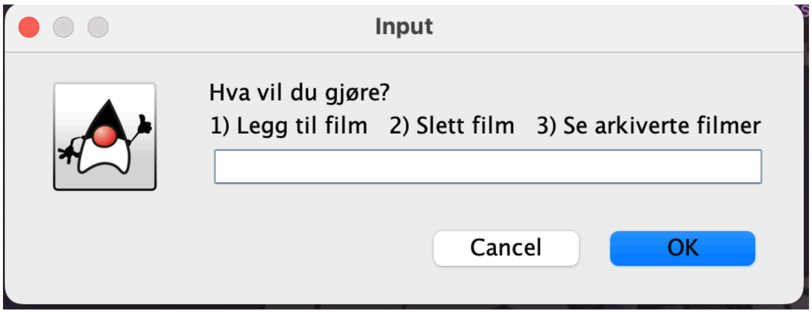
Innlevering 1

Gruppe 32:

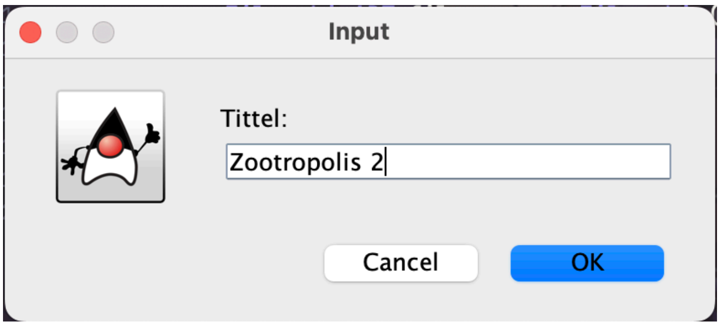
- Grethe Just-Olsen
- Andrea Ringstad
- Solfrid Bryn
- Ragnhild Grande

Oppgave 1

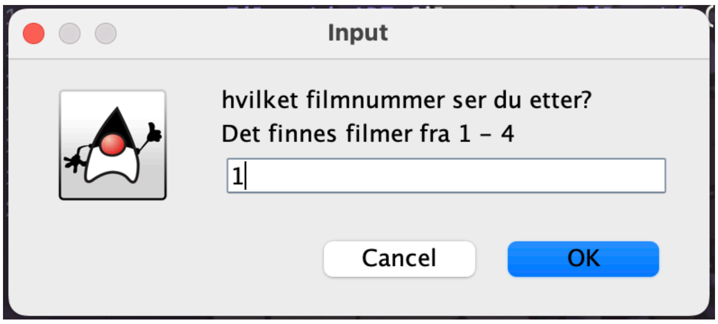
Når vi først åpner programmet, åpner det et dialogvindu der du kan velge mellom forskjellige valg fra 1 til 3:



For eksempel, hvis du vil legge til en film, vil det komme opp en rekke pop-up-vinduer som spør deg om å skrive inn informasjon til filmen.



Du har muligheten til å søke opp film basert på filmnummeret.



Og hvis du vil slette en film, vil den nyeste filmen ta plassen til filmen som var slettet.

Oppgave 3

a)

i. $4n^2 + 50n - 10 \rightarrow O(n^2)$

ii. $10n + 4log(n) + 30 \rightarrow O(n)$

iii. $13n^3 + 22n^2 + 50n + 20 \rightarrow O(n^3)$

iv. $35 + 13log(n) \rightarrow O(log(n))$

b)

En tilordning for en algoritme er når en verdi lagres i en variabel ved hjelp av = operatoren. For denne algoritmen har vi fire tilordninger.

I algoritmen halveres verdien av i. Antall ganger du kan dele n på to før du kommer til 1 er proposjonalt med log_2 n. Det er alltid like mye jobb uansett hvor stor n er. Det betyr at antall operasjoner er det samme hver gang løkken kjøres. Dermed vil alt som skjer inne i løkken ta konstant tid per gjennomkjøring. Det vil si at den totale kjøretiden vokser logaritmisk med n. Derfor er algoritmens effektivitet: O(log n).

c)

Denne algoritmen har fem tilordninger.

Ytre løkke starter på 1 og øker med en. Denne kjører n ganger. Det vil si O(n). Den indre forløkken dobles for hver gang, og kjører log n ganger (O(log n)). Deretter adderer vi den ytre og den indre forløkkens effektivitet. Hele algoritmens effektivitet blir O(n log n).

d)

Areal = $2\pi r^2 \Rightarrow O(r^2)$

Omkrets = $2\pi r \Rightarrow O(r)$

e)

I verste tilfelle vil algoritmen sammenligne vært element med de resterende elementene i tabellen og ikke finne noen duplikater. Den ytre for-løkken vil gå igjennom (n-1) iterasjoner. For hver iterasjon vil den indre for-løkken gå igjennom (n - indeks -1) iterasjoner. Antall sammenligninger vokser derfor proporsjonalt med n^2 og effektiviteten til algoritmen vil være $O(n^2)$.

f)

O-notasjonen er:

$t1(n) \rightarrow O(n^3)$

$t2(n) \rightarrow O(log(n))$

$t3(n) \rightarrow O(n * log(n))$

$t4(n) \rightarrow O(n)$

Rangering(best→verst):

t2 → t4 → t3 → t1

g)

Kodesnittet viser en variabel k som blir plusset på med 5 for hver n. Dette er også en måte å si at k = 5n. Det ligner på måten vår forståelse av tid fungerer på, nemlig at det vokser lineært. Det fins ikke sekund^2, bare et sekund, som blir lagt til med 1 for hvert sekund som går. Da er vekstfunksjonen T(n) = cn (der c er en konstant), fordi det er lineær vekst i koden og i måten tid fungerer på.

Deretter, hvis vi bruker metoden currentTimeMillis(), skal vi i teorien få verdier som ligner på T(n) = cn. Resultatene varierer på grunn av at målingen blir forstyrret, men når man kjører metoden flere ganger og sammenligner resultatene, er det tydelig at det følger T(n) = cn. Det er i utgangspunktet en lineær funksjon.