Asymptotisk analyse af algoritmers køretider

Analyse af køretid

Recall: Vi ønsker at vurdere (analysere) algoritmer på forhånd inden vi bruger lang tid på at implementere dem.

De to primære **spørgsmål:**

- Løser algoritmen opgaven (er den korrekt)?
- Er algoritmen effektiv (hvad er køretiden)?

Vi fokuserer i disse slides på det andet spørgsmål.

Analyse af køretid

Recall: I vores analyse er en algoritmes køretid lig antallet af basale operationer, som den udfører i RAM-modellen (for worst case input). Dette antal operationer er en voksende funktion $f(n)$ af inputstørrelsen n.

1. Vi vil starte med at undersøge, hvor godt teoretiske analyser i RAM-modellen ser ud til at passe med algoritmers observerede køretid på virkelige computere.
2. Vi vil dernæst introducere et redskab, kaldet asymptotisk analyse, til at sammenligne $f(n)$ for forskellige algoritmer på en tilpas (u)præcis måde.

Mål: at vi kan grovsortere algoritmer efter voksehastigheden af deres køretider, så vi kan undgå at implementere dem, som ikke har en chance for at være hurtigst.

Analyse vs. virkeligheden

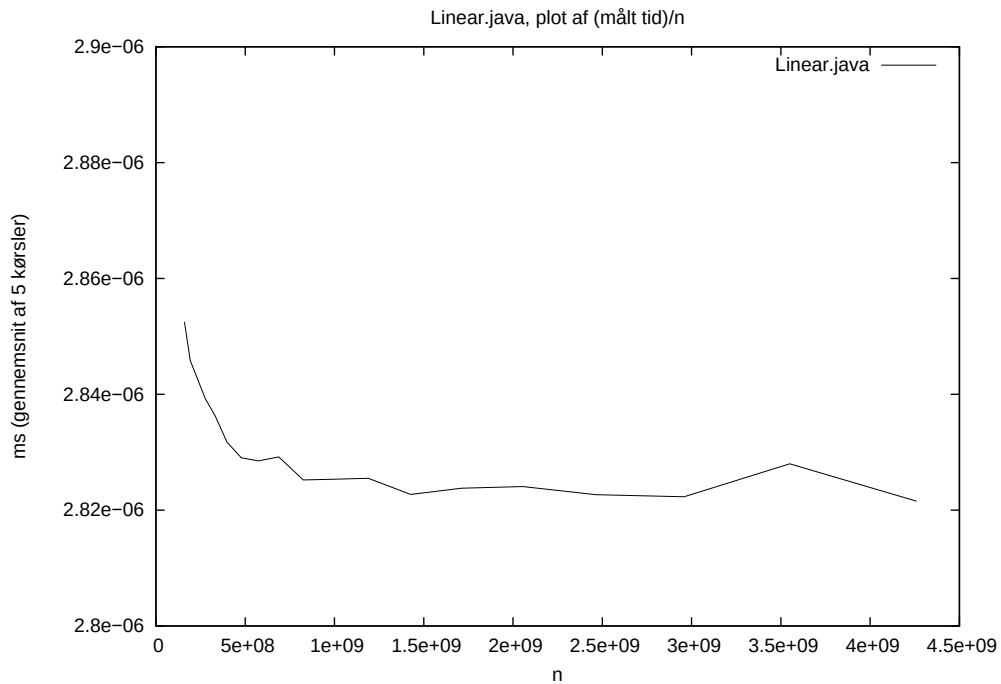
```
public class Linear {
  public static void main(String[] args) {

    long time = System.currentTimeMillis();
    long n = Long.parseLong(args[0]);
    long total = 0;
    for(long i=1; i<=n; i++){
      total = total + 1;
    }
    System.out.println(total);
    System.out.println(System.currentTimeMillis() - time);
    }
}
```

Analyse

$$\text{Tid}(n) = c_1 \cdot n + c_0$$

$$\frac{\text{Tid}(n)}{n} = \frac{c_1 \cdot n + c_0}{n} = c_1 + \frac{c_0}{n}$$



Virkelighed

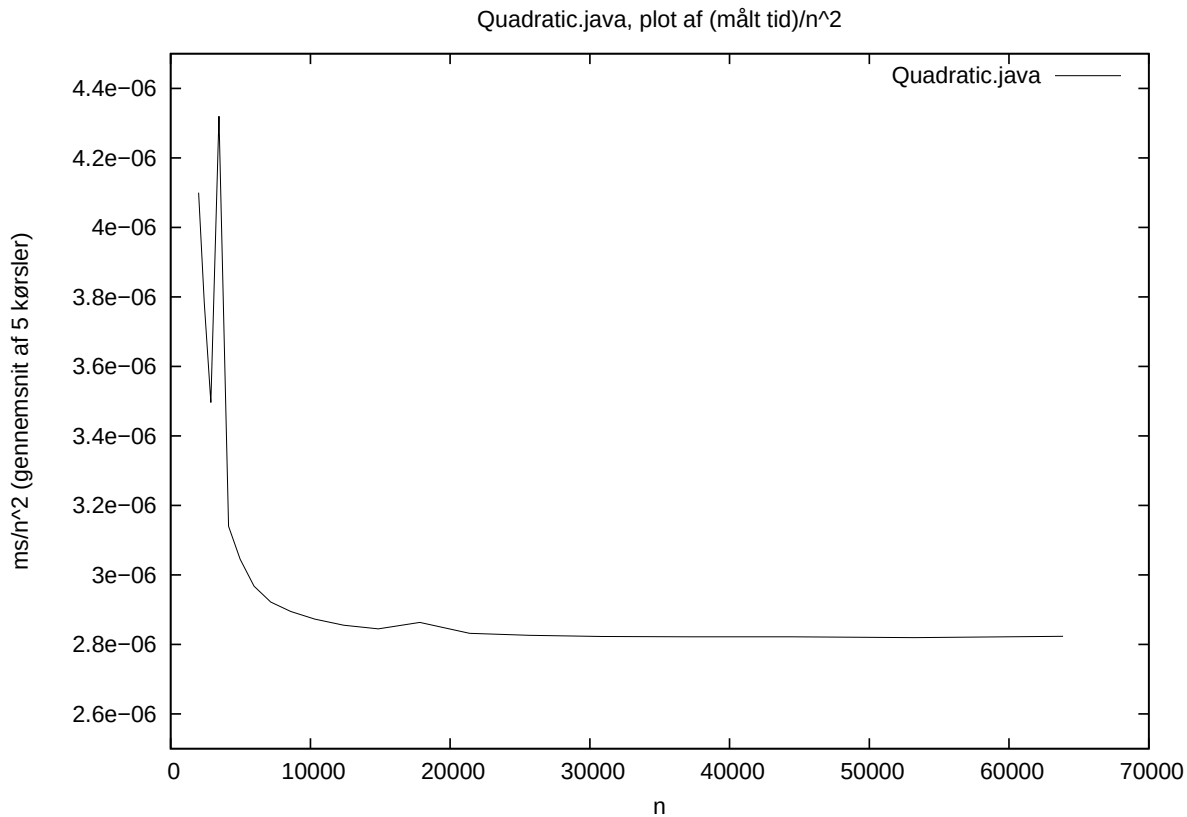
x-akse:
inputstørrelse n

y-akse:
(målt tid)/n

Analyse vs. virkeligheden

```
for(long i=1; i<=n; i++){
  for(long j=1; j<=n; j++){
    total = total + 1;
  }
}
```

$$\begin{aligned}\text{Tid}(n) &= (c_2 \cdot n + c_1) \cdot n + c_0 \\ &= c_2 \cdot n^2 + c_1 \cdot n + c_0\end{aligned}$$



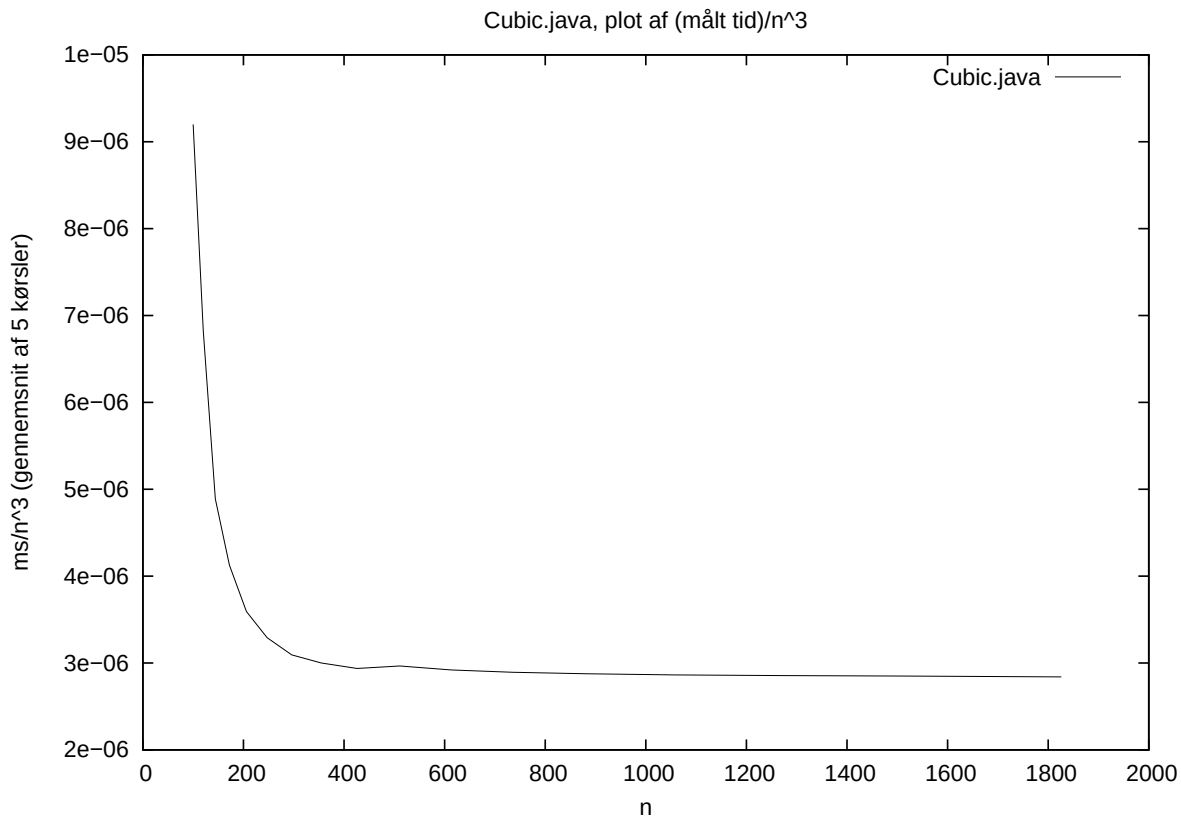
x-akse:
inputstørrelse n

y-akse:
(målt tid)$/n^2$

Analyse vs. virkeligheden

```
for(long i=1; i<=n; i++){
  for(long j=1; j<=n; j++){
    for(long k=1; k<=n; k++){
      total = total + 1;
    }
  }
}
```

$$\begin{aligned}\text{Tid}(n) &= ((c_3 \cdot n + c_2) \cdot n + c_1) \cdot n + c_0 \\ &= c_3 \cdot n^3 + c_2 \cdot n^2 + c_1 \cdot n + c_0\end{aligned}$$



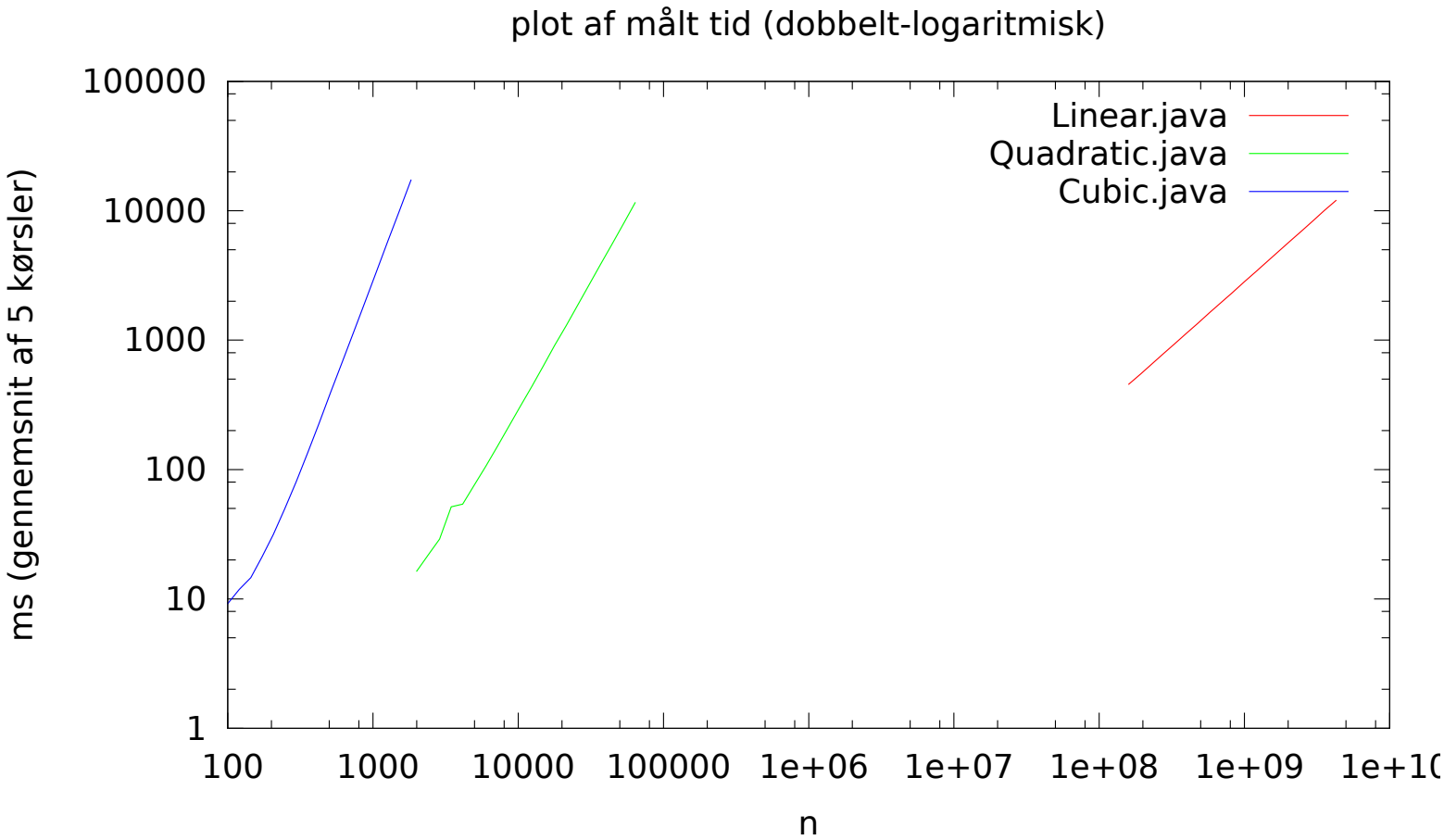
x-akse:
inputstørrelse n

y-akse:
(målt tid)$/n^3$

Analyse vs. virkeligheden

Konklusion: Det ser ud til at analyser i RAM-modellen forudser den rigtige køretid ret godt, i hvert fald for de afprøvede eksempler.

Linear vs. kvadratisk vs. kubisk



Man ser at funktionerne n, n^2 og n^3 står for meget forskellige effektiviter.

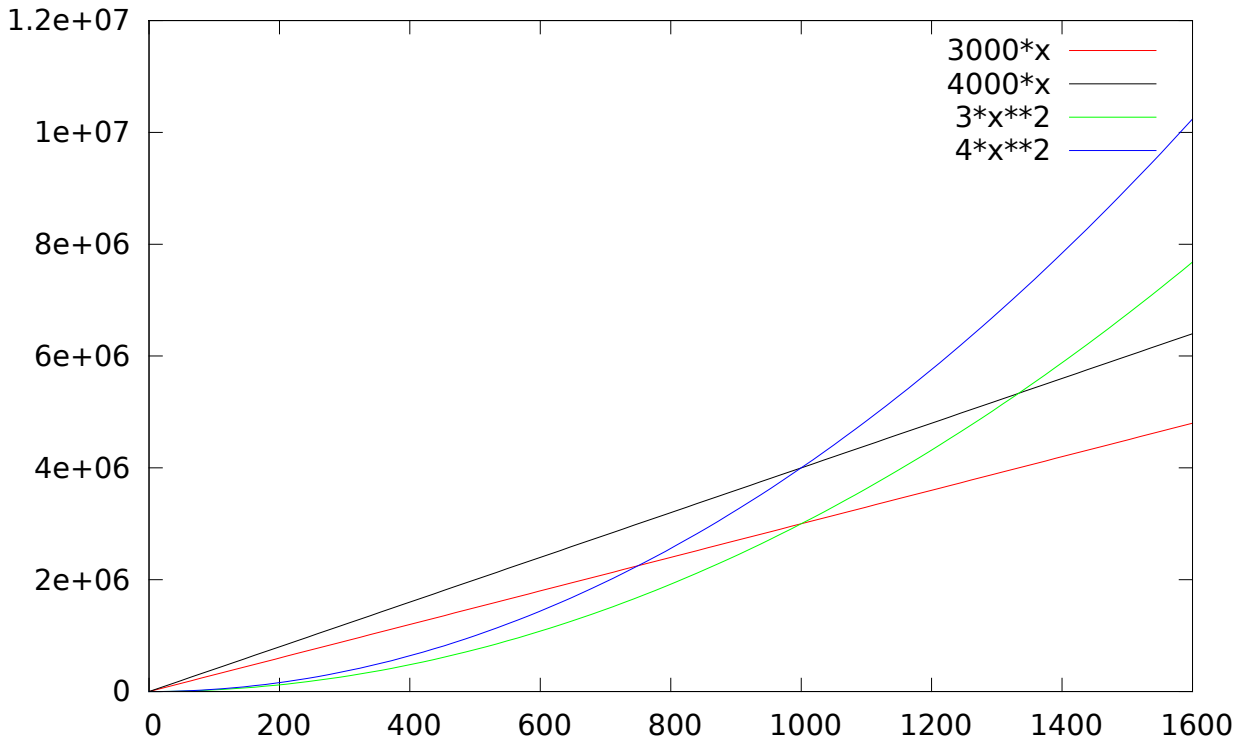
I analysen optræder i virkeligheden en del konstanter (som vi typisk har svært ved at kende præcist), f.eks. $c_1 \cdot n + c_0$. Mon disse betyder noget?

Multiplikative konstanter

Multiplikative konstanter ligegyldige hvis voksehastighed er forskellig:

$f(n) = 3000n$ $\quad$ $h(n) = 3n^2$

$g(n) = 4000n$ $\quad$ $k(n) = 4n^2$



Vinder $3000n$ over $4n^2$? Ja: $\quad 3000n < 4n^2 \Leftrightarrow 3000 < 4n \Leftrightarrow 750 < n$

Faktisk vinder $c_1 \cdot n$ **altid** over $c_2 \cdot n^2$: $\quad c_1 \cdot n < c_2 \cdot n^2 \Leftrightarrow c_1/c_2 < n$

Voksehastighed

Vi ønsker derfor at sammenligne funktioners essentielle voksehastighed på en måde, så der ses bort fra multiplikative konstanter.

En sådan sammenligning kan bruges til at lave en grovsortering af algoritmer, inden vi laver implementationsarbejde.

> Hvis to algoritmer A og B har voksehastigheder, hvor algoritme B altid (for store n) vil tabe til algoritme A, uanset hvilke multiplikative konstanter der er i udtrykkene for voksehastigheder, så vil der som regel ikke være nogen pointe i at implementere algoritme B.

Bemærk: I ovenstående situation behøver vi ikke kende konstanterne for at kunne lave denne vurdering. Vi kan derfor lave køretidsanalyse uden at bekymre os om at kende størrelsen af de indgående konstanter præcist.

Asymptotisk notation

Så vi ønsker et værktøj til at sammenligne funktioners essentielle voksehastighed på en måde, så der ses bort fra multiplikative konstanter.

Princippet for vores værktøj vil være følgende: for en funktion $f(n)$ vil vi betragte alle skaleringer af den

$$\{c \cdot f(n) \mid \text{alle } c > 0\}$$

som lige gode.

I det følgende vil vi kalde denne mængde af funktioner for $f(n)$'s klasse.

Asymptotisk notation

Baseret på dette princip definerer vi for voksehastighed for funktioner fem relationer svarende til de fem klassiske ordens-relationer:

$$\leq \quad \geq \quad = \quad < \quad >$$

De vil, af historiske årsager, blive kaldt for:

$$O \quad \Omega \quad \Theta \quad o \quad \omega$$

Hvilket udtales således:

"O", "Omega", "Theta", "lille o", "lille omega"

De fem definitioner beskrives over de næste fem sider.

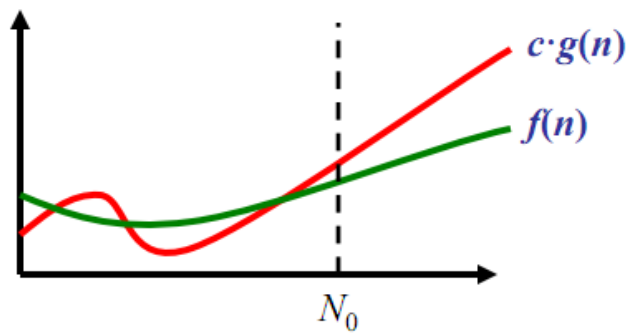
O

Definition: $f(n) = O(g(n))$

hvis $f(n)$ og $g(n)$ er funktioner $N \longrightarrow R$ og

findes $c > 0$ og N_0 så *for alle* $n \geq N_0$:

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



Mening: $f \leq g$ i voksehastighed

Princip: $f(n)$ vokser højst så hurtigt som funktioner fra $g(n)$'s klasse.

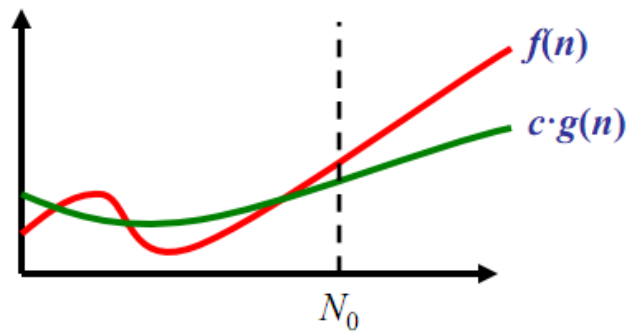
Omega

Definition: $f(n) = \Omega(g(n))$

hvis $f(n)$ og $g(n)$ er funktioner $N \rightarrow R$ og

findes $c > 0$ og N_0 så *for alle* $n \geq N_0$:

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



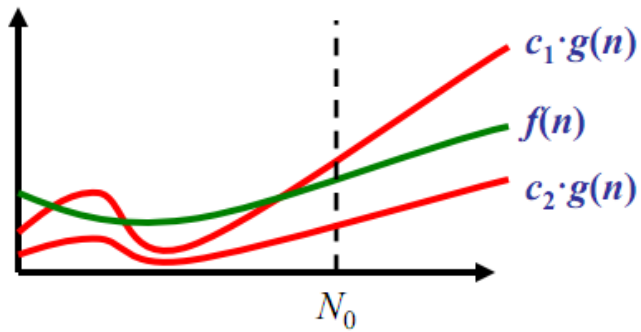
Mening: $f \geq g$ i voksehastighed

Princip: $f(n)$ vokser mindst så hurtigt som funktioner fra $g(n)$'s klasse.

Theta

Definition: $f(n) = \theta(g(n))$

hvis $f(n)$=O($g(n)$) og $f(n)$= $\Omega(g(n)$



Mening: $f = g$ i voksehastighed

Princip: $f(n)$ vokser lige så hurtigt som funktioner fra $g(n)$'s klasse.

Lille o

Definition: $f(n) = o(g(n))$

hvis $f(n)$ og $g(n)$ er funktioner $N \longrightarrow R$ og

for alle $c > 0$, *findes* N_0 så *for alle* $n \geq N_0$:

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

Mening: $f < g$ i voksehastighed

Princip: $f(n)$ vokser langsommere end alle funktioner fra $g(n)$'s klasse.

Lille omega

Definition: $f(n) = \omega(g(n))$

hvis $f(n)$ og $g(n)$ er funktioner $N \longrightarrow R$ og

for alle $c > 0$, *findes* N_0 så *for alle* $n \geq N_0$:

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

Mening: $f > g$ i voksehastighed

Princip: $f(n)$ vokser hurtigere end alle funktioner fra $g(n)$'s klasse.

Asymptotisk notation

Man kan nemt vise, at disse definitioner opfører sig som forventet af ordens-relationer. F.eks.:

$$f(n) = o(g(n)) \;\Rightarrow\; f(n) = O(g(n)) \qquad (\text{jvf. } x < y \Rightarrow x \leq y)$$

$$f(n) = \Theta(g(n)) \;\Rightarrow\; f(n) = O(g(n)) \qquad (\text{jvf. } x = y \Rightarrow x \leq y\ )$$

$$f(n) = O(g(n)) \;\Leftrightarrow\; g(n) = \Omega(f(n)) \qquad (\text{jvf. } x \leq y \Leftrightarrow y \geq x\ )$$

$$f(n) = o(g(n)) \;\Leftrightarrow\; g(n) = \omega(f(n)) \qquad (\text{jvf. } x < y \Leftrightarrow y > x\ )$$

$$f(n) = O(g(n)) \text{ og } f(n) = \Omega(g(n)) \;\Rightarrow\; f(n) = \Theta(g(n))$$
$$(\text{jvf. } x \leq y \text{ og } x \geq y \Rightarrow x = y\ )$$

Asymptotisk analyse i praksis

De asymptotiske forhold mellem de fleste funktioner f og g kan afklares ved følgende to sætninger (som kan vises ud fra definitionerne):

$$\text{Hvis } \frac{f(n)}{g(n)} \to k > 0 \text{ for } n \to \infty \text{ så gælder } f(n) = \Theta(g(n)) \qquad (1)$$

$$\text{Hvis } \frac{f(n)}{g(n)} \to 0 \text{ for } n \to \infty \text{ så gælder } f(n) = o(g(n)) \qquad (2)$$

Eksempler:

$$\frac{20n^2 + 17n + 312}{n^2} = \frac{20 + 17/n + 312/n^2}{1} \to \frac{20 + 0 + 0}{1} = 20 \text{ for } n \to \infty$$

$$\frac{20n^2 + 17n + 312}{n^3} = \frac{20/n + 17/n^2 + 312/n^3}{1} \to \frac{0 + 0 + 0}{1} = 0 \text{ for } n \to \infty$$

Asymptotisk analyse i praksis

Derudover er det godt at kende følgende fact fra matematik:

$$\text{For alle } a > 0 \text{ og } b > 1 \text{ gælder } \frac{n^a}{b^n} \to 0 \text{ for } n \to \infty \tag{3}$$

Dvs. ethvert polynomium er $o()$ af enhver exponentialfunktion

Eksempelvis giver dette at:

$$\frac{n^{100}}{2^n} \to 0 \text{ for } n \to \infty$$

hvoraf ses

$$n^{100} = o(2^n)$$

Asymptotisk analyse i praksis

Samt følgende fact:

$$\text{For alle } a, d > 0 \text{ og } c > 1 \text{ gælder } \frac{(\log_c n)^a}{n^d} \to 0 \text{ for } n \to \infty \qquad (4)$$

> Dvs. enhver logaritme (selv opløftet i enhver potens) er $o()$ af ethvert polynomium.

Eksempelvis giver dette at:

$$\frac{(\log n)^3}{n^{0.5}} \to 0 \text{ for } n \to \infty, \text{ hvoraf ses } (\log n)^3 = o(n^{0.5})$$

Asymptotisk analyse i praksis

Regel (4) kan i øvrigt nemt ses at være en variant af regel (3) [dette er ikke pensum]:

For $c > 1$ og $d > 0$, sæt $N = \log_c(n)$ og $b = c^d$. Så haves

$$\frac{(\log_c n)^a}{n^d} = \frac{N^a}{(c^{\log_c(n)})^d} = \frac{N^a}{c^{d \log_c(n)}} = \frac{N^a}{(c^d)^{\log_c(n)}} = \frac{N^a}{(c^d)^N} = \frac{N^a}{b^N}$$

Da $\log_c(n)$ er en voksende og ubegrænset funktion, gælder at $n \to \infty$ er det samme som $N = \log_c(n) \to \infty$.

Eksempler på funktioner for voksehastighed

Med regel (1)–(4) kan man vise, at følgende funktioner er sat i stigende voksehastighed (mere præcist, at den ene er $o()$ af den næste):

$$1, \quad \log n, \quad \sqrt{n}, \quad n, \quad n \log n,$$

$$n\sqrt{n}, \quad n^2, \quad n^3, \quad n^{10}, \quad 2^n$$

Disse har ret forskellig effektivitet i praksis:

	n	$n \log n$	n^2	n^3	n^{10}	2^n
1 Minut	$6,0 \cdot 10^{10}$	$1,9 \cdot 10^9$	245.000	3.910	12	36
1 Måned	$2,6 \cdot 10^{15}$	$5,7 \cdot 10^{13}$	50.900.000	137.000	35	51

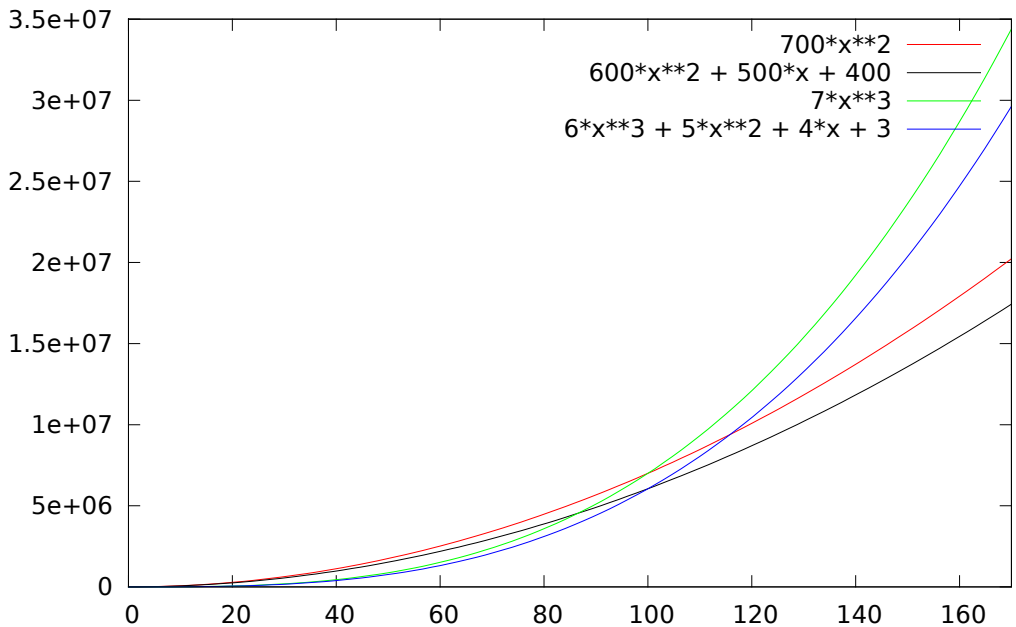
Tabellen viser, hvilke inputstørrelser n man kan klare, hvis algoritmen skal udføre det angivne antal CPU-operationer, og den skal være færdig efter henholdsvis et minut og en måned. Det er antaget, at en CPU kan lave 10^9 operationer per sekund. [Se udleveret program for beregningerne.]

Dominerende led

For funktioner med flere led (dele med plus imellem), vil leddet/ledene med størst voksehastighed bestemme samlet voksehastighed. Eksempel:

$$f(n) = 700n^2 \qquad g(n) = 7n^3$$

$$h(n) = 600n^2 + 500n + 400 \qquad k(n) = 6n^3 + 5n^2 + 4n + 3$$



Figuren passer med beregninger:

Dominerende led

$$\frac{6n^3+5n^2+4n+3}{7n^3} = \frac{6+5/n+4/n^2+3/n^3}{7} \to \frac{6+0+0+0}{7} = 6/7 \text{ for } n \to \infty$$

Dvs. $6n^3+5n^2+4n+3 = \Theta(7n^3)$

$$\frac{600n^2+500n+400}{700n^2} = \frac{600+500/n+400/n^2}{700} \to \frac{600+0+0}{700} = 6/7 \text{ for } n \to \infty$$

Dvs. $600n^2+500n+400 = \Theta(700n^2)$

$$\frac{600n^2+500n+400}{6n^3+5n^2+4n+3} = \frac{600/n+500/n^2+400/n^3}{6+5/n+4/n^2+3/n^3} \to \frac{0+0+0}{6+0+0+0} = 0 \text{ for } n \to \infty$$

Dvs. $600n^2+500n+400 = o(6n^3+5n^2+4n+3)$

Eksempel 1 på asymptotisk analyse af algoritmer

Hvad er den asymptotiske køretid af følgende algoritme?

```
ALGORITME1(n)
    i = 1
    while i ≤ n
        i = i + 2
```

Løkken har $\lceil n/2 \rceil = \Theta(n)$ iterationer.

Hver iteration tager mellem c_1 og c_2 tid for (ukendte) konstanter c_1 og c_2. Dvs. hver iteration tager $\Theta(1)$ tid.

Derfor er køretiden $\Theta(n \cdot 1) = \Theta(n)$.

Vi har her undladt at snakke om tiden for den første linje samt for initialisering af løkken. En mere præcis analyse vil give et udtryk af typen $c_1 \cdot n + c_0$, men vi undlader at snakke om led, som klart er domineret af andre led.

Eksempel 2 på asymptotisk analyse af algoritmer

Hvad er den asymptotiske køretid af følgende algoritme?

```
ALGORITME2(n)
    s = 0
    for i = 1 to n
        for j = i downto 1
            s = s + 1
```

Der er n iterationer af den yderste løkke. For hver af disse er der højst n iterationer af den inderste løkke. Hver iteration af den inderste løkke tager $\Theta(1)$ tid. Så køretiden er $O(n \cdot n \cdot 1) = O(n^2)$.

For $i \geq n/2$ (dvs. for $n/2$ iterationer af den yderste løkke) er der mindst $n/2$ iterationer af den inderste løkke. Så køretiden er $\Omega(n/2 \cdot n/2 \cdot 1) = \Omega(n^2)$.

Derfor er køretiden alt i alt $\Theta(n^2)$.

[Alternativ analyse: den inderste løkke kører $(1 + 2 + 3 + \cdots + n) = (n+1)n/2 = \Theta(n^2)$ gange.]

Eksempel 3 på asymptotisk analyse af algoritmer

Hvad er den asymptotiske køretid af følgende algoritme?

```
ALGORITME3(n)
    s = 0
    for i = 1 to n
        for j = i to n
            for k = i to j
                s = s + 1
```

Der er n iterationer af den yderste løkke. For hver af disse er der højst n iterationer af den midterste løkke. For hver af disse er der højst n iterationer af den inderste løkke. Hver iteration af den inderste løkke tager $\Theta(1)$ tid. Så køretiden er $O(n \cdot n \cdot n \cdot 1) = O(n^3)$.

For $i \leq n/4$ (dvs. for $n/4$ iterationer af den yderste løkke) er der $n/4$ iterationer af den midterste løkke med $j \geq 3n/4$. For disse gælder $j - i \geq n/2$, så for disse har den inderste løkke mindst $n/2$ iterationer. Så køretiden er $\Omega(n/4 \cdot n/4 \cdot n/2 \cdot 1) = \Omega(n^3)$.

Derfor er køretiden alt i alt $\Theta(n^3)$.

Opsummering

Arbejdsprincip: Sammenlign først voksehastigheder af algoritmer via asymtotisk analyse, og implementer normalt kun den med laveste voksehastighed. For to algoritmer med samme voksehastighed, implementer begge og mål deres køretider.