Velkommen til repetisjonstime!

Kompleksitet og O-notasjon

Plan for timen

- Kompleksitet
- O-notasjon

Kompleksitet

- Hvor mye "ressurser" en algoritme bruker.
- Som oftest snakk om tid
- Avhenger av input
  - Som regel bruker en algoritme lenger tid hvis den får større input

```
def insertion_sort(A):
    for i from 0 to <len(A):
        j ← i
        while j>0 and A[j]<A[j-1]:
            A[j],A[j-1] ← A[j-1],A[j]
            j ← j-1
    return A
```

Vi sier $n = len(A)$

Skjer n ganger: 2

1

Skjer maks j ganger: 2

2

1

$n = 5$



$insTid(n) = 3n + n \cdot (n-1) \cdot \frac{5}{2}$

$$insTid(n) = 3n + \frac{5}{2} \cdot n \cdot (n-1)$$

$instid(0) = 0$

$instid(1) = 3$

$instid(2) = 11$

$instid(3) = 24$

$instid(1000) = 2\ 500\ 500$

$instid(2000) = 10\ 001\ 000$

dobbelt så stor

ca. 4 ganger større

```
def bubble sort(A):
    for last from len(A)-1 to = 0 :
        for i from 0 to = last:
            if A[i]>A[i+1]:
                A[i], A[i+1] ← A[i+1], A[i]
```

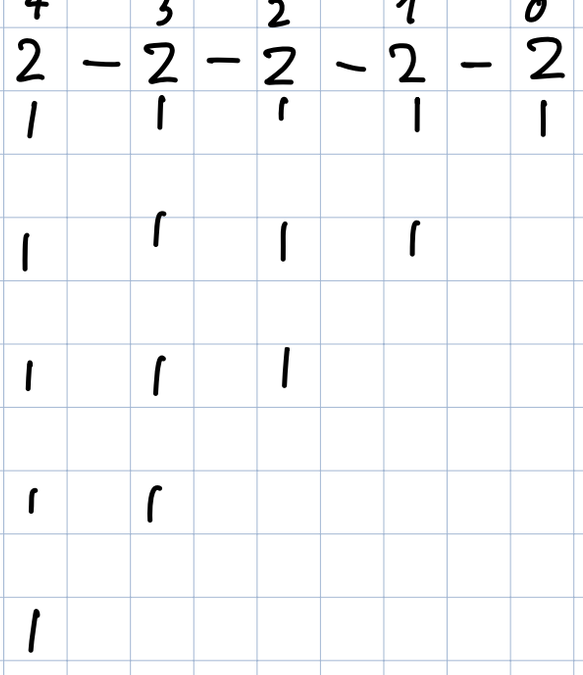
n ganger: 2

last +1 ganger: 2

1

2

last = $\overset{4}{2} - \overset{3}{2} - \overset{2}{2} - \overset{1}{2} - \overset{0}{2}$



$bubbleTid(n) = 2n + \frac{3}{2}n \cdot (n+1)$

$bubbleTid(n) \approx 2n + n(n+1) \cdot \frac{5}{2}$

$insTid(n) \approx 3n + n \cdot (n-1) \cdot \frac{5}{2}$

$bubbleTid(0) \approx 0$

$bubbleTid(1) \approx 10$

$bubbleTid(2) \approx 30$

$bubbleTid(3) \approx 60$

$insTid(0) = 0$

$insTid(1) = 3$

$insTid(2) = 11$

$insTid(3) = 24$

$bubbleTid(1000) \approx 5\ 005\ 000$

$bubbleTid(2000) \approx 20\ 010\ 000$

$insTid(1000) = 2\ 500\ 500$

$insTid(2000) = 10\ 001\ 000$

dobbelt så stor

ca. 4 ganger større

```
def elem(x, A):
    for i from 0 to < len(A):
        if A[i] = x:
            return true
    return false
```

```
maks n ganger: 2
    1
    maks 1 gang: 1
1
```

elemTid(n) = 3n + 1

elemTid(1000) = 3001

elemTid(2000) = 6001

dobbelt så stor

fortsatt bare dobbelt!

```
def elemSorted(x, A):
    low ← 0
    high ← len(A) - 1
    while low < high:
        middle ← (low + high)/2
        if A[middle] = x:
            return true
        elif A[middle] > x:
            high ← middle - 1
        else
            low ← middle + 1
    return false
```

$elemSTid(n) = 3 + \log_2(n) \cdot 4$

litt større for hver dobling ↗

1
1
maks $\log_2(n)$ ganger:
1
1

1
1
1
1

$elemSTid(1024) = 43$

$elemSTid(2048) = 47$

$elemSTid(4096) = 51$

$elemSTid(8192) = 55$

- Vi kan lage (matematiske) funksjoner som beskriver kjøretiden til en algoritme
  - $bubbleTid(n) \simeq 2n + n(n+1) \cdot \frac{5}{2}$
  - $insTid(n) = 3n + n \cdot (n-1) \cdot \frac{5}{2}$
  - $elemTid(n) = 3n + 1$
  - $elemSTid(n) \simeq 3 + \log_2(n) \cdot 4$
- Noen av disse er mye mer effektive enn andre
- Men det er litt vanskelig å se
- Vi må ha noe som hjelper oss å dra ut essensen

O-notasjon

- Generalisering/gruppering av disse funksjonene
- Forteller oss kun om endring i output (tid) i henhold til endring i input
- $O(n^2)$ sier at for "store nok" input, når input blir dobbelt så stort, blir ikke output mer enn "ca." 4 ganger større
  - Det høres ut som insTid og bobbleTid!
- $O(n)$ sier at ... når input blir dobbelt så stort, blir ikke output mer enn ca dobbelt så stort
  - elemTid

Som regel bryr vi oss bare om hvilken kompleksitetsklasse en algoritme er i, og da blir analysen enklere

```
def insertion sort(A):
    for i from 0 to <len(A):
        j ← i
        while j>0 and A[j]<A[j-1]:
            A[j],A[j-1] ← A[j-1],A[j]
            j ← j-1
    return A
```

n ganger

maks j ganger

$O(n^2)$

```
def elem(x, A):
    for i from 0 to <len(A):
        if A[i] = x:
            return true
    return false
```

n ganger

$O(n)$

$O(n^2)$ betyr egentlig $O(f)$, hvor $f(n) = n^2$

$f(k) = k^2 = k^2$

$f(2k) = (2k)^2 = 4k^2$

$g(n) = 2n^2$

$g(k) = 2k^2$ → fortsatt 4x

$g(2k) = 2 \cdot (2k)^2 = 8k^2$

altså $O(n^2) = O(2n^2)$

$insTid(n) = 3n + n \cdot (n-1) \cdot \frac{5}{2}$

$insTid \in O(insTid)$

$= O(3n + n(n-1) \cdot \frac{5}{2})$

$=$

$h(n) = n^2 + 1$

$h(k) = k^2 + 1$

$h(2k) = 4k^2 + 1$

$$insTid(n) = 3n + n \cdot (n-1) \cdot \frac{5}{2}$$

$$insTid \in O(insTid)$$

$$= O(3n + n(n-1) \cdot \frac{5}{2})$$

$$= O(3n + (n^2 - n)\frac{5}{2})$$

$$= O(3n + \frac{5}{2}n^2 - \frac{5}{2}n)$$

$$= O(\frac{5}{2}n^2)$$

$$= O(n^2)$$

$O(1)$: Uavhengig av input

$O(n)$: "Lineær". Når input blir dobbelt så stort, blir output dobbelt så stort.

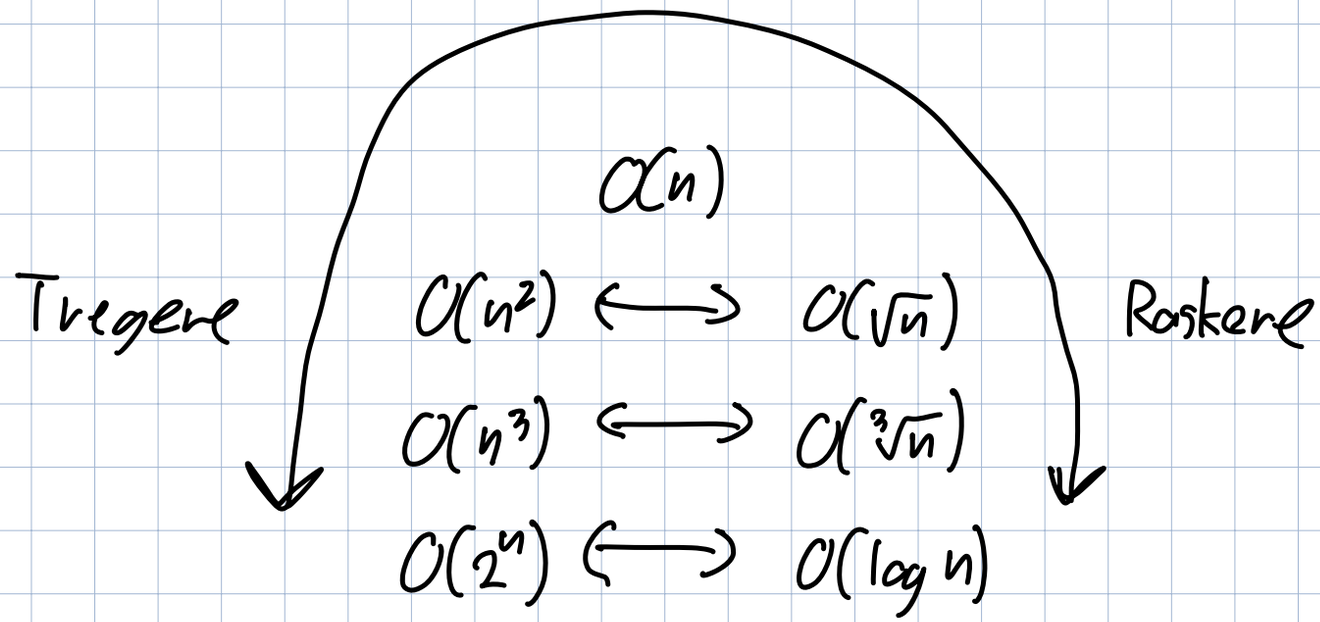
$O(n^2)$: "Kvadratisk". Når input blir dobbelt så stort, blir output 4x større.

$O(\log n)$: "Logaritmisk". Når input blir dobbelt så stort, blir output "litt" større

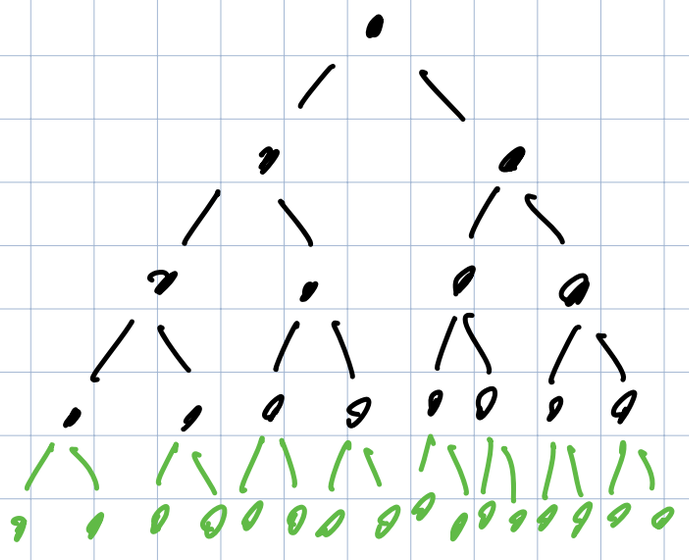
$O(2^n)$: "Eksponensiell". Når input blir "litt større", blir output dobbelt så stort.

$O(n^n)$: "Fæl". Når input blir "litt større", blir output ??? (VELDIG MYE)

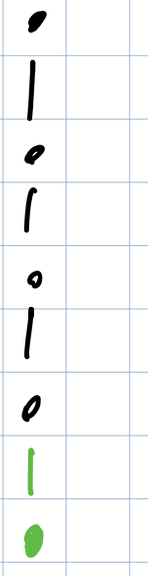
$O(\sqrt{n})$: Når input blir 4x så stort, blir output dobbelt så stort.


O(n)
Tregere
O(n²) ⟷ O(√n)
Raskere
O(n³) ⟷ O(∛n)
O(2ⁿ) ⟷ O(log n)

n (antall noder) = 15



$n = 31$



$h = 4$

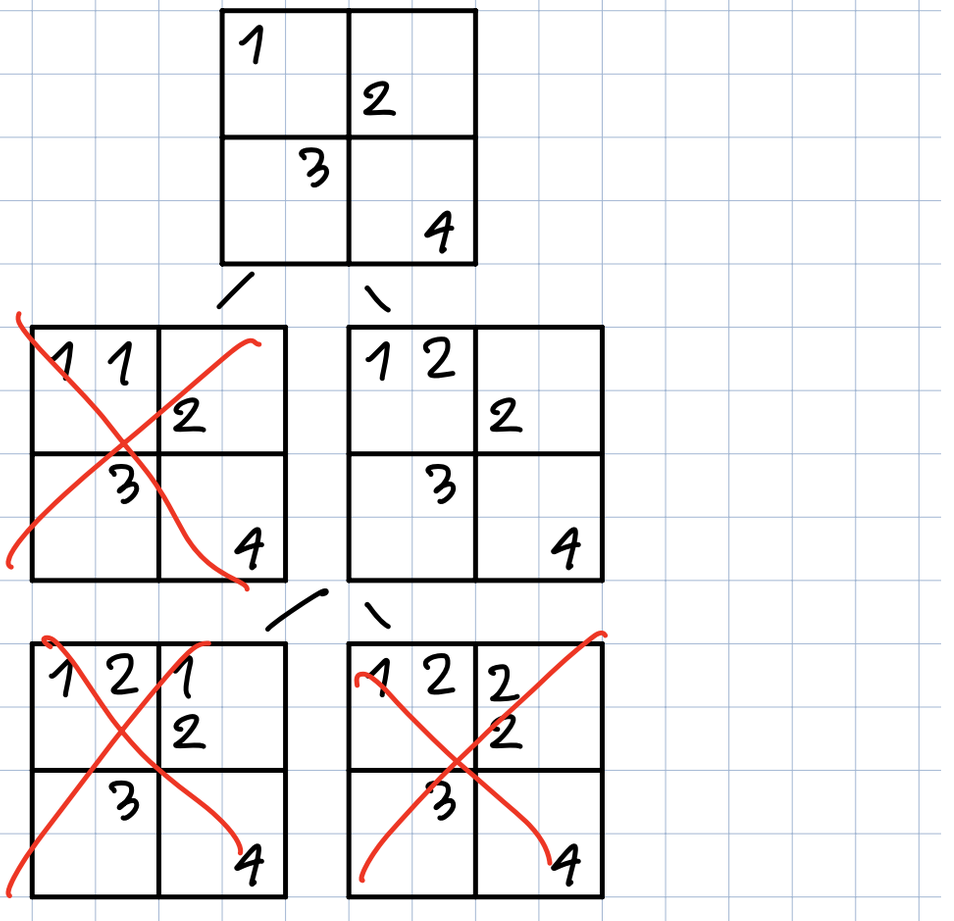
$h = 5$

$n \approx 2^h$

$h \approx \log_2 n$

```
def løsSudoku(sudoku):
    plass ← en ledig plass
    for i from 1 to =n:
        put i på plass i sudoku
        if løsSudoku(sudoku):
            return true
```

n: antall rader i sudokuen



Litt mer presist f.eks innTid f.eks $O(n^2)$

Generelt: en funksjon $f(x)$ er i $O(g(x))$ hvis

Det finnes en n_0 og en c slik at:

for alle $n > n_0$ "for store nok input"

så $f(n) \leq c \cdot g(n)$

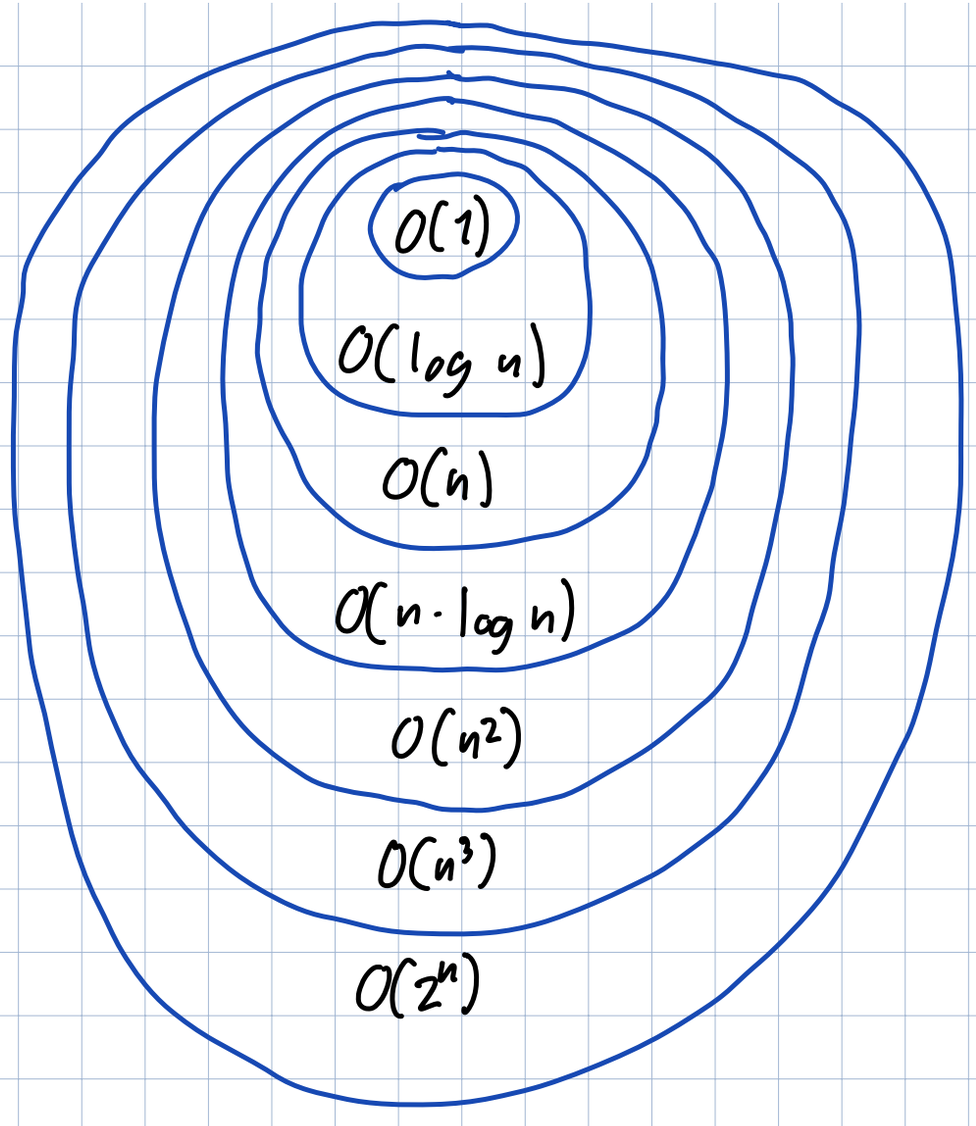
$f(x) = 2x^3 + 1$

$g(x) = x^2$

$c = 2$

$f(n) \leq 2 \cdot g(n)$

$2x^3 + 1 \in O(n^2)$


O(1)
O(log n)
O(n)
O(n · log n)
O(n²)
O(n³)
O(2ⁿ)