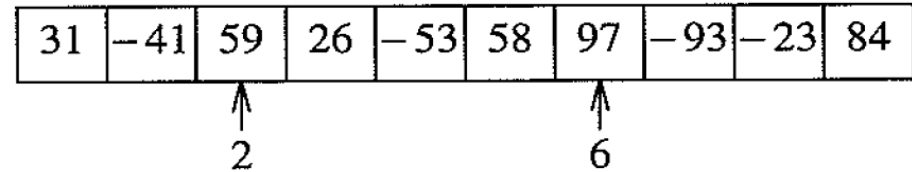
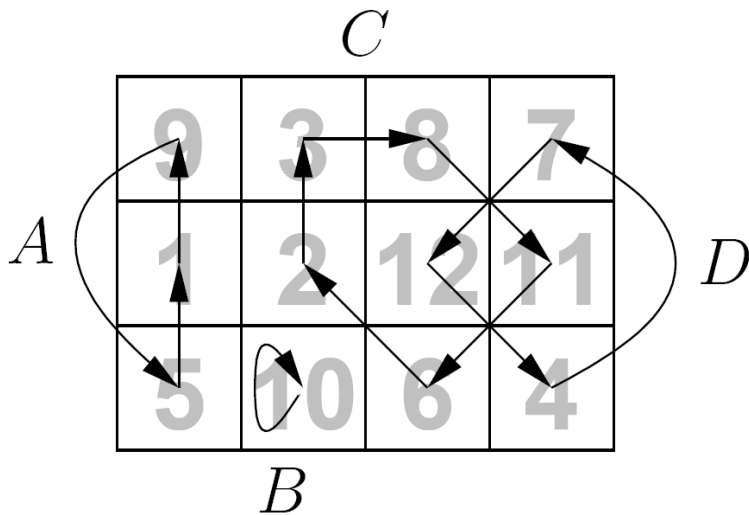


Eksempler på en beregningsprocess...



Maximum delsum

Puslespil ved ombytninger

Max-Delsum: Algoritmiske idéer

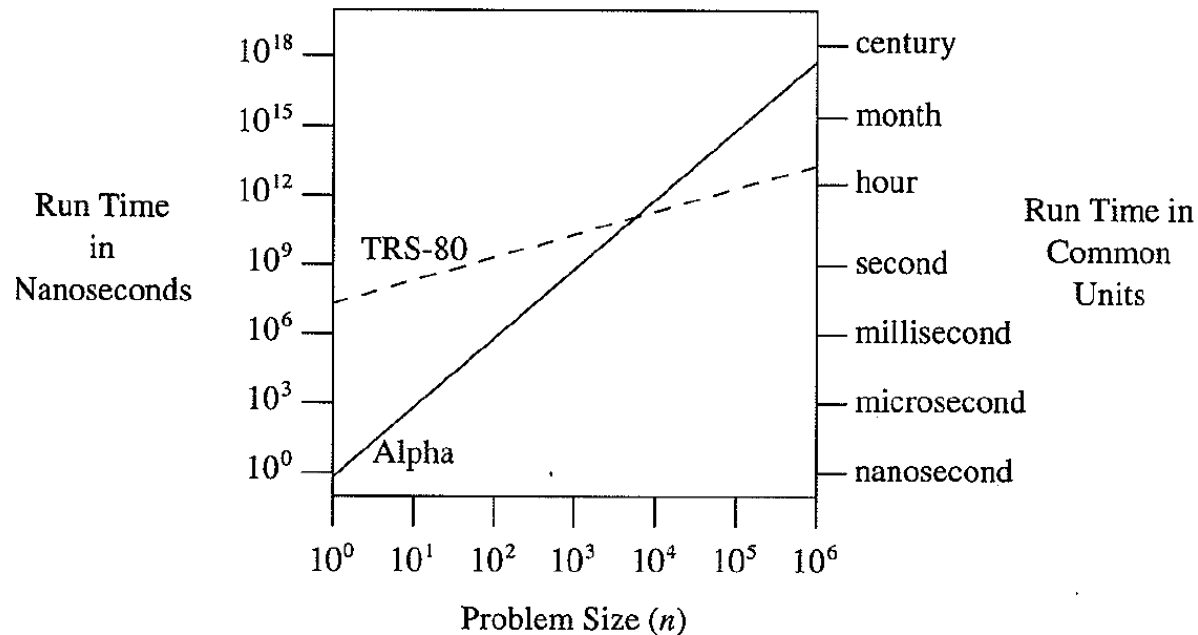
Algoritme	# additioner	Idé
1	$\sim n^3$	Naive løsning
$2 + 2b$	$\sim n^2$	Genbrug beregninger
3	$\sim n \cdot \log n$	Del-og-kombiner
4	$\sim n$	Inkrementel

Sammenligning

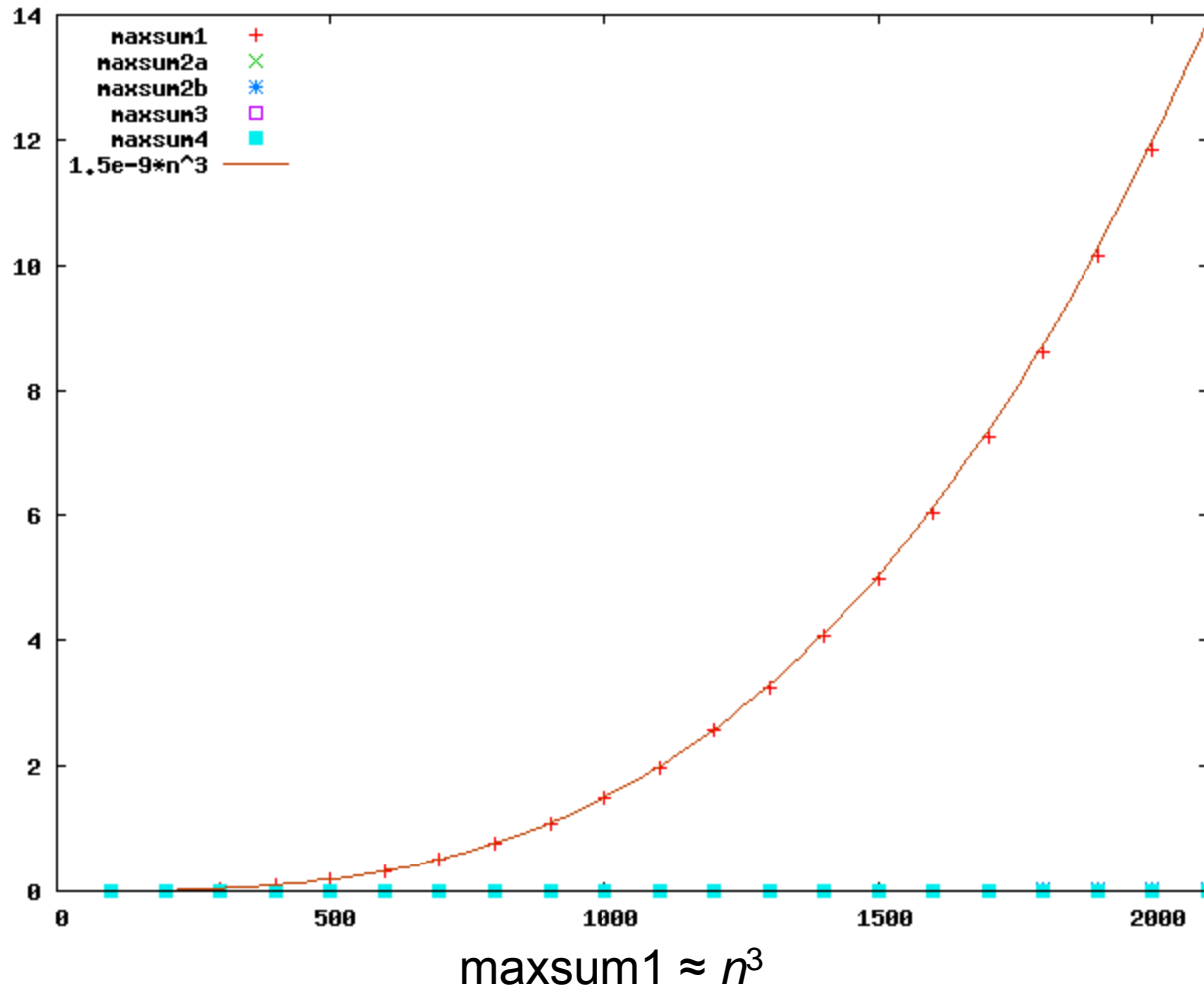
ALGORITHM		1	2	3	4
Run time in nanoseconds		$1.3n^3$	$10n^2$	$47n \log_2 n$	$48n$
Time to solve a problem of size	10^3	1.3 secs	10 msecs	.4 msecs	.05 msecs
	10^4	22 mins	1 sec	6 msecs	.5 msecs
	10^5	15 days	1.7 min	78 msecs	5 msecs
	10^6	41 yrs	2.8 hrs	.94 secs	48 msecs
	10^7	41 millennia	1.7 wks	11 secs	.48 secs
Max size problem solved in one	sec	920	10,000	1.0×10^6	2.1×10^7
	min	3600	77,000	4.9×10^7	1.3×10^9
	hr	14,000	6.0×10^5	2.4×10^9	7.6×10^{10}
	day	41,000	2.9×10^6	5.0×10^{10}	1.8×10^{12}
If n multiplies by 10, time multiplies by		1000	100	10+	10
If time multiplies by 10, n multiplies by		2.15	3.16	10-	10

Sammenligning: n^3 og n

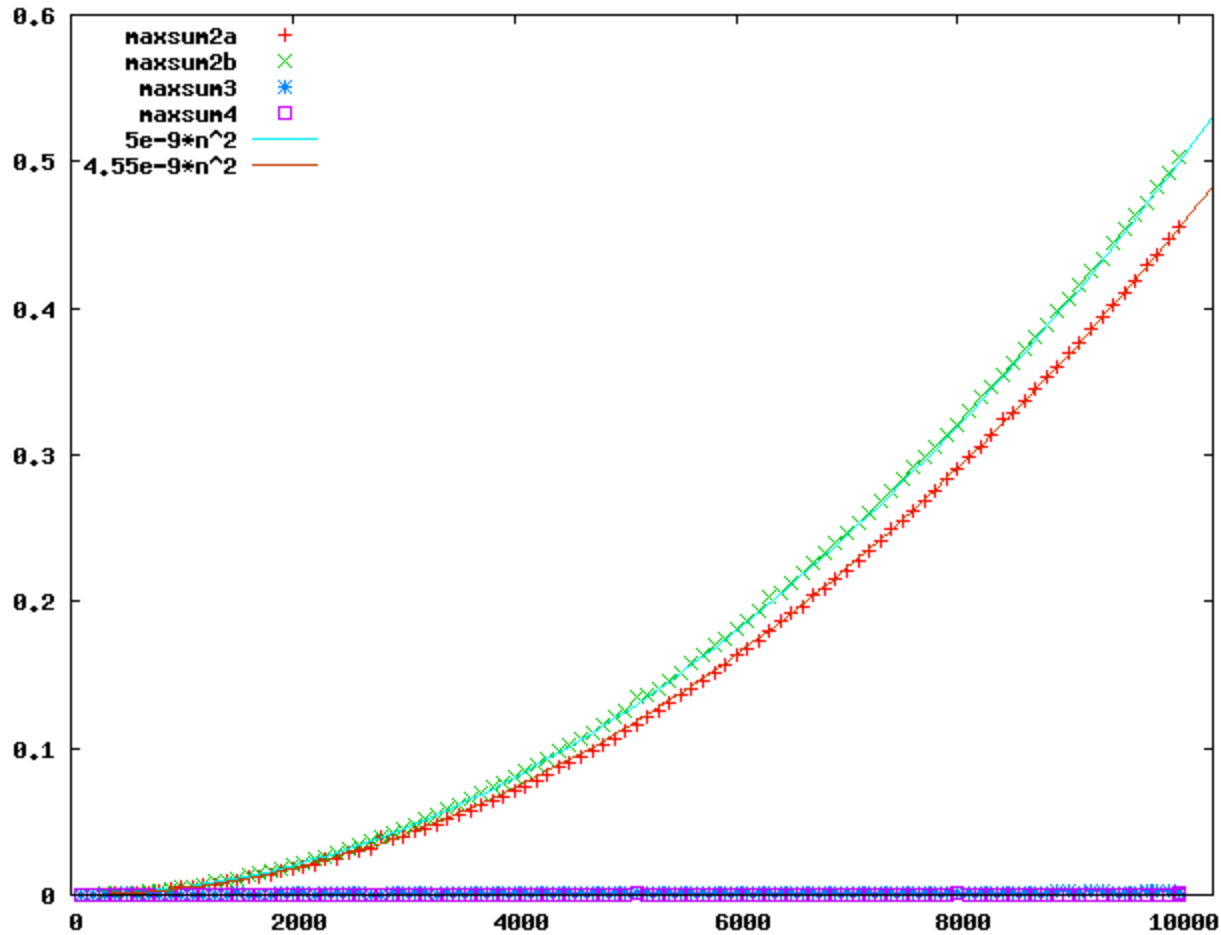
n	ALPHA 21164A, C, CUBIC ALGORITHM	TRS-80, BASIC, LINEAR ALGORITHM
10	0.6 microseconds	200 millisecs
100	0.6 milliseconds	2.0 secs
1000	0.6 secs	20 secs
10,000	10 mins	3.2 mins
100,000	7 days	32 mins
1,000,000	19 yrs	5.4 hrs



Sammenligning 2009

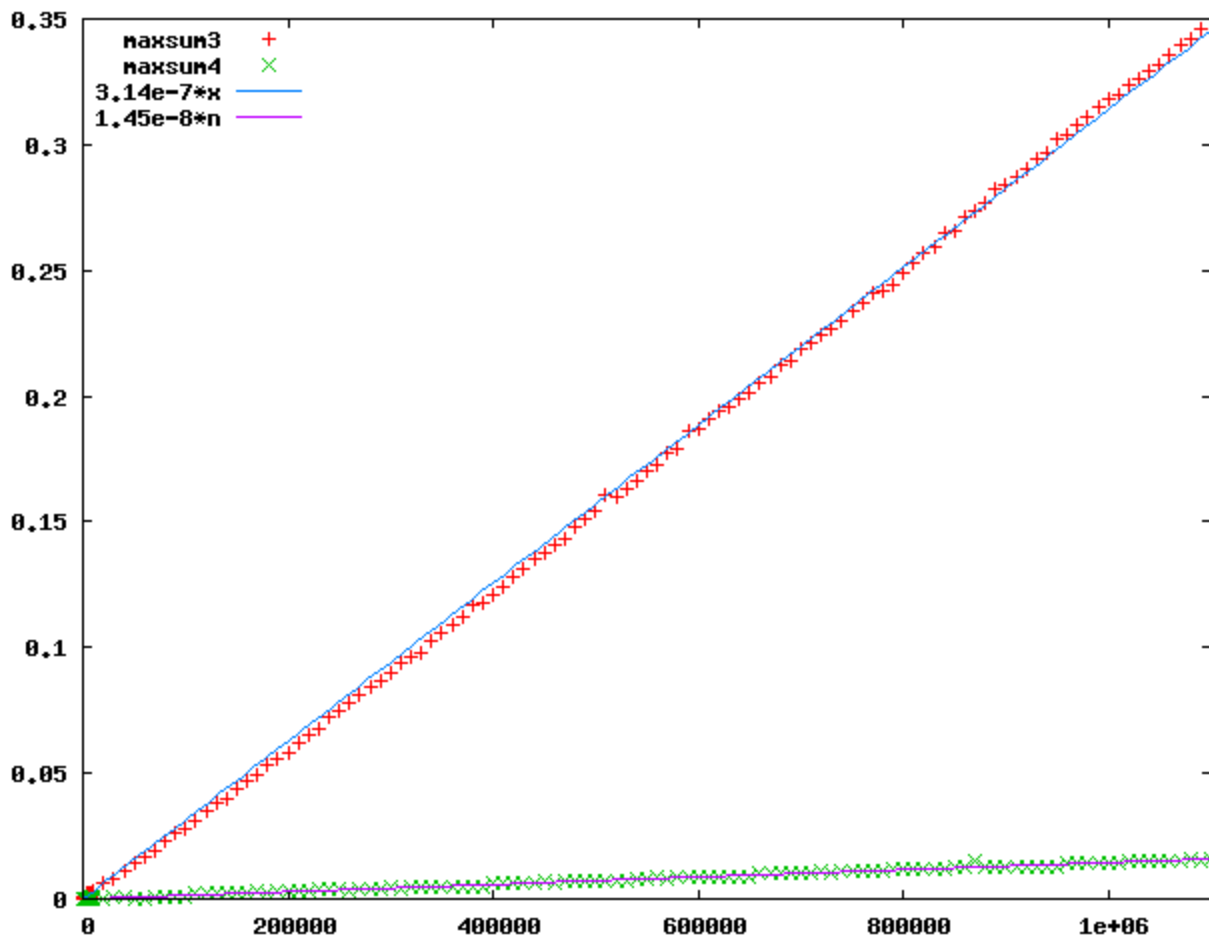


Sammenligning 2009



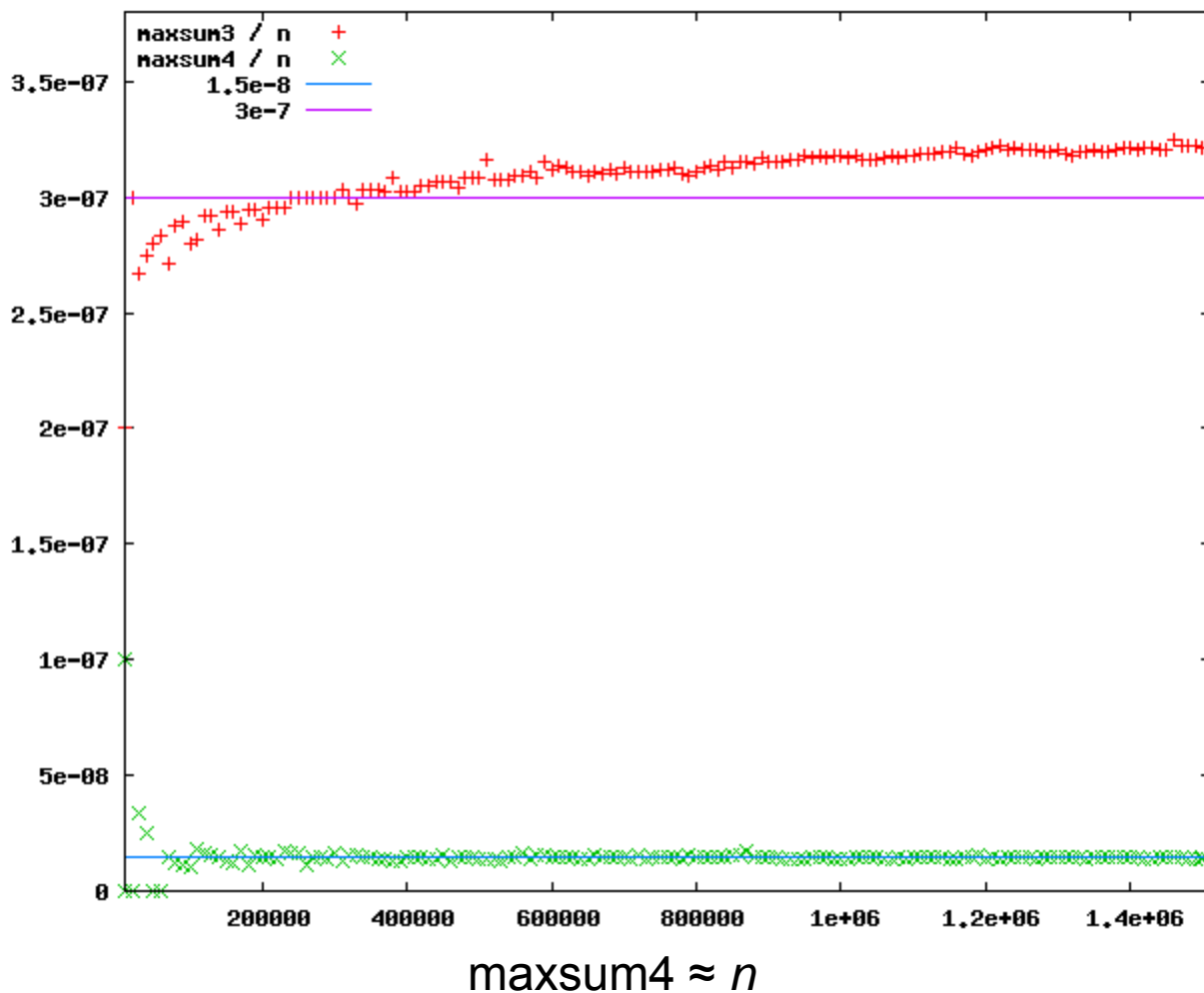
maxsum2a og maxsum2b $\approx n^2$

Sammenligning 2009



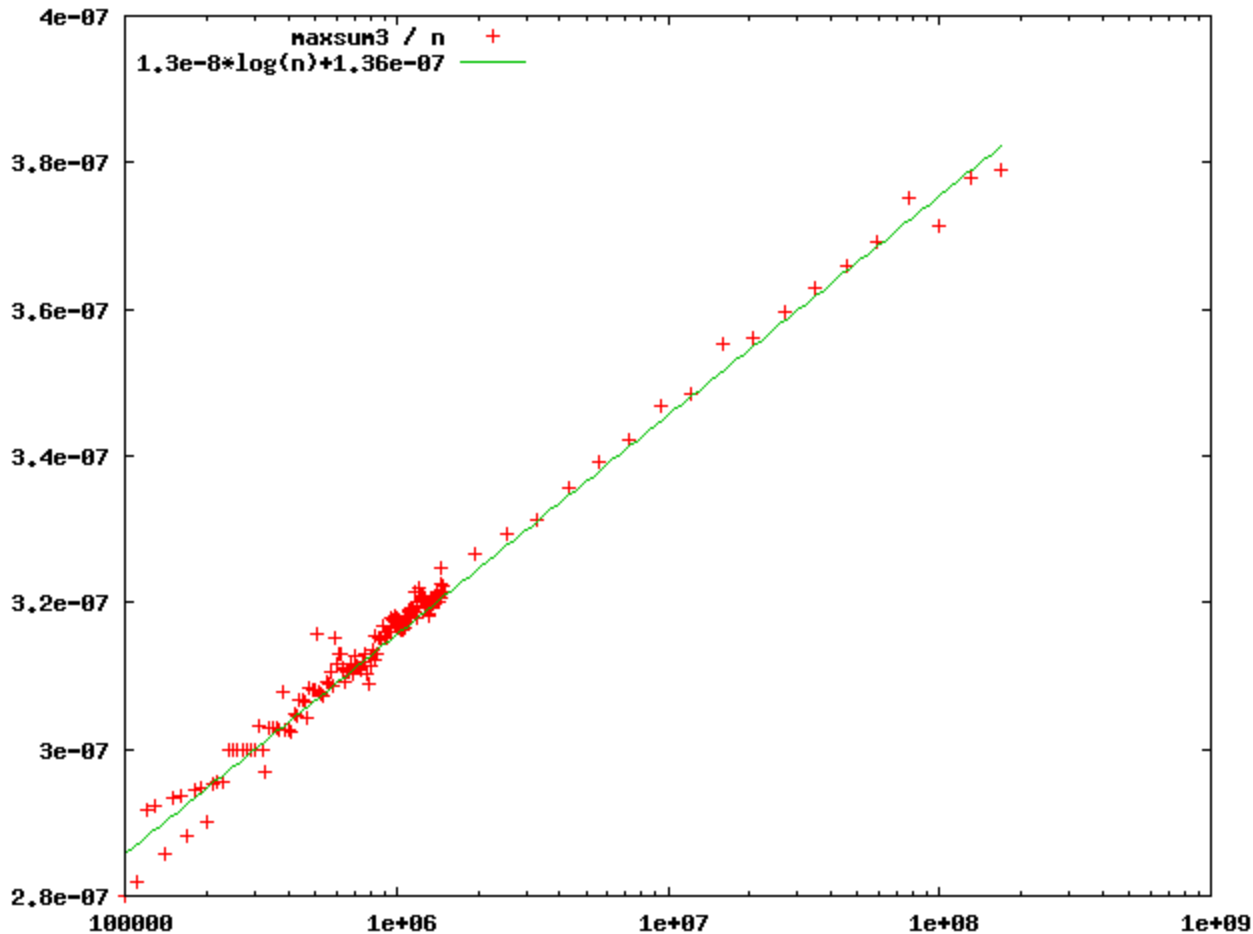
maxsum3 og maxsum4 $\approx n$???

Sammenligning 2009



x-akse = n , y = sekunder, hvert eksperiment gennemsnit af 10 kørsler (gcc 4.1.2, C, Linux 2.6.18, Intel Xeon 3 GHz)

Sammenligning 2009

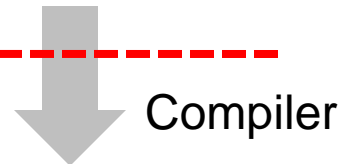
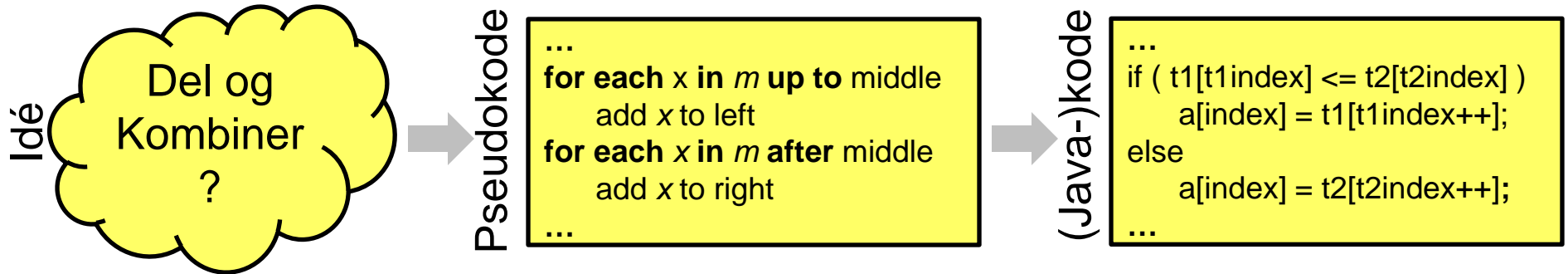


$$\text{maxsum3} \approx c_1 \cdot n \cdot \log n + c_2 \cdot n$$

x-akse = n , y = sekunder, hvert eksperiment gennemsnit af 10 kørsler (gcc 4.1.2, C, Linux 2.6.18, Intel Xeon 3 GHz)

**Hvad er udførelstiden
for en algoritme?**

Fra Idé til Programudførelse



Mikrokode

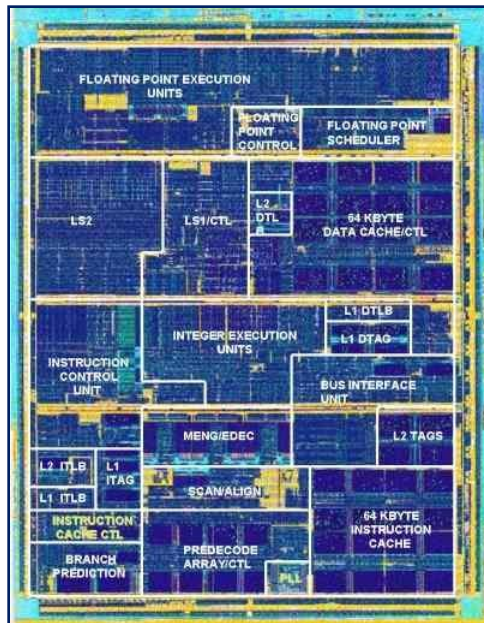
Virtuel hukkomelse/
TLB

L1, L2,... cache

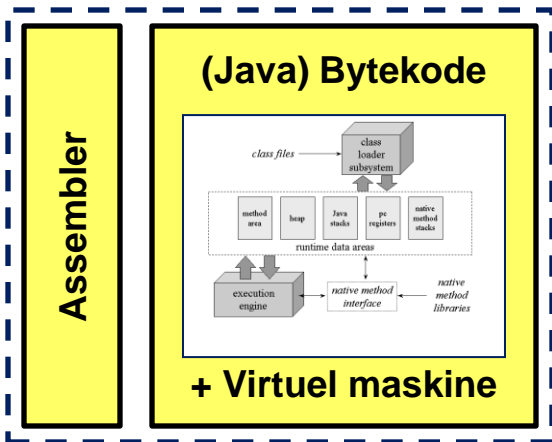
Branch Prediction

Pipelining

...



Maskinkode



Maskiner har forskellig hastighed...

Maskine	Tid (sek)
camel19	8.9
molotov	10.2
harald	26.2
gorm	7.8

Tid for at sortere linierne i en 65 MB web log på forskellige maskiner på Institut for Datalogi

Idé:

Argumenter om algoritmer uafhængig af maskine

Design af Algoritmer

Korrekt algoritme

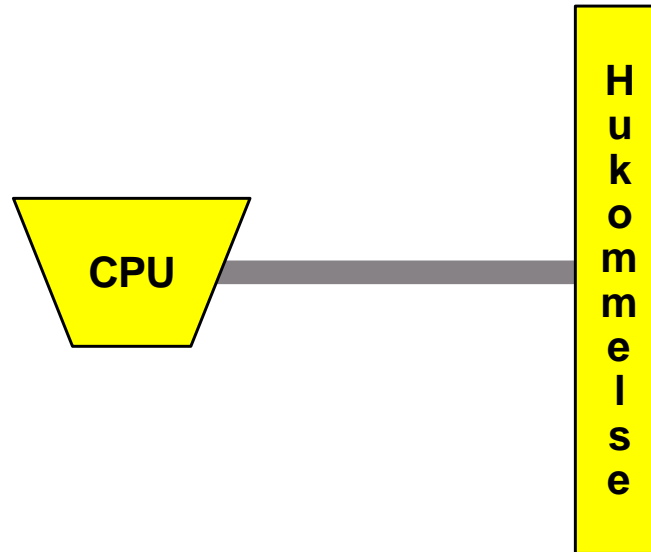
- algoritmen **standser** på alle input
- output er det **rigtige** på alle input

Effektivitet

- Optimer algoritmerne mod at bruge **minimal** tid, plads, additioner,... eller **maximal** parallelisme...
- $\sim n^2$ er bedre end $\sim n^3$: **asymptotisk tid**
- Mindre vigtigt : **konstanterne**
- Resouceforbrug: **Worst-case** eller **gennemsnitlig**?

RAM Modellen

(Random Access Machine)



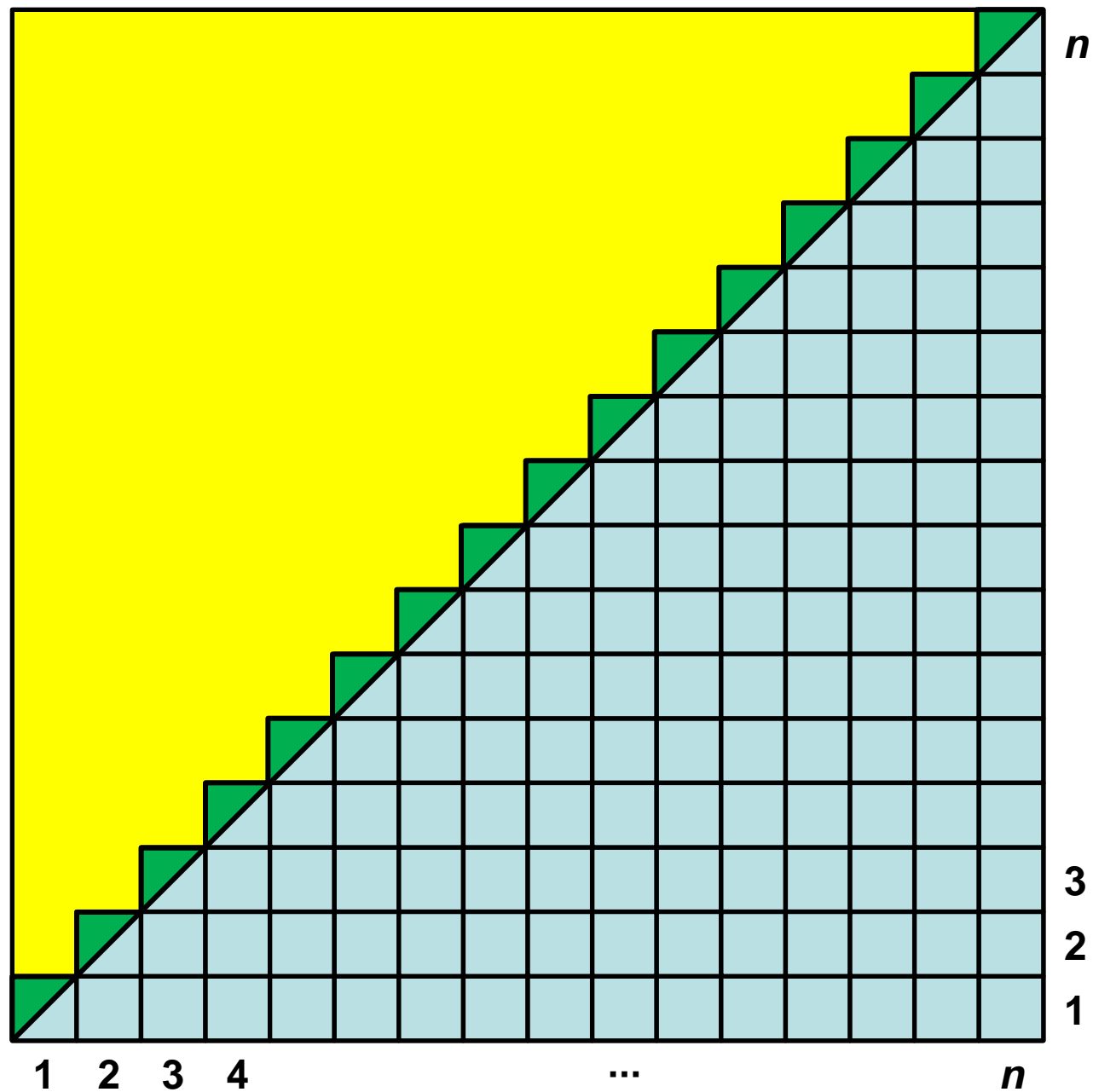
- Beregninger sker i CPU
- Data gemmes i hukommelsen
- Basale operationer tager **1 tidsenhed**:
+, -, *, AND, OR, XOR, **get(i)**, **set(i,v)**, ...
- Et maskinord indeholder **$c \cdot \log n$ bits**

Eksempel: Insertion-Sort

INSERTION-SORT(A)

```
1  for  $j = 2$  to  $A.length$ 
2       $key = A[j]$ 
3      // Insert  $A[j]$  into the sorted sequence  $A[1 .. j - 1]$ .
4       $i = j - 1$ 
5      while  $i > 0$  and  $A[i] > key$ 
6           $A[i + 1] = A[i]$ 
7           $i = i - 1$ 
8       $A[i + 1] = key$ 
```

$$\begin{aligned} 1 + 2 + \dots + n \\ &= \frac{n^2}{2} + \frac{n}{2} \\ &= \frac{n(n+1)}{2} \end{aligned}$$



Insertion-Sort (C)

```
insertion(int a[], int N)
{ int i, j, key;

  for(j=1; j < N; j++)
  { key = a[j];
    i = j-1;
    while( i>=0 && a[i] > key )
      { a[i+1] = a[i];
        i--;
      }
    a[i+1] = key;
  }
}
```

```
insertion:
pushl   %ebp
movl   %esp, %ebp
pushl   %edi
pushl   %esi
pushl   %ebx
subl   $12, %esp
cmpl   $1, 12(%ebp)
jle    .L3
movl   8(%ebp), %edx
xorl   %ebx, %ebx
movl   8(%ebp), %eax
movl   $1, -16(%ebp)
movl   4(%edx), %edx
addl   $4, %eax
movl   %eax, -20(%ebp)
movl   %edx, -24(%ebp)
.p2align 4,,7
.L6:
movl   8(%ebp), %ecx
leal   0(%ebx,4), %esi
movl   (%ecx,%ebx,4), %eax
cmpl   -24(%ebp), %eax
jle    .L8
movl   %ecx, %edi
leal   -4(%esi), %ecx
leal   (%ecx,%edi), %edx
jmp    .L9
.p2align 4,,7
.L16:
movl   (%edx), %eax
movl   %ecx, %esi
subl   $4, %edx
subl   $4, %ecx
cmpl   -24(%ebp), %eax
jle    .L8
.L9:
movl   -20(%ebp), %edi
subl   $1, %ebx
movl   %eax, (%edi,%esi)
jns    .L16
.L8:
movl   -16(%ebp), %edi
movl   8(%ebp), %edx
leal   (%edx,%edi,4), %eax
.L5:
movl   -24(%ebp), %ecx
movl   -20(%ebp), %edx
addl   $1, -16(%ebp)
movl   -16(%ebp), %edi
movl   %ecx, (%edx,%ebx,4)
cmpl   %edi, 12(%ebp)
jle    .L3
movl   4(%eax), %edx
movl   %edi, %ebx
addl   $4, %eax
subl   $1, %ebx
movl   %edx, -24(%ebp)
jns    .L6
jmp    .L5
.L3:
addl   $12, %esp
popl   %ebx
popl   %esi
popl   %edi
popl   %ebp
ret
```

Eksempel: Insertion-Sort

- Eksempel på **pseudo-kode**
- Detaljeret analyse – stort arbejde
- Tid: **worst-case** ($\sim n^2$) og **best-case** ($\sim n$) meget forskellige
- Tid: **gennemsnitlige** ($\sim n^2$)
- Hurtigere på \sim sorterede input: **adaptive**

Asymptotisk notation

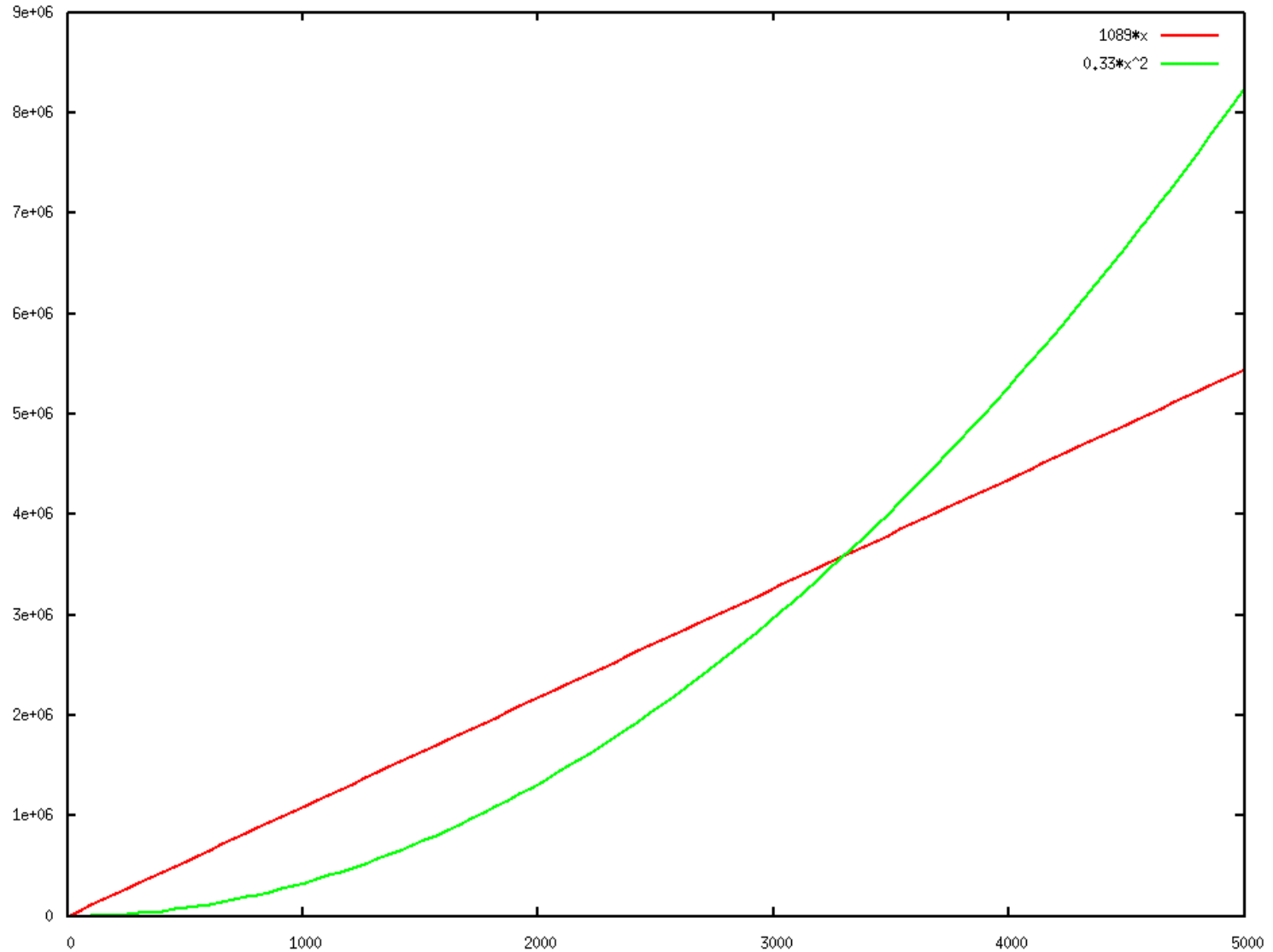
- Grundlæggende antagelse:
 - $\sim n^2$ er bedre end $\sim n^3$
 - Konstanter ikke vigtige
- **Matematisk formel** måde at arbejde med " \sim "
- Eksempler:

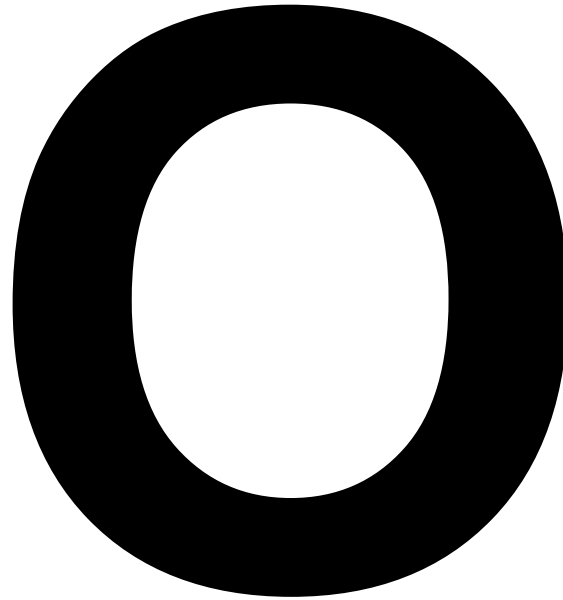
$$87 \cdot n^2 \quad " \leq " \quad 12 \cdot n^3$$

$$1089 \cdot n \quad " \leq " \quad 0.33 \cdot n^2$$

$$7 \cdot n^2 + 25 \cdot n \quad " \leq " \quad n^2$$

$1089 \cdot x$ VS $0.33 \cdot x^2$





- notation

... og vennerne

Ω (store omega)

θ (theta)

ω (lille omega)

o (lille o)

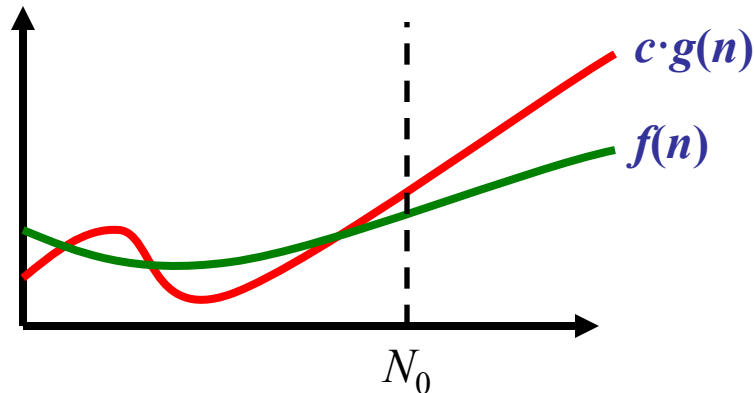
O-notation

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

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

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

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



Intuitivt: $f(n)$ er "mindre end er lig med" $g(n)$, eller $g(n)$ "dominerer" $f(n)$

Eksempel: Insertion-Sort

INSERTION-SORT(A)

```
1  for  $j = 2$  to  $A.length$ 
2       $key = A[j]$ 
3      // Insert  $A[j]$  into the sorted sequence  $A[1 .. j - 1]$ .
4       $i = j - 1$ 
5      while  $i > 0$  and  $A[i] > key$ 
6           $A[i + 1] = A[i]$ 
7           $i = i - 1$ 
8       $A[i + 1] = key$ 
```

Tid $O(n^2)$

Eksempler : O - regneregler

$$f(n) = O(g(n)) \rightarrow c \cdot f(n) = O(g(n))$$

$$f_1(n) = O(g_1(n)) \text{ og } f_2(n) = O(g_2(n)) \rightarrow$$

$$f_1(n) + f_2(n) = O(\max(g_1(n), g_2(n)))$$

$$f_1(n) \cdot f_2(n) = O(g_1(n) \cdot g_2(n))$$

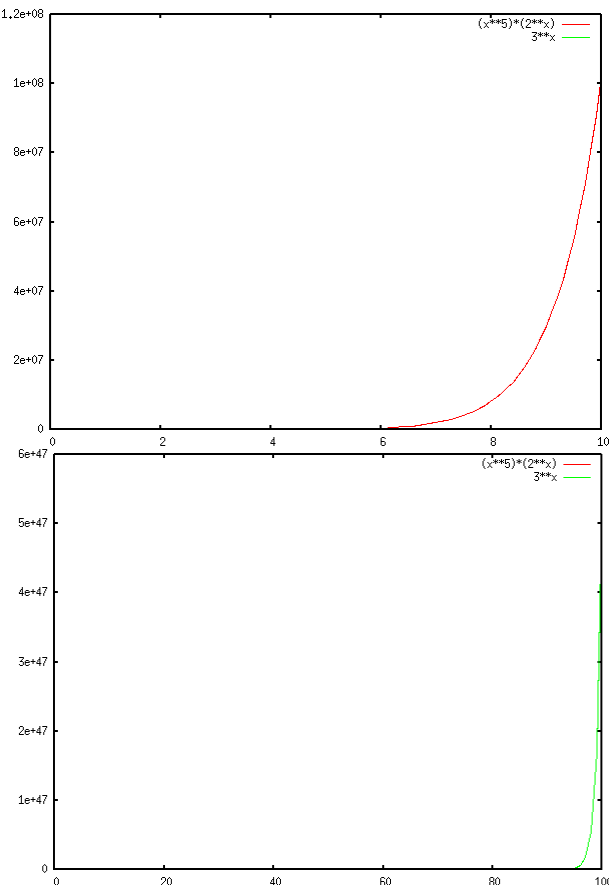
$$c_k \cdot n^k + c_{k-1} \cdot n^{k-1} + \dots + c_2 \cdot n^2 + c_1 \cdot n + c_0 = O(n^k)$$

Eksempler : O

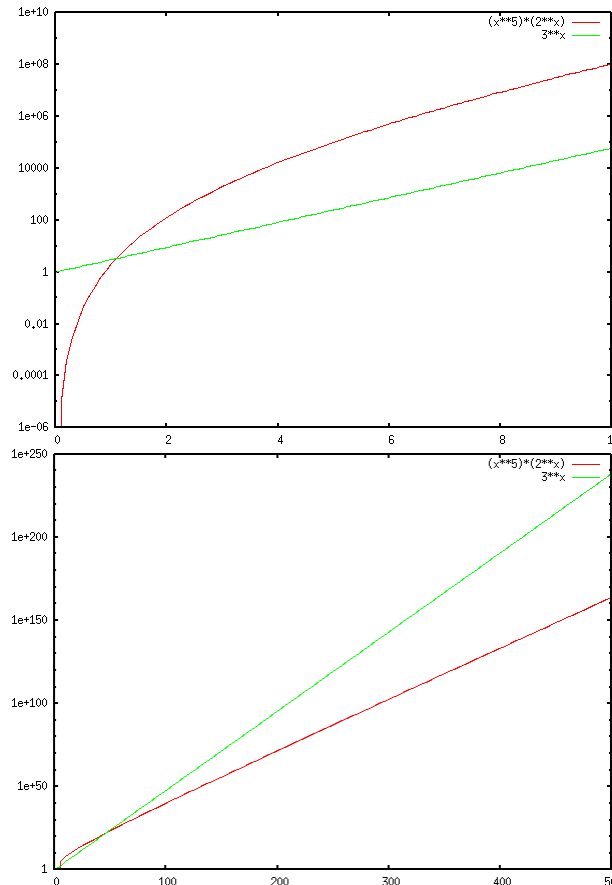
- $3 \cdot n^2 + 7 \cdot n = O(n^2)$
- $n^2 = O(n^3)$
- $\log_2 n = O(n^{0.5})$
- $(\log_2 n)^3 = O(n^{0.1})$
- $n^2 \cdot \log_2 n + 7 \cdot n^{2.5} = O(n^{2.5})$
- $2^n = O(3^n)$
- $n^5 \cdot 2^n = O(3^n)$

Visuel test af $n^5 \cdot 2^n = O(3^n)$?

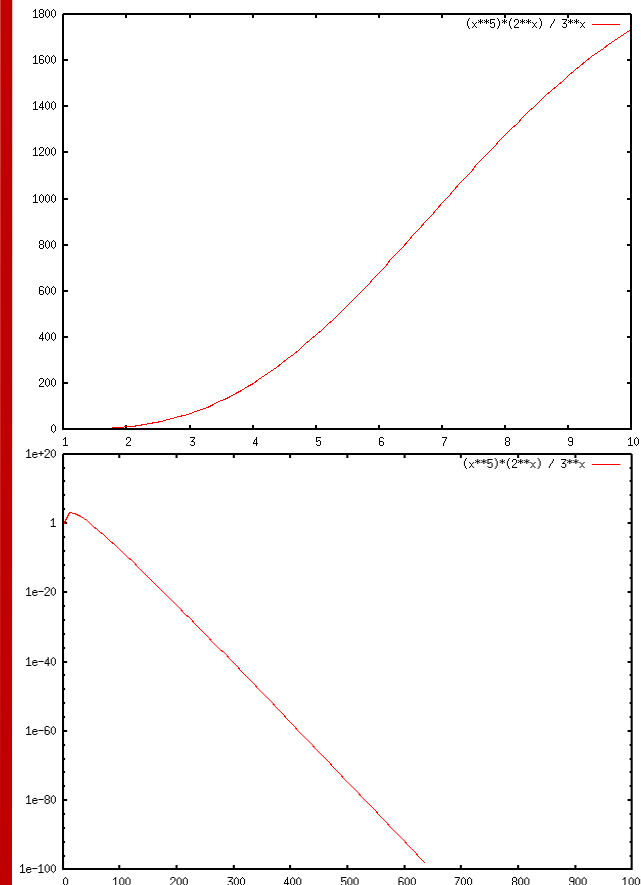
Plot af de to funktioner
– ikke særlig informativ



Plot af de to funktioner
med logaritmisk y-akse
– første plot misvisende



Plot af brøken mellem
de to funktioner
– første plot misvisende



Bevis for $n^5 \cdot 2^n = O(3^n)$

Vis $n^5 \cdot 2^n \leq c \cdot 3^n$ for $n \geq N_0$ for passende valg af c og N_0

Bevis:

$$(5/\log_2(3/2))^2 \leq n \quad \text{for } n \geq 73$$

$$\Downarrow \quad 5/\log_2(3/2) \leq \sqrt{n} = n/\sqrt{n} \leq n/\log_2 n \quad \text{da } \sqrt{n} \geq \log_2 n \text{ for } n \geq 17$$

$$\Downarrow \quad 5 \cdot \log_2 n \leq n \cdot \log_2(3/2)$$

$$\Downarrow \quad \log_2(n^5) \leq \log_2(3/2)^n$$

$$\Downarrow \quad n^5 \leq (3/2)^n$$

$$\Downarrow \quad n^5 \cdot 2^n \leq 3^n$$

Dvs. det ønskede gælder for $c = 1$ og $N_0 = 73$.

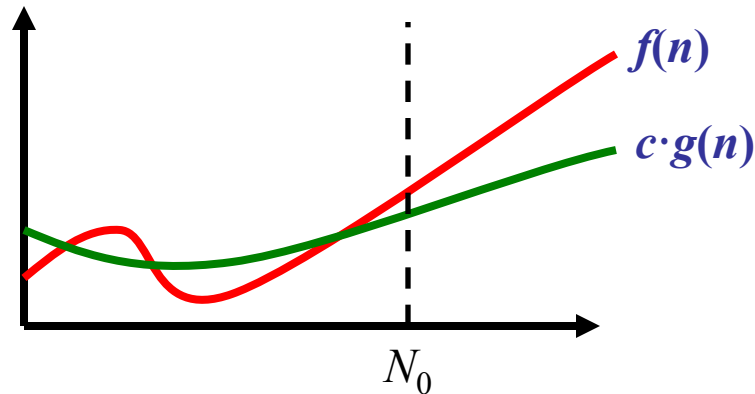


Ω -notation

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 $n \geq N_0$:

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

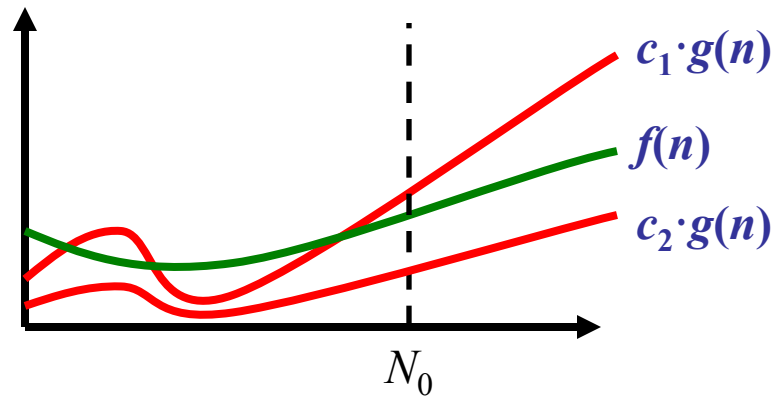


Intuitivt: $f(n)$ er "større end er lig med" $g(n)$, eller $g(n)$ er "domineret af" $f(n)$

θ -notation

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

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



Intuitivt: $f(n)$ og $g(n)$ er "assymptotisk ens"

o-notation (lille o)

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

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

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

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

Intuitivt: $f(n)$ er "skarpt mindre end" $g(n)$

ω -notation

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

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

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

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

Intuitivt: $f(n)$ er "skarpt større end" $g(n)$

Algoritme Analyse

- RAM model
- O-notation

... behøver ikke at beskrive og analysere algoritmer i detaljer !