

# Øving 1, Algoritmer og datastrukturer 2020

## Rekursiv programmering

1. Oppgave 2.1–1 på side 28 i boka ( $n$  er et heltall,  $x$  er et desimaltall)
2. Oppgave 2.2–3 på side 38 i boka
3. Beregn samme eksponent med programmeringsspråkets egen metode. (For Java: `Math.pow(x,n)`, for C: `pow(x, n)` ). Sjekk at metodene deres regner riktig. Ta tiden her også, for å se hva som er raskest.

## Krav til innlevering

- Kildekode til to rekursive programmer, som beregner eksponenter på hver sin måte.
- Kildekode må ikke avhenge av en mappestruktur. Vi som retter, har andre mapper enn deg. (Min maskin har ikke «C:\Users\studentnavn\prosjekt\...» Den har ikke en gang noe som heter «C:»)
- Programmene skal virke, og regne korrekt. I tillegg til tidtaking, tar dere med testkode som f.eks beregner  $2^{10}=1024$  og  $3^{14} = 4782969$ .
- Last opp kildekode direkte i blackboard, IKKE zipfiler. Det går mye raskere å se over øvingen på den måten. Det skal gå greit å laste opp flere filer på én øving.
- 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 fikk ulik kjøretid. Asymptotisk analyse av programmene er en bra måte å gjøre dette. (Og også nyttig trening i å gjøre slike analyser.) Analysen (eller annen forklaring) må også stemme med tidsmålingene.

**Tips:**

- Ikke glem å forklare forskjellene i tidsforbruk mellom 2.1–1 og 2.1–3. Det kan f.eks. gjøres ved å analysere og finne kompleksiteten for begge programmene. Her kan mastermetoden være nyttig!
- I 2.2–3 beregner dere  $x^2$  som  $x*x$ , for å unngå uendelig rekursjon.
- Både 2.1–1 og 2.2–3 skal løses med rekursive kall. Andre løsninger er mulig, men blir ikke godkjent fordi dette skal være trening i rekursjon!
- For å ta tiden på kjøring med stor  $n$ , kan  $x^n$  lett bli et veldig stort tall. F.eks. er  $2^{64}$  større enn det største heltallet, men 64 er ikke noen stor eksponent. Derfor, la  $x$  være et desimaltall av typen `double`. Da kan programmet deres f.eks. beregne  $1.001^{5000}$  uten å få problemer. Men  $n$  må være et heltall.
- Eksponenter går uansett fort å beregne, tidtakermetoden jeg viste dere i den første forelesningen kan komme godt med. Det er ikke bare å øke opp eksponentene, Java får problemer med over 5000...