DAT102 – Våren 2026

Øvingsoppgaver uke5 (26. - 30. januar)

Introduksjon

Oppgavene i dette settet inngår som del av den første obligatoriske øvingen!

Vi anbefaler at dere er 3-4 på gruppen, men aldri flere. Hvis dere lager grupper på fire, kan to og to jobbe tett sammen i smågrupper med å programmere samtidig og diskutere underveis. De to gruppene blir enige om en felles innlevering slik at dere leverer inn som en gruppe på fire. Grupper som er mindre enn tre, kan få redusert tilbakemelding siden vi har begrenset rettekapasitet.

For å oppnå en best mulig faglig utvikling må du være aktiv på lab. **Det betyr at alle må kode.**

Generelle råd:

- Les gjennom øvingen med en gang den blir utlevert.
- Start med å jobbe individuelt.
- Diskuter med de andre i gruppen.
- Legg en plan for arbeidet.

Mer informasjon om selve innleveringen står i selve **Oppgaven i Canvas**.

Oppgave 1

= Filmarkiv-oppgaven fra uke4 (tabell-implementasjon).

Oppgave 2

= Filmarkiv-oppgaven fra uke4 (lenket-liste-implementasjon).

Oppgave 3

Innledning

La oss anta det foreligger en algoritme for et spesielt problem der n (positivt heltall) er et mål for størrelsen på problemet. Et eksempel: Dersom problemet er sortering, vil størrelsen på problemet være antall elementer som skal sorteres. Tidsforbruket *t(n)* til algoritmen er ofte avhengig av problemstørrelsen.

> **Definisjon**: Vi sier at *t(n)* er av orden *f(n)* hvis det fins positive konstanter *c* og *N* slik at *t(n)* ≤ *cf(n)* for alle *n* ≥ *N* der *t(n)* ikke er negativ. I stor O-notasjon er skrivemåten *t(n)* er *O(f(n))*, der *t(n)* er vekstfunksjonen.

Det betyr at når *n* ≥ *N* vil grafen til *t(n)* ligge under grafen til *cf(n)*. Følgende kan vises:

$O\big(kf(n)\big)$ er $O(f(n))$ der k er en positiv konstant

$O\big(f(n)\big) + O(g(n))$ er $O(f(n) + g(n))$

$O\big(f(n)\big)O(g(n))$ er $O(f(n)g(n))$

Vi har også at *O(k)* der *k* er en konstant er *O(1)*, for eksempel O(10000) er O(1). Det betyr at tiden er uavhengig av størrelsen på problemet.

Eksempel: Gitt en algoritme som bruker $3n^2 + 9n$ operasjoner, dvs. $t(n) = 3n^2 + 9n$. La oss vise at $t(n)$ er $O(n^2)$ ut fra definisjonen over.

Siden $9n \leq n^2$ for alle *n* ≥ *9* har vi at $3n^2 + 9n \leq 3n^2 + n^2 = 4n^2$ for alle *n* ≥ *9*. Vi lar $f(n) = n^2, c = 4, N = 9$ i definisjonen over. Altså har vi at $t(n) = 3n^2 + 9n$ er $O(n^2)$.

Kommentar: Vi ser at det er det dominerende leddet (det leddet som vokser raskest når n øker) her n^2, som vi angir med O-notasjonen. Det betyr at i oppgaver der vi bare skal angi tidsforbruket til en algoritme uttrykt i O- notasjon behøver vi ikke finne c og N. Noen ganger der vi skal måle tidsforbruket mer nøyaktig må vi ta hensyn til konstanten i det dominerende leddet + eventuelt de lavere ordens leddene (leddene som vokser senere enn det dominerende leddet).

En aritmetisk rekke er summen av tall der avstanden mellom tallene er den samme. Formelen for en aritmetisk rekke er ((førsteLedd + sisteledd)/2)*antallLedd. Denne rekken vil ofte dukke opp i faget.

$$1 + 2 + \cdots + n = \frac{(1+n)}{2}n = \frac{n(n+1)}{2} = \frac{1}{2}n^2 + \frac{1}{2}n$$

Siste steget har vi tatt med så det blir lettere å se hvor raskt rekken vokser. Noen ganger stopper vi rekken på *n-1.* Hva blir summen da?

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. $4n^2 + 50n - 10$
ii. $10n + 4\log_2 n + 30$
iii. $13n^3 + 22n^2 + 50n + 20$
iv. $35 + 13\log_2 n$

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.

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.

Oppgave 3 d)

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.

Oppgave 3 e)

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.

Oppgave 3 f)

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. $t_2(n) = 10 \log_2 n + 20$
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.

Oppgave 3 g)

Gitt følgende metode:

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

Det finnes flere heltalstyper i Java. I DAT100 har vi stort sett brukt int, men husk at alle heltalstyper i Java har en øvre grense (i motsetning til i matematikk). Om vi trenger større heltall enn ca. $2*10^9$, kan vi bruke heltalstypen long. For å angi et long-heltall skriver vi en stor «L» bak tallet, for eksempel: 10000000000L. (For å øke lesbarheten kan det også skrives slik 10_000_000_000L)

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?

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.