MAT101 → OBLIG 1: EXERCISE

INNLEVERING 1

~ Semester: Vår 2024 ~

Gruppe nr. 38:

Johan Levent Gencher

Aleksander Rutle

Håvard Andreas Nesheim Kvalsøren

Skjermbilder: Programkjøring

```
1) Legg til film
2) Skriv ut film
3) Søk tittel
4) Søk produsent
5) Statistikk
6) Slett film
7) Avslutt
3
Skriv tittel:
Sh
Resultat:
Film nr: 1
Produsent: Aron Warner, John H. Williams, Jeffrey Katzenberg
Tittel: Shrek
År: 2001
Sjanger: COMEDY
Filmselskap: DreamWorks


Film nr: 3
Produsent: Aron Warner, John H. Williams, David Lipman
Tittel: Shrek 2
År: 2004
Sjanger: COMEDY
Filmselskap: DreamWorks
```

```
1) Legg til film
2) Skriv ut film
3) Søk tittel
4) Søk produsent
5) Statistikk
6) Slett film
7) Avslutt
4
Skriv produsent:
golimo
Resultat:
Film nr: 2
Produsent: Sveinung Golimo, John M. Jacobsen
Tittel: Max Manus
År: 2008
Sjanger: ACTION
Filmselskap: Filmkameratene AS
```

```
1) Legg til film
2) Skriv ut film
3) Søk tittel
4) Søk produsent
5) Statistikk
6) Slett film
7) Avslutt
6
Velg film (film nr):
5
Finner ingen filmer med det nr.
1) Legg til film
2) Skriv ut film
3) Søk tittel
4) Søk produsent
5) Statistikk
6) Slett film
7) Avslutt
5
Antall filmer: 2
Antall actionfilmer: 1
Antall dramafilmer: 0
Antall historiefilmer: 0
Antall scififilmer: 0
Antall komediefilmer: 1
```

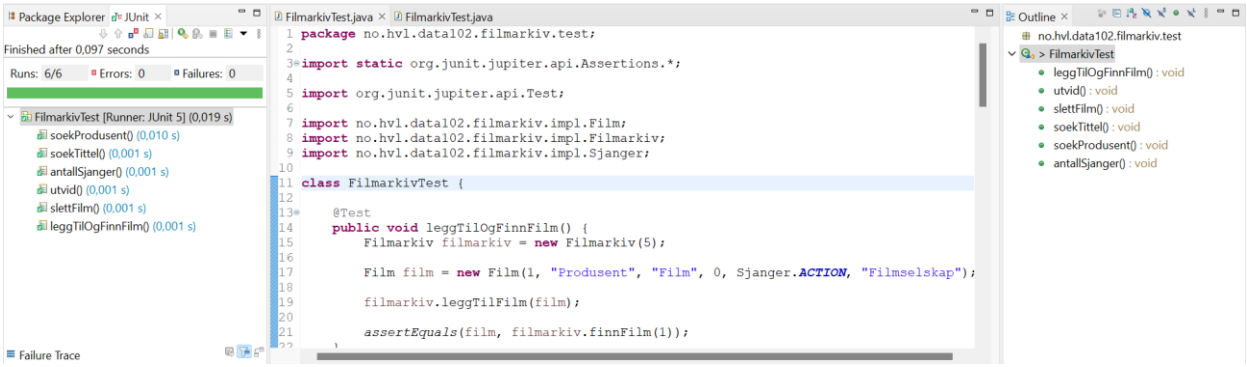
```
1) Legg til film
2) Skriv ut film
3) Søk tittel
4) Søk produsent
5) Statistikk
6) Slett film
7) Avslutt
5
Antall filmer: 3
Antall actionfilmer: 1
Antall dramafilmer: 0
Antall historiefilmer: 0
Antall scififilmer: 0
Antall komediefilmer: 2

1) Legg til film
2) Skriv ut film
3) Søk tittel
4) Søk produsent
5) Statistikk
6) Slett film
7) Avslutt
6
Velg film (film nr):
3
Vellykket
```

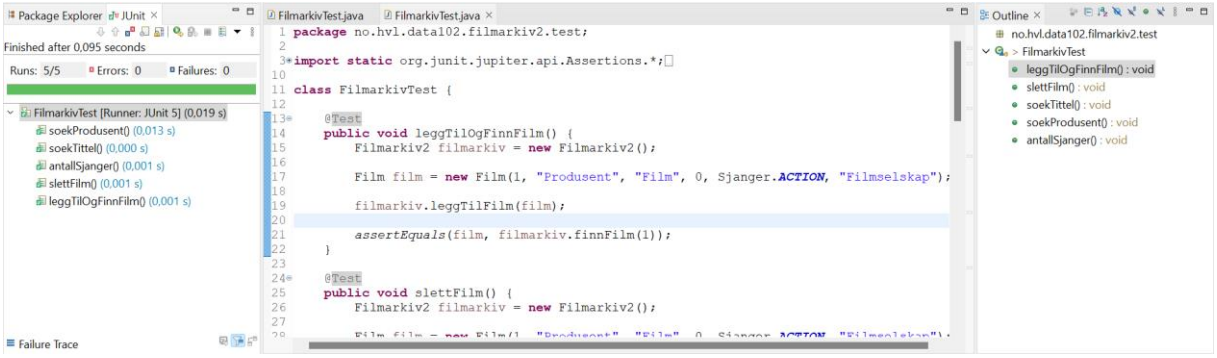
```
1) Legg til film
2) Skriv ut film
3) Søk tittel
4) Søk produsent
5) Statistikk
6) Slett film
7) Avslutt
7
```

Skjermbilder: Kjøring av enhetstester

Oppgave 1



Oppgave 2



Oppgave 3 a)

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

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

Identifiserer den dominerende termen:

$4n^2$: Andreordens ledd.

$50n$: Førstegrads ledd, vokser saktere enn $4n^2$.

-10: En konstant som ikke påvirker vekstraten til funksjonen

Derfor vil størrelsesordenen være kvadratisk $O(n^2)$ siden det lineære og konstante leddet blir relativt ubetydelig for store verdier.

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

Identifiserer den dominerende termen:

$10n$: Lineært ledd.

$4\log_2 n$: Logaritmisk ledd, vokser saktere enn lineært.

30: Konstant ledd, påvirker ikke vekstraten.

Derfor vil størrelsesordenen være lineært O(n) siden det logaritmiske og konstante leddet blir relativt ubetydelig for store verdier.

iii. $13n^3 + 22n^2 + 50n + 20$

Identifiserer den dominerende termen:

$13n^3$: Tredjeordens ledd.

$22n^2$: Andreordens ledd, vokser saktere enn kubisk.

$50n$: Lineært ledd, vokser saktere enn kubisk og kvadratisk.

20: Konstant ledd, påvirker ikke vekstraten.

Derfor vil størrelsesordenen være kubisk $O(n^3)$ siden det kvadratiske, lineære og konstante leddet blir relativt ubetydelig for store verdier.

iv. $35 + 13\log_2 n$

Identifiserer den dominerende termen:

$13\log_2 n$: Logaritmisk ledd, vokser saktere enn lineært.

35: Konstant ledd, påvirker ikke vekstraten.

Derfor vil størrelsesordenen være logaritmisk O(log n) siden det konstante leddet ikke påvirker vekstraten.

Oppgave 3 b)

Gitt følgende algoritme:

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

Finn antall tilordninger (=) for algoritmen og effektiviteten uttrykt i O-notasjon. Begrunn svaret.

Tilordninger:

1. Initialisering av sum: 'sum = 0;'. 1 tilordning.
2. Initialisering av løkkevariabel 'i': 'int i = n;'. 1 tilordning.
3. Oppdatering av 'i': 'i = i/2'. $\log_2 n$ tilordning.
4. Oppdatering av 'sum': 'sum = sum + i;'. $\log_2 n$ tilordning.

Løkken kjører ved å dele n på 2 helt til n > 1, dette kan altså utrykkes som logaritmen til n. Derfor blir kan antall tilordninger skrives som $2 + 2\log_2 n$, n>1.

Effektiviteten til algoritmen kan skrives O(log n) siden størrelsesordenen er logaritmisk.

Oppgave 3 c)

Gitt følgende algoritme:

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

Finn antall tilordninger (=) for algoritmen og effektiviteten uttrykt i O-notasjon. Begrunn svaret.

Tilordninger:

1. Initialisering av sum: 'sum = 0;'. 1 tilordning.
2. Initialisering av løkkevariabel 'i': 'int i = 1;'. 1 tilordning.
3. Oppdatering av 'i': 'i++'. n tilordninger.
4. Initialisering av 'j' for hver iterasjon av 'i': 'int j = 1;' n tilordninger.
5. Oppdatering av 'j': 'j = j * 2'. $n\log_2 n$ tilordninger.
6. Oppdatering av sum: 'sum += i * j;' $n\log_2 n$ tilordninger.

Løkken kjører ved å plusse i med 1 helt til i > n. Utefra rekkefølgen av operasjoner som vist over kan antall tilordninger skrives som $2 + 2n + 2n\log_2 n$, n>i tilordninger.

Effektiviteten av algoritmen kan derfor skrives som O(nlog n).

Oppgave 3d)

Vi ser på en sirkel med radius r. Da vil areal og omkrets være gitt med formlene:

$2\pi r^2$ og $2\pi r$

Angi i stor O-notasjon hvordan areal og omkrets vokser. Dette har ikke direkte med en algoritme å gjøre, men er med for å sjekke om dere har forstått begrepene vekstfunksjon og stor O-notasjon. Oppgaven er svært lett om dere har skjønt begrepene.

Siden vi er interessert i hvordan disse utrykkene vokser kan vi fjerne konstanter som 2 og π. Da ser vi at arealet av sirkelen vokser kvadratisk $O(r^2)$ og omkretsen av sirkelen vokser lineært (r).

Oppgave 3e)

Følgende metode avgjør om en tabell med n elementer inneholder minst ett duplikat:

```
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;
}
```

Finn antall sammenligninger i verste tilfelle for algoritmen og effektiviteten uttrykt i O-notasjon. Begrunn svaret.

I verste tilfelle for algoritmen blir det nødvendig å kjøre gjennom den ytre løkka n-1 ganger, og for hver gjennomgang av den ytre løkka vil den indre løkka kjøre gjennom elementene som står igjen. Det ser vi at løkka vil bli kjørt gjennom (n-1) + (n-1-1) + (n-1-1-1) + … 2 + 1 + 0 ganger. Vi vet at summen for de første n naturlige tallene er n(n+1)/2, og ved å erstatte n med n-1 får vi (n-1)(n)/2.

Dermed blir antall sammenligninger (n-1)(n)/2.

I en stor O-notasjon tar vi bare med den største graden av n fra hver operasjon i metoden. I denne metoden vil vi stå igjen med en n fra den ytterste løkken og en n fra den innerste løkken.

Stor O-notasjonen til denne metoden blir da O(n*n) = O(n^2).

Oppgave 3f)

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

i. $t_1(n) = 8n + 4n^3$
ii. $t_2(n) = 10\log2(n)+20$
iii. $t_3(n) = 20n+2n*\log_2(n)+11$
iv. $t_4(n) = 4\log_2(n) + 2n$

Hva er O-notasjonen for de ulike vekstfunksjonene?

Ranger vekstfunksjonene etter hvor effektive de er (fra best til verst). Anta at n er stor.

Her er vekstfunksjonene omgjort til O-notasjon.

i. $t_1(n) = O(n^3)$
ii. $t_2(n) = O(\log_2(n))$
iii. $t_3(n) = O(n*\log_2(n))$
iv. $t_4(n) = O(n)$

Hvis vi rangerer disse fra best til verst, får vi:

1. $t_4(n) = O(\log_2(n))$
2. $t_2(n) = O(n)$
3. $t_3(n) = O(n*\log_2(n))$
4. $t_1(n) = O(n^3)$

Oppgave 3g)

Gitt følgende metode:

```
Public static void tid(long n) {
    Long k = 0;
    For (long i = 1; i <= n; i++) {
        K = k + 5;
    }
}
```

[…]

Public static long currentTimeMillis().

Denne metoden kan kalles før og etter tid()-metoden og så beregner man tiden det har gått mellom de to kallene.

Hvorfor er vekstfunksjonen tid()-metoden T(n) = cn, der c er en konstant?

I metoden har vi en løkke som utgjør en konstant operasjon, og adderer 5 til k. Vi vet at tidsbruken på operasjonen er konstant ettersom operasjonen ikke er basert på noen variabel. Siden denne operasjonen blir utført n ganger, kan vi si at T(n) = cn, der c er tiden det tar å utføre operasjonen.

Vi ønsker å måle tiden for n=10^7, 10^8 og 10^9 når vi kaller tid()-metoden. Tidsmålingene blir ikke helt nøyaktige siden currentTimeMillis er basert på systemklokken og ikke på prosessortiden. Det er flere kilder som kan forstyrre måling av tiden basert på systemklokken. Du får mer nøyaktige tider ved å kjøre metoden flere ganger og så finne gjennomsnittet.

Hvordan stemmer resultatene med vekstfunksjonen? Diskuter.

Vi har en kode som måler tiden som tid()-metoden bruker for n=10^7, 10^8 og 10^9. Vi prøver å kjøre den flere ganger:

6 8 30	5 7 30	8 12 30	6 7 29	6 8 27

Her har vi resultatene, der den øverste linjen viser n=10^7, midterste viser n=10^8 og nederste viser n=10^9 i millisekund. I følge vekstfunksjonen T(N) = cn burde øverste resultat være 100 ganger mindre enn nederste resultat, men ifølge våre verdier stemmer ikke dette. Mest sannsynlig er det andre årsaker som fører til at koden ikke kjører på optimal/jevn fart. For eksempel kan resten av koden være ineffektiv eller uoptimalisert, noe som skader resultatet vi får.