Denne videoen handler om...

1. Gjennomgang av O-notasjon med eksempler
2. Hvordan finne riktig c og n_0?
3. Hvordan analysere koden sin?
4. Når kan “trege” algoritmer være bedre enn “raske”?

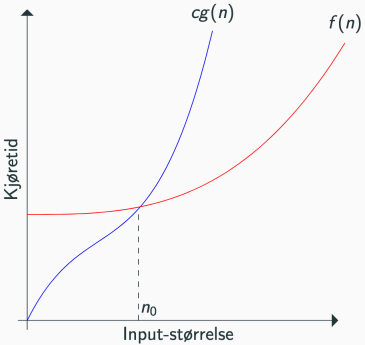
O-notasjon – Eksempler

O notasjon

La $f(n)$ være kjøretiden på en instans av størrelse n og la g være en funksjon fra heltall til reelle tall.

$f(n)$ er $O(g(n))$ hvis det finnes en konstant c og en $n_0 \geq 1$ slikt at for alle $n \geq n_0$:

$$f(n) \leq cg(n)$$



Intuisjon: $f(n)$ vokser mindre raskt enn $g(n)$ for store n.

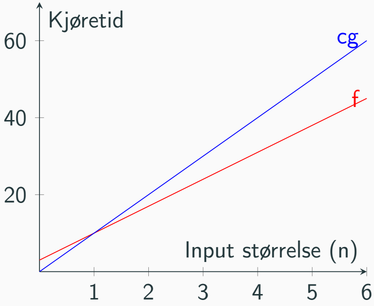
O notasjon – Eksempel

$f(n)$ er $O(g(n))$ hvis det finnes en konstant c og en $n_0 \geq 1$ slikt at for alle $n \geq n_0$:

$$f(n) \leq cg(n)$$

Eksempel:

$7n + 3$ er $O(n)$:
$f(n) = 7n + 3, g(n) = n$.
Velg: $c = 10$ og $n_0 = 1$.
$f(1) = 7 + 3 = 10, g(1) = 1$, altså $f(1) = 10g(1) = 10$
$f(2) = 14 + 3 = 17, g(2) = 2$, altså $f(2) < 10g(2) = 20$
Generelt: $f(n) = 7n + 3 \leq 10n = 10g(n)$ for alle $n \geq n_0 = 1$.



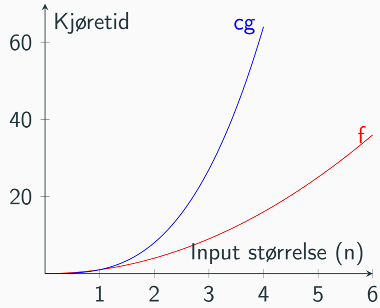
O notasjon – Eksempel

$f(n)$ er $O(g(n))$ hvis det finnes en konstant c og en $n_0 \geq 1$ slikt at for alle $n \geq n_0$:

$$f(n) \leq cg(n)$$

Eksempel:

n^2 er $O(n^3)$:
$f(n) = n^2, g(n) = n^3$.
Velg: $c = 1$ og $n_0 = 1$.
$f(1) = 1, g(1) = 1$, altså $f(1) = 1g(1) = 1$
$f(2) = 4, g(2) = 8$, altså $f(2) < 1g(2) = 8$
Generelt: $f(n) = n^2 \leq n^3 = 1g(n)$ for alle $n \geq 1$.



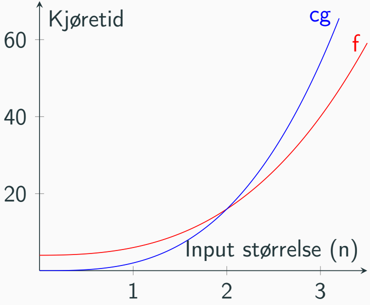
O notasjon – Eksempel

$f(n)$ er $O(g(n))$ hvis det finnes en konstant c og en $n_0 \geq 1$ slikt at for alle $n \geq n_0$:

$$f(n) \leq cg(n)$$

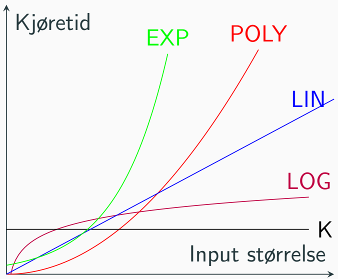
Eksempel:

$n^3 + n^2 + 4$ er $O(n^3)$:
$f(n) = n^3 + n^2 + 4, g(n) = n^3$.
Velg: $c = 2$ og $n_0 = 2$.
$f(1) = 6, g(1) = 1$, altså $f(1) > 2g(1) = 2$ (!)
$f(2) = 16, g(2) = 8$, altså $f(2) = 2g(2) = 16$
$f(3) = 40, g(3) = 27$, altså $f(3) = 40 < 2g(3) = 56$
Generelt: $f(n) = n^3 + n^2 + 4 \leq 2n^3 = cg(n)$ for alle $n \geq 2$.



Oppsummering

- K $<$ LOG $<$ LIN $<$ POLY $<$ EXP



Hvordan finne riktig g?

- Det "største leddet" er det som teller:
    - $2^n + n^{23}$ er $O(2^n)$
    - $n^5 + n^4$ er $O(n^5$ (og også, f.eks. $O(2^n)$)

Når kan "trege" algoritmer være bedre enn "raske"?

O-notasjon er forenklende!

- abstreherer bort konstanter, analyse rettet mot store instanser
- men dette kan gjøre en stor forskjell!
- Anta vi har to algoritmer som løser samme problem, en med kjøretid $f_1(n) = 100n^2$ og en med $f_2(n) = n^3$.

- for "store" instanser er $f_1(n) < f_2(n)$
- men for alle $n < 100$ er $f_2(n) < f_1(n)$
- for små instanser (input størrelse mindre enn 100) er den "trege" algoritmen raskere!