Oblig 1 DAT102

* **Liste over deltakere**

Tobias Hanssen

Andreas Falkmo

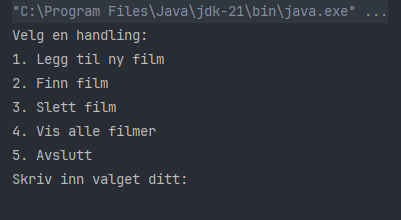
Henrik Berntsen

Jørgen Troland

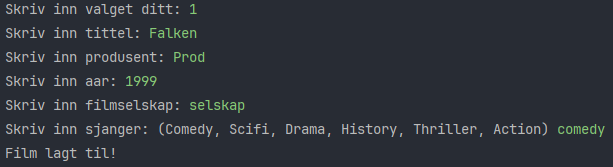
**Skjermbilder:**

**Main:**

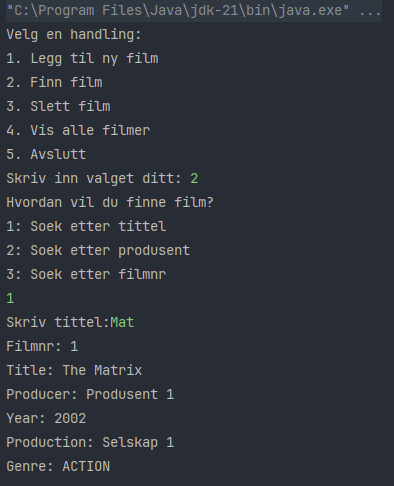
**Velg handling:**



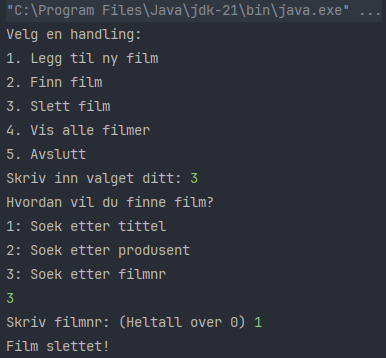
Legg til film:



Finn film:



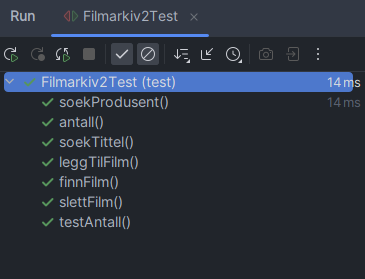
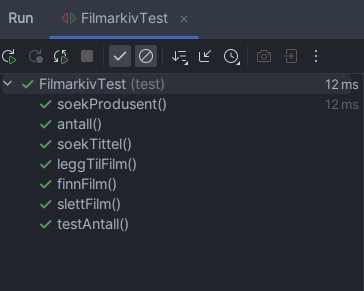
Slett film:



Vis alle filmer:



Tester:



**Oppgave 3 A)**

* i) 4+ 50n - 10 = **O ()**
* ii) 10n + 4 = **O (n)**
* iii) 13+ 22+ 50n + 20 = **O ()**
* iv) 35 + 13log n = **O (log n)**
* **Oppgave 3 B)**

Første iterasjon: i = n

Andre iterasjon: i = n/2

Tredje iterasjon: i = n/4

Vi ser at antall tilordninger avhenger av antall gjennomganger av løkken, og tilordningen til i halveres gjennom hver iterasjon. Dette kan uttrykkes på følgende måtes: så lenge i =< 1.

( k representerer antall iterasjoner)

Dette kan løses som en ligning: 1. multipliser med 2k, da får vi: n = . 2. Ta logaritme på begge sider: log2 n = log2 k = log n

Derfor kan vi beskrive effektiviteten og antall tilordninger som: **O (log n)**

* **Oppgave 3 C)**

Her er en dobbel for løkke, så da startet vi med den ytre. I den ytre for løkken finner vi initialiseringen ( 1 tilordning ), så betingelsen: *n* + 1 (tar med den siste som vil feile). Deretter finner vi økningen *n*.

Den ytre vil da ha 2n + 2 tilordninger.

Den indre for løkken inneholder initialisering (1 tilordning), deretter betingelse: log2n + 1 ( tar med siste hvor den mislykkes), hvor j ganges med 2 for hver iterasjon. Deretter har vi økningen, som skjer *n* ganger, og kan uttrykkes ved log2n. Til slutt har vi tilordning ( sum += i \* j ) som skjer n ganger.

Den indre vil da ha initialiseringen (1), betingelsen (log2n + 1), økningen (log2n), tilordningen (n) som gir oss: n \* 2log2n +2.

Totalt vil dette gi oss 2n + 2 + n (2Log2n + 2) tilordninger.

Det dominerende leddet for veksten når *n* blir stor, vil være uttrykket vi fant for antall tilordninger i den indre løkken. Dette var da: n \* 2Log2n + 2. Skriver vi dette ved O-notasjon vil vi få: O(n log n)

* **Oppgave 3 D)**

Formelen for areal av en sirkel er: 2, og O- notasjon for dette ville vært: **O(**

Formelen for omkrets av en sirkel er: 2, og O- notasjon for dette ville vært: **O(r)**

* **Oppgave 3 E)**

Antall sammenligninger i verste tilfelle for algoritmen og effektiviteten. Vi må se på antall iterasjoner i begge løkkene. Antall iterasjoner i den ytre løkken er (n-1), den går fra 0 til n - 2. I den indre løkken er antall iterasjoner ( n - indeks - 1), den grå fra indeks + 1 til n - 1.

Forenklet blir dette (n(n-1)) / 2. Og i O-notasjon blir dette O().

Effektiviteten i O-notasjon = **O(**

Antall sammenligninger i verste tilfelle = **(n(n-1))/2**

* **Oppgave 3 F)**

i) O-notasjon for 8n + 4= **O (**

ii) O-notasjon for 10log2n + 20= **O(log n)**

iii) O-notasjon for 20n + 2nlog2n + 11 = **O(n log n)**

iv) O-notasjon for 4log2n + 2n = **O(n)**

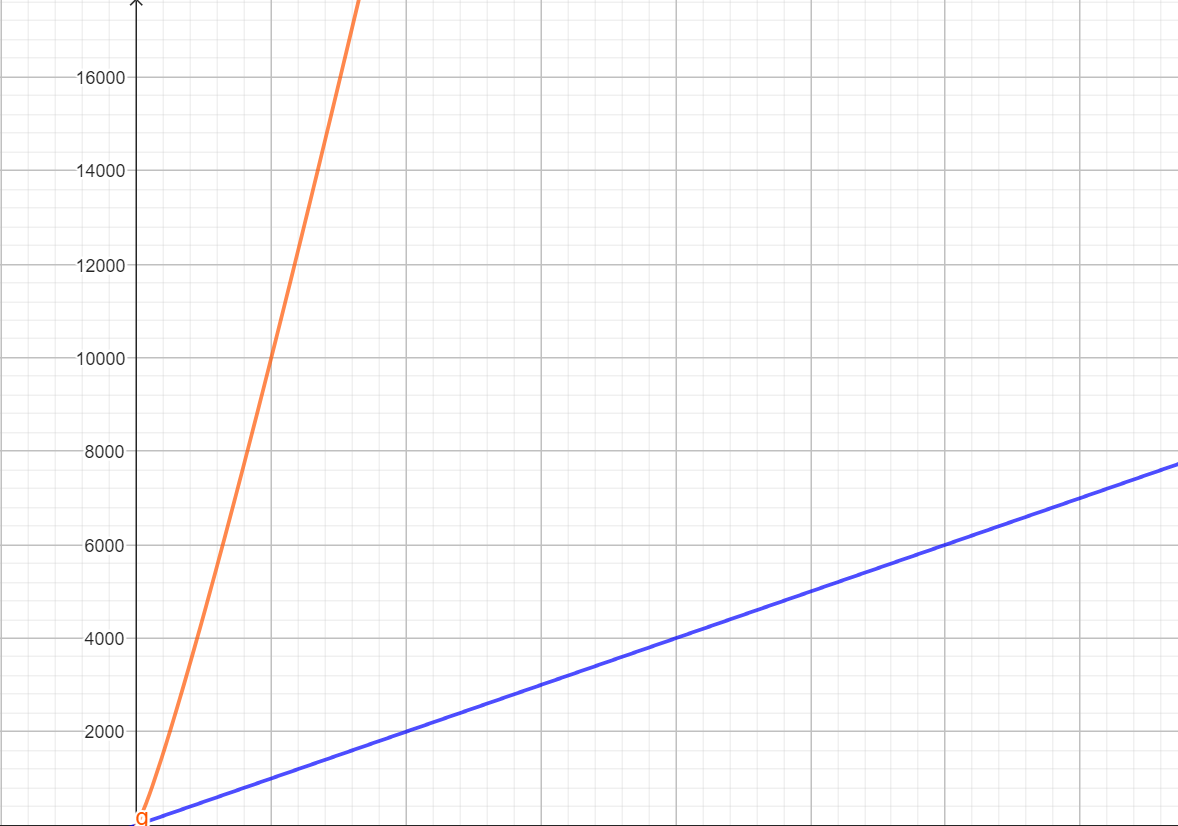
**Rangering av de ulike O-notasjonene:**

En kubisk funskjon vil vokse svært raskt, spesielt når n blir stor.

En logaritmisk funskjon vil vokse svært sakte, nesten vannrett på en graf etterhvert.

På bildet under blir t3 og t4 representert. Den oransje funskjonen er t3, og den blå er t4. t4 vil være en en linær graf, og ha tregere vekst enn t3.

**Dermed vil rangeringen bli ( fra best til verst): t2 < t4 < t3 <t1**



**Oppgave 3 G)**

i)

Ut ifra metoden vi ser i starten av oppgaven ser vi at for hver gang løkken kjøres bli variablen k økt med 5. Vi ser også fra for-løkken at løkken går fra 1 til n. Som vil si at hvis n = 5 går løkken 5 ganger og k blir der igjen økt med 5, 5 ganger. Som igjen kan skrives som k\*n. Så i vårt tilfelle kan vi si at k og c er det samme. k er konstanten i vekstfunksjonen tid()-metoden T(n)=cn.

ii)

Vi vet at T(n)=cn, hvor c er en konstant. Vi kan da forvente at resultatene vil vise oss at tiden øker linært med n. Vi kan også for å være mer sikker på resultatene hvis vi kjører metoden flere ganger også finner gjennomsnittet. Resultatene stemmer ganske greit med vekstfunksjonen. De er litt opp og ned fra måling til måling men det er mest sannsyneligvis av eksterne årsaker.

Etter å ha kjørt programmet 5 ganger, var resultatene følgende (i millisekunder):

= 3, 3, 4, 4, 3

= 7, 8, 7, 9, 11

= 19, 18, 19, 17, 18

Gjennomsnittet for følgende ble : 3,4 - 8,4 - 18,2. Dette stemmer fint overens for å sikre mot eksterne årsaker som kan påvirke resultatet.