Oppgave 2 øving 1

Oppgave 2 a)

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)

```
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 nå vi analyserer løkker!

Svar:

Sum = sum +1 er lik O(1), denne blir repetert n*n ganger altså n^2 tilordninger.

Koden har derfor en O-notasjon lik $O(n^2)$ fordi denne koden er i en løkke som blir gjentatt n ganger, og denne koden igjen er inni enda en løkke som blir gjentatt n ganger. Altså n^2.

Oppgave 2 c)

Algoritme A:

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

Algoritme B:

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

Algoritme C:

```
sum = n*(n + 1)/2
```

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 =	n	n^2	1
Addisjoner +	n	n^2	1
Multiplikasjoner *	0	0	1
Divisjoner /	0	0	1
Totalt antall operasjoner	2n	$2n^2$	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

t(n)	**$t(10^6)/ 10^6$**
$\log_2 n$	**0,00002 [sekunder]**
n	**1 [sekunder]**
$n \log_2 n$	**19,93 [sekunder]**
n^2	**11,57 [dager]**
n^3	**31 688 [år]**

Oppgave 2 e)

første løkke: n-1 ganger

andre løkke: n-2 ganger, da det er 1 mindre enn første løkke å sammeligne med.

Formel: $(n^2 – 3n + 2) * x$, hvor x er antall ganger hvor det blir gjort en sammenligning.

O-notasjon ved stor verdi for n er da: $O(n^2)$

Oppgave 2 f)

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

→ $O(n^3)$

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

→ $O(\log_2 n)$

iii) $t_3 = 20n + 2n\log_2 n + 2n$

→ $O(n\log_2 n)$

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

→ $O(n\log_2 n)$

Av algoritmene ovenfor så er algoritme «ii» mest effektiv og algoritme «i» minst effektiv ved stor verdi av n. Dette kan vi se gjennom å sammenligne dem som vekstfunksjoner. Da ser vi at $O(n^3)$ og $O(n\log_2 n)$ har begge større økning over tid enn $O(\log_2 n)$. Dette er også logisk med tanke på at en eksponent som n^3 fort kan bli veldig stor som vi kan se i oppgave 2d.

Oppgave 2 g)

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

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

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

→ Det vil være endringer utifra hvilke maskin/ kompilator som er brukt. Derfor har vi en ekstra konstant.

```
public static void main(String[] args) {
          long startTime = System.currentTimeMillis();
          //tid(100000000L);
          //tid(1000000000L);
          tid(10000000000L);
          long endTime = System.currentTimeMillis();
          long tidbrukt = endTime - startTime;
          System.out.println("tid i millisekund: " + tidbrukt);
     }
```

Ved bruk av Java-metoden ovenfor (`System.currentTimeMillis()`) så kan vi beregne systemtiden gått fra start til slutt av den delen fra koden vi var interessert i.

Kjørt på min maskin ble tidene slik:

«100000000L» → ~ 71 millisek.

«1000000000L» → ~ 658 millisek

«10000000000L» → ~ 6597 millisek.