Øving 2, Algoritmer og datastrukturer

Rekursiv programmering

Dere skal skrive to programmer for rekursiv multiplikasjon. Dette er i blant nyttig, fordi det finnes enkle prosessorer som ikke har multiplikasjon i instruksjonssettet. Da kan multiplikasjonen gjøres som en serie addisjoner.

Dere skal lage to ulike multiplikasjonsmetoder. Hensikten er å analysere og måle hvordan ulike rekursive programmer får ulik kjøretid. Begge metoder beregner produktet $n \cdot x$, der n er et positivt heltall, og x er et desimaltall.

Metode 1

Multiplikasjon med et heltall kan defineres rekursivt slik:

$$n \cdot x = \begin{cases} x & \text{hvis } n = 1 \\ x + (n-1) \cdot x & \text{hvis } n > 1 \end{cases}$$

Metode 2

Multiplikasjon med heltall kan også defineres slik:

$$n \cdot x = \begin{cases} x & \text{hvis } n = 1 \\ \frac{n}{2} \cdot (x + x) & \text{hvis } n \text{ er partall} \\ x + \frac{n-1}{2} \cdot (x + x) & \text{hvis } n \text{ er oddetall} \end{cases}$$

Begge metoder bryter altså rekursjonen når $n = 1$, og gjør ett rekursivt kall ellers.

Programmer begge metodene, og sjekk at de regner korrekt for odde og like n. Ikke bruk programmeringsspråkets multiplikasjon «*» noe sted, bruk rekursive kall til metoden dere skriver på. Metode 1 og 2 skal ikke kalle hverandre, bare seg selv.

Tidsmålinger og analyse

Gjør tidsmålinger for begge metodene, med små og store n. Hvis de er implementert korrekt, vil dere se at tidsforbruket for store n blir veldig ulikt.

Forklar forskjellen, ved å analysere begge metodene. (Finn kjøretiden med Θ-notasjon.)

Krav til innlevering

- Kildekode til to rekursive programmer, som beregner produkter på hver sin måte. Matematikken trenger ikke egentlig rekursjon; men dette er en øving i rekursjon – så rekursjon skal brukes i begge programmene. Det er også greit å ha ett stort program som inneholder begge metodene.
- Som alltid, unngå mappestrukturer, zip, package, ...
- Programmene skal virke, og regne korrekt. (Beregninger med desimaltall kan ha små avrundingsfeil, det er greit!) I tillegg til tidtaking, tar dere med testkode som f.eks beregner $13 \cdot 2,5 = 32,5$ og $14 \cdot 10,1 = 141,4$.
- Rapporten skal ha tidsmålinger for begge programmer og ulike n. For å få godkjent, må det være tydelig at kjøretidens avhengighet av n er ulik for de to programmene. (Og dermed er det ikke mulig å komme i mål hvis man bare måler på én n. Da er oppgaven misforstått.)
- Rapporten må kunne forklare hvorfor de to programmene får ulik kjøretid, ved hjelp av asymptotisk analyse. Analysen må passe med tidsmålingene.

Tips:

- Noen programmeringsspråk får problemer med n større enn 5000, fordi kallstakken fylles opp for metode 1. I så fall, ikke gå høyere. Bruk i stedet mange repetisjoner, for å få nok tid. Tidtakermetoden jeg viste dere i den første forelesningen kan komme godt med.
- Analyse av programmer som metode 2, er beskrevet på side 39 i læreboka. (mestermetoden)
- Den raskeste metoden for å skille mellom partall og oddetall, er slik:

  ```
  if (n & 1) { //oddetall } else { //partall}
  ```

  Her brukes bitwise-and for å sjekke om det er oddetall. Detaljene forklares i en senere forelesning.

- Debugging: legg inn utskrift først i metoden, som skriver ut hva parametrene n og x er. Skriv også ut hva som returneres, rett før `return`. Så ser dere bedre hvordan rekursjonen foregår. (Fjern debug-utskrifter før tidtaking!)