# Øving 2, Algoritmer og datastrukturer

## Rekursiv programmering

Dere skal skrive to programmer for rekursive eksponenter. Datamaskiner kan vanligvis ikke beregne en eksponent direkte, men må bruke et program. Hvis eksponenten er et desimaltall, brukes taylorrekker e.l. Når eksponenten er heltall, kan det gjøres raskere. Det skal vi prøve nå.

Dere skal lage tre ulike eksponentmetoder. Hensikten er å se hvordan ulike programmer får ulik kjøretid. Begge metoder beregner eksponenten $x^n$, der $x$ er et desimaltall, og $n$ er et positivt heltall.

### Metode 1

Eksponent med et heltall kan defineres rekursivt slik:

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

Bruk formelen som oppskrift. $x^{n-1}$ må beregnes med rekursivt kall til metode 1.

### Metode 2

Eksponent med heltall kan også defineres slik:

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

Bruk formelen som oppskrift. Her beregnes eksponenter med rekursive kall til metode 2.

## Metode 3

Bruk programmeringsspråkets egen eksponentmetode, for å sammenligne hastighet.

| | |
|---|---|
| java | Math.pow(x, n) |
| python | pow(x, n) |
| C,C++ | pow(x, n) |

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

Programmer disse metodene, og sjekk at de regner korrekt for tilfellet $5.0^{11}$.

## 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 $\Theta$-notasjon.)

## Krav til innlevering

- Kildekode til to rekursive programmer, som beregner eksponenter 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 med begge metodene i.
- 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 $5^{11} = 48828125$.
- Rapporten skal ha tidsmålinger for de tre regnemåtene, og ulike $n$ for hver av dem. For å få godkjent, må det være tydelig at kjøretidens avhengighet av $n$ er ulik for de to rekursive programmene. (Og dermed er det ikke mulig å komme i mål hvis man bare måler på én $n$. Da er oppgaven misforstått.) Tidsmåling for den tredje måten, er bare for sammenligning.
- 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 her.
- Hvis en eksponent på 5000 gir overflow, prøv å beregne $1,002^{5000}$. Det blir ikke plagsomt høyt.
- Analyse av programmer som metode 2, er beskrevet på side 39 i læreboka.
- 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. Dette er raskere enn å bruke modulusdivisjon.
- 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. Husk å fjerne slikt før tidsmålinger.