skritt blir hele tabellen iterert igjennom av den indre forløkka som går fra 0 til $n-1$.

Antall sammenligninger i verste tilfellet blir altså at den teller hele tabellen helt til n , dersom den ikke finner noe duplikat.

Antall sammenligninger blir:

$$n \cdot (n + 1) / 2$$

Oppg 2f)

- i) Effektivitet: $O(n^3)$
- ii) Effektivitet: $\log n$
- iii) Effektivitet: $n \log n$
- iv) Effektivitet: $O(n)$

Det er algoritme $t_1 (8n+4n^3)$ som er minst effektiv og mest tidkrevende.
Denne algoritmen er kubisk, og vokser eksponensielt.

Algoritme $t_2 (10 \log n + 20)$ er algoritmen som er minst tidkrevende og som vokser minst av disse, og vil derfor være mest effektiv i tilfellet hvor n er veldig stor. Den har lavest tidskompleksitet.

Oppg 2g)

```
tid(1000000000L);
```

A screenshot of a Java IDE's console window. The title bar shows 'Console', 'Problems', and 'Debug Shell'. The console text shows '<terminated> Main [Java Application] C:\Program Files\J' followed by a list of numbers: 31, 29, 29, 28, 28, 29, 30, 31, 32, 30.

Her varierte resultatet mellom ca. 28 til 32

```
tid(10000000000L);
```

A screenshot of a Java IDE's console window. The title bar shows 'Console', 'Problems', and 'Debug Shell'. The console text shows '<terminated> Main [Java Application] C:\Program Files\Java\jdk-14.0.2' followed by a list of numbers: 298, 304, 301, 303, 301, 309, 307, 302, 303, 304.

Her varierte resultatet mellom ca. 298 til 309

```
tid(100000000000L);
```

A screenshot of a Java IDE's console window. The title bar shows 'Console', 'Problems', and 'Debug Shell'. The console text shows '<terminated> Main [Java App]' followed by a list of numbers: 3020, 3028, 3023, 3019, 3057, 3029, 3025, 3030, 3003, 3012.

Her varierte resultatet mellom ca. 3012 til 3057

Resultatene stemmer slik at for hver 0 som blir lagt til i tid, så øker også resultatene med faktor på n, men med et større slingsmoment som er like stor relativt til n.

Oppg3)

- i) `antallSjanger(Sjanger sjanger)` sin O-notasjon er uttrykt som $O(n)$ i både `Filmarkiv.java` og `Filmarkiv2.java`
- ii) `skrivUtstatistikk(FilmarkivADT film)` sin O-notasjon er uttrykt som $O(n)$ i `Filmarkiv.java`
`skrivUtstatistikk(FilmarkivADT film)` sin O-notasjon er uttrykt som $O(n)$ i `Filmarkiv2.java`

Begrunnelse:

`antallSjanger` sin O-notasjon er lik i begge filmarkivene, da i det første filmarkivet så går den gjennom tabellen fra første indeks og sjekker hvilken sjanger det er. Tidskompleksiteten til operasjonen vil derfor øke med N antall filmer i arkivet, uavhengig av hvor mange sjangere som finnes. I det andre filmarkivet vil operasjonen starte ved første node, og gå gjennom alle noder til det er tomt, og antall noder øker med N antall filmer som er lagt til i arkivet, som gjør at antall operasjoner øker med N også.

`skrivUtstatistikk(array)`: `skrivUtstatistikk` tar i bruk metoden `hentTabell`, som peker på tabellen ($O(1)$), og går deretter gjennom tabellen for å sammenligne alle filmsjangere. Derfor vil tidskompleksiteten i denne operasjonen øke med N antall filmer i tabellen.

`skrivUtstatistikk(list)`: her tar `skrivUtstatistikk` i bruk metoden `hentFilmTabell`, hvor den går gjennom alle nodene i listen og legger elementene i en tabell, og etter dette går den gjennom tabellen, og skriver ut statistikken.

Dette gjør at tidskompleksiteten av operasjonen først må gjennom N antall noder, og deretter N antall indekser i tabellen, som vil bli uttrykt som $O(n)$, men algoritmen vil være $2n$.