Hva er O-notasjon?
O-notasjon er en matematisk notasjon som gir en asymptotisk tilnærming til en funksjon g(x) som skrives O(g(x)). Notasjonen gir altså en enkel beskrivelse av utviklingen til g(x) når x øker. I IT-sammenheng så vil dette være relevant for kjøretiden for gitte algoritmer altså tiden det tar for at en algoritme skal kjøre.

Begreper:
"c": er en konstant som avhenger seg av hardware, programmeringspråk, software enviroment, implementasjonsdetaljer osv.

Instans "n": En størrelse som definerer kjøretiden av algoritmen i form av input.

nO: En "Integer" konstans altså et tall konstant.

Definisjon:
La f(n) være kjøretiden på en instans av størrelse "n" og la "g" være en funksjonen fra heltall til reelle tall. f(n) er O(g(n)) hvis det finnes en konstant "c" og en nO >= 1 slikt at for alle n >= nO:

f(n) <= cg(n)

Vi ser at f(n) vil alltid være tilnærmet mindre eller lik cg(n). Den vil altså aldri bli over.

Eksempler:
3n er O(n): for c = 4 er 3n <= cn = 4n

Her ser vi at 3n (f(n)) er mindre eller lik 4n (cg(n)) og følger regelen over punkt til punkt.

10^141 er O(1): for c = 10^141 + 1 er 10^141 <= cn = 10^141 + 1

For et stort tall så kan vi bare sette inn 1 siden det er et stort tall så ved å addere dette tallet med "cn" altså 10^141 med 1, så vil den være større enn f(n).

Betegnelser:	**Stor-O**	**Notat**
Konstant tid	O(1)	Vokser ikke når "n" blir større
Logaritmisk tid	O(log(n))	I praksis veldig raskt
Linær tid	O(n)	I praksis veldig raskt

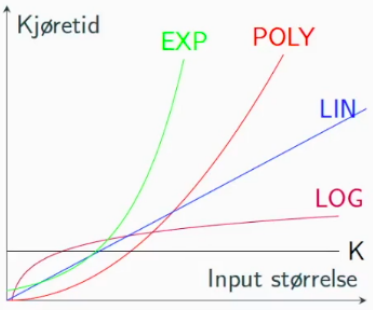
Kvadratisk tid	O(n^2)	Ofte raskt nok
Kubisk tid	O(n^3)	Ofte raskt nok
Polynomiell tid	O(n^k)	Regnes som medgjørlig
Eksponensiell tid	O(a^n),a > 1	For treigt, blir skjeldent brukt

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 $n \geq n_0 = 1.$

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 $n \geq 1.$

- 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. $O(2^n))$

Eksempel: $7n$ er $O(n)$

f g

$c \cdot g(n) - f(n) \geq 0$

$c \cdot n - 7n \geq 0$

$(c-7)n \geq 0$ $\quad c > 7$. Velge $c = 8$

$n \geq 0 \rightarrow n_0 = 1$

$n \geq n_0 \rightarrow n \geq 0$

Eksempel: $4n^3 + n^2 + 3$ er $O(n^3)$

f g

$c \cdot g(n) - f(n) \geq 0$

$c \cdot n^3 - (4n^3 + n^2 + 3) \geq 0$

$(c-4)n^3 - n^2 - 3 \geq 0$ $\quad$ Velge $c = 5$

$n^3 - n^2 \geq 3$

$n_0 = 1$: $1^3 - 1^2 = 0 \not\geq 3$ ✗

$n_0 = 2$: $2^3 - 2^2 = 8 - 4 \geq 3$ ✓

- 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!