

<https://xkcd.com/1667/>

# Tidskompleksitet og asymptotisk notation

Algoritmer og Datastrukturer  
Københavns Universitet, forår 2024

Rasmus Pagh





# Vejledende løsninger

Ugeopgaver

Opgavesamling

CLRS solution manual

Udvalgte andre opgaver

# Plan for i dag

- Hvad gør en algoritme god?
- Beregningsmodeller
- Case study: Insertion sort
- Asymptotisk notation
- Nogle grundlæggende funktioner brugt i kurset

# Hvad gør en algoritme god?

- **Korrekthed:** På ethvert input returneres et korrekt output
- **Hastighed:** Antal beregningsskridt for at nå frem til output skal være lille
- **Pladseffektivitet:** Mængden af hukommelse, algoritmen bruger, skal være lille
- **Enkelhed:** Kan implementeres korrekt
- **Fleksibilitet:** Kan let tilpasses hvis problemformuleringen ændres
- ...
- Vi siger at en algoritme er **optimal** hvis det ikke er muligt at forbedre den

# Eksempel: Binær søgning

- Kode fra Wikipedia (Pascal syntax):

```
function binary_search(A, n, T) is
    L := 1
    R := n
    while L ≤ R do
        m := floor((L + R) / 2)
        if A[m] < T then
            L := m + 1
        else if A[m] > T then
            R := m - 1
        else:
            return m
    return unsuccessful
```

## Input:

A: Sorteret array, elementer  $A[1], \dots, A[n]$

n: Antal elementer i A

T: Et element, der skal søges efter

## Output:

Index m hvor  $A[m] = T$  hvis det findes, ellers unsuccessful.

- Korrekthed, hastighed, pladseffektivitet?

# Kørsel af binary\_search(A, 7, 10)

- Kode fra Wikipedia (Pascal syntax):

```
function binary_search(A, n, T) is
    L := 1
    R := n
    while L ≤ R do
        m := floor((L + R) / 2)
        if A[m] < T then
            L := m + 1
        else if A[m] > T then
            R := m - 1
        else:
            return m
    return unsuccessful
```

Registre

--	--	--	--	--

Stak

103	7	10				
-----	---	----	--	--	--	--

A    n    T    L    R    m



2	3	5	7	11	13	17
---	---	---	---	----	----	----

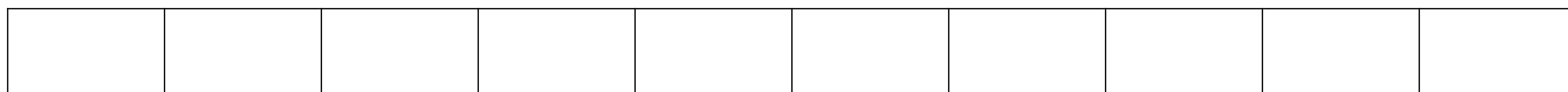
Heap

# Computere er komplekse



# Beregningsmodeller

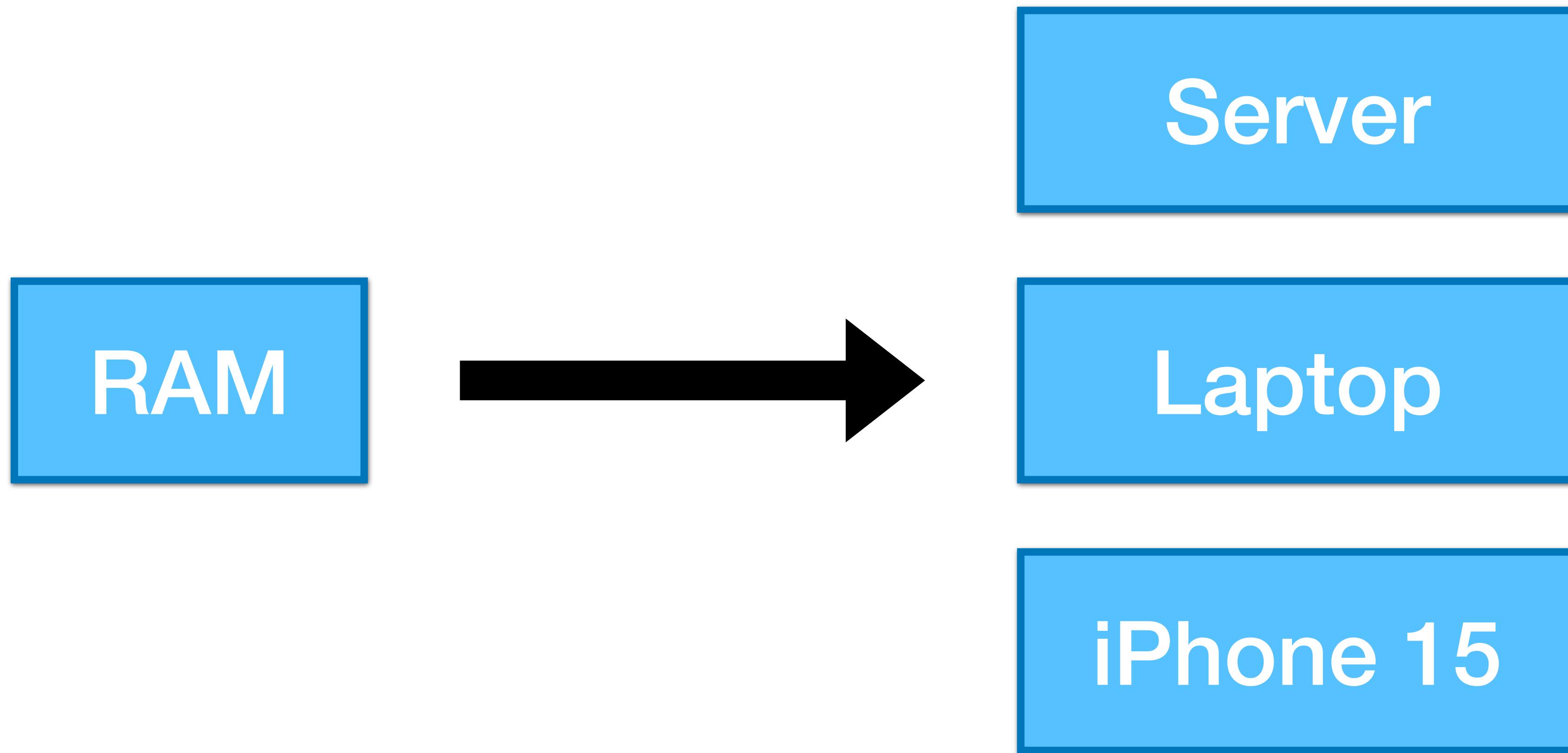
- Vi vil gerne sige noget om en algoritme *uafhængigt* af den konkrete computer, den udføres på
- Random Access Machine (RAM) er en simpel abstraktion af en computer:
  - Har et konstant antal *registre* til at gemme mellemresultater ( $w$  bits i hver)
  - Hukommelsen er en tabel af *maskinord*, der hver består af  $w$  bits
- Input til en algoritme ligger i de første  $n$  maskinord i hukommelsen



# Køretid i RAM modellen

- Kan i ét tidsskridt: Udføre en aritmetisk operation, læse eller skrive et maskinord, udføre logiske operationer lave betinget hop (branching), ...
- Køretiden for en RAM algoritme = antal tidsskridt indtil den terminerer
- Værstefald (worst case) køretid (for inputstørrelse  $n$ ), er *maksimum* af køretiden på alle input af størrelse  $n$
- Når vi taler om “køretid af en algoritme” mener vi oftest værstefalds analysen

# Analyser kan overføres



# Hvad RAM modellen abstraherer væk

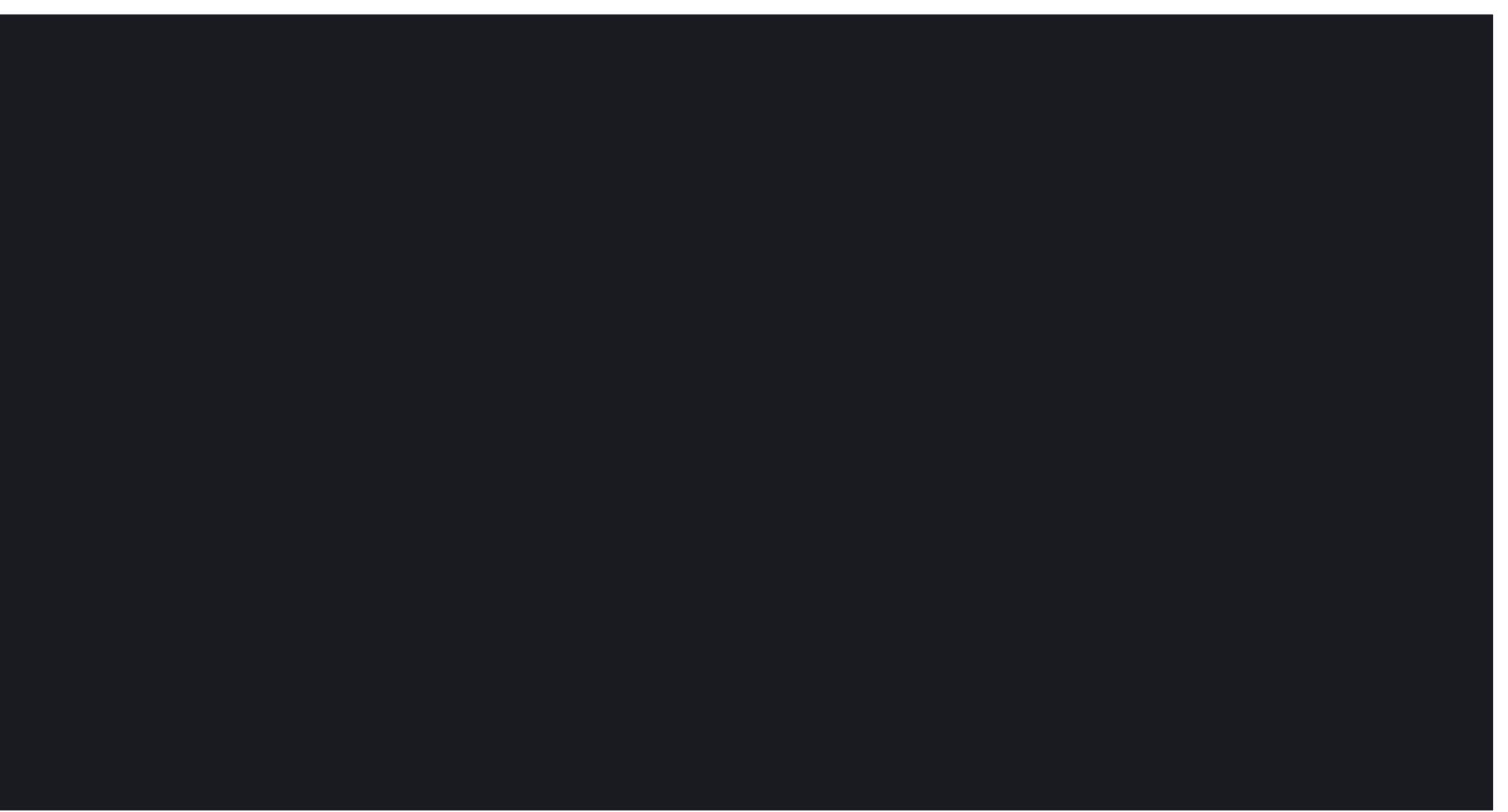
- Nogle maskinoperationer gør > 1 ting (SIMD, compare-swap, ...)
- Uafhængige operationer kan ofte køre parallelt
- Hukommelsesadgange er ikke lige hurtige (cache vs hukommelse vs SSD)
- Computere har flere kerner, der kan arbejde parallelt
- Computere har flere *typer* kerner (typisk CPU(er), GPU, måske FPGA, TPU)
- Store beregninger kan fordeles over mange computere
- ...
- **Fokus i kurset er RAM:**  
1-trådet asymptotisk skalérbarhed, ikke optimering, parallelisme, etc.

# Case study: Insertion sort

```
INSERTION-SORT( $A, n$ )
```

```
for  $j = 2$  to  $n$ 
     $key = A[j]$ 
    // Insert  $A[j]$  into the sorted sequence  $A[1 \dots j - 1]$ .
     $i = j - 1$ 
    while  $i > 0$  and  $A[i] > key$ 
         $A[i + 1] = A[i]$ 
         $i = i - 1$ 
     $A[i + 1] = key$ 
```

Animation from [algostructure.com](http://algostructure.com)



# Case study: Insertion sort

```
INSERTION-SORT( $A, n$ )
```

```
    for  $j = 2$  to  $n$ 
         $key = A[j]$ 
        // Insert  $A[j]$  into the sorted sequence  $A[1 \dots j - 1]$ .
         $i = j - 1$ 
        while  $i > 0$  and  $A[i] > key$ 
             $A[i + 1] = A[i]$ 
             $i = i - 1$ 
         $A[i + 1] = key$ 
```

	<i>cost</i>	<i>times</i>
$c_1$	$n$	
$c_2$	$n - 1$	
$0$	$n - 1$	
$c_4$	$n - 1$	
$c_5$	$\sum_{j=2}^n t_j$	
$c_6$	$\sum_{j=2}^n (t_j - 1)$	
$c_7$	$\sum_{j=2}^n (t_j - 1)$	
$c_8$	$n - 1$	

Køretid:  $c_1n + c_2(n - 1) + c_4(n - 1) + c_5 \sum_{j=2}^n t_j + c_6 \sum_{j=2}^n (t_j - 1) + c_7 \sum_{j=2}^n (t_j - 1) + c_8(n - 1)$

# Øvelse

- Hvilket input giver værsteafalds (størst mulige) køretid på et input af længde  $n$ ?
- Hvad er  $t_j$  på dette input?
- (Hvad er køretiden som funktion af  $n$ ?)

INSERTION-SORT( $A, n$ )

```
for  $j = 2$  to  $n$ 
    key =  $A[j]$ 
    // Insert  $A[j]$  into the sorted sequence  $A[1 \dots j - 1]$ .
     $i = j - 1$ 
    while  $i > 0$  and  $A[i] > key$ 
         $A[i + 1] = A[i]$ 
         $i = i - 1$ 
     $A[i + 1] = key$ 
```

Køretid:  $c_1n + c_2(n - 1) + c_4(n - 1) + c_5 \sum_{j=2}^n t_j + c_6 \sum_{j=2}^n (t_j - 1) + c_7 \sum_{j=2}^n (t_j - 1) + c_8(n - 1)$

# Værstefalds analyse af insertion sort

Køretid med  $t_j = j$ :

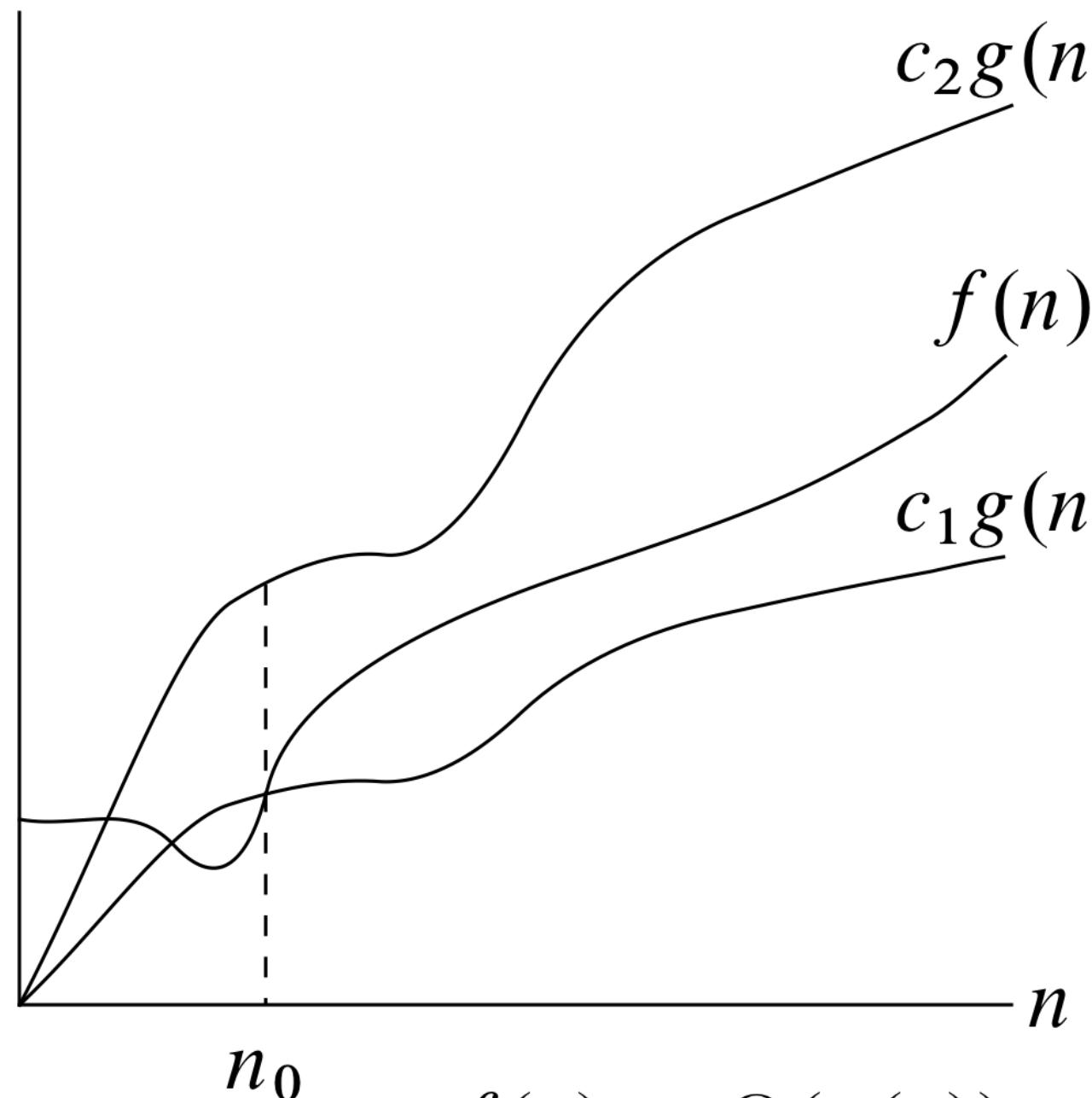
$$\begin{aligned} & c_1n + c_2(n - 1) + c_4(n - 1) + c_5 \sum_{j=2}^n t_j + c_6 \sum_{j=2}^n (t_j - 1) + c_7 \sum_{j=2}^n (t_j - 1) + c_8(n - 1) \\ & \leq c_1n + c_2(n - 1) + c_4(n - 1) + c_5 \sum_{j=2}^n j + c_6 \sum_{j=2}^n (j - 1) + c_7 \sum_{j=2}^n (j - 1) + c_8(n - 1) \\ & = \left( \frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2} \right) n^2 + \left( c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8 \right) n - (c_2 + c_4 + c_5 + c_8) \\ & = a_2n^2 + a_1n + a_0 \end{aligned}$$

for passende konstanter  $a_0, a_1, a_2$ .

# Asymptotisk notation

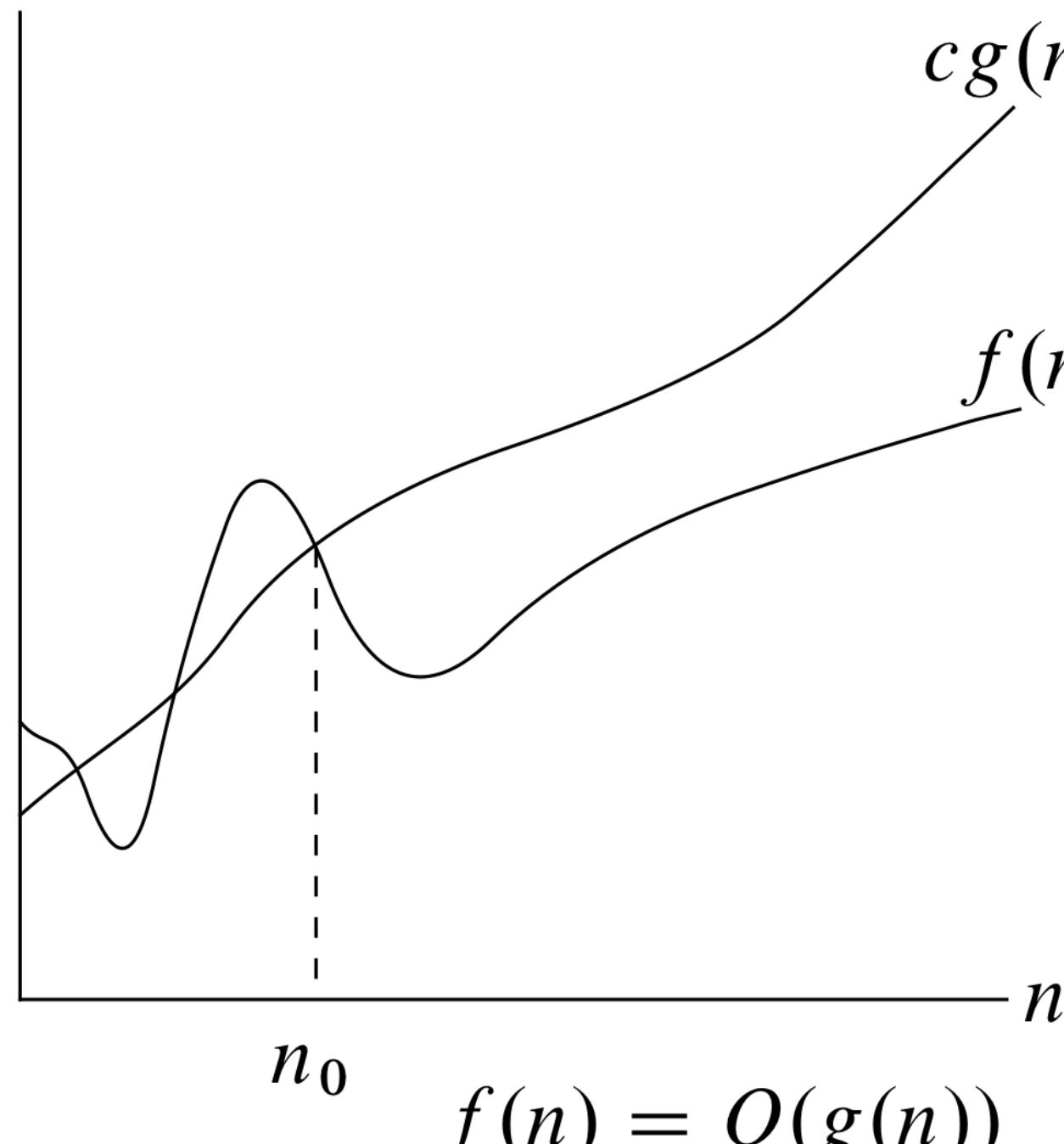
- **Idé:**
  - Abstrahér detaljerne væk, fokusér på hvad der sker når  $n$  er stor
  - Ignorér konstante faktorer (det gør RAM modellen allerede)
- Lad  $T(n)$  betegne **værstefalds køretiden** på et input af længde  $n$  – vi så at  $T(n) \leq a_2n^2 + a_1n + a_0$  for passende konstanter  $a_0, a_1, a_2$
- For store  $n$  er denne grænse *domineret* af  $a_2n^2$ , mens de to *lavere-ordens termer*  $a_1n$  og  $a_0$  er ubetydelige
- I *asymptotisk notation* er værstefalds køretiden  $\Theta(n^2)$

# Asymptotisk notation i et billede



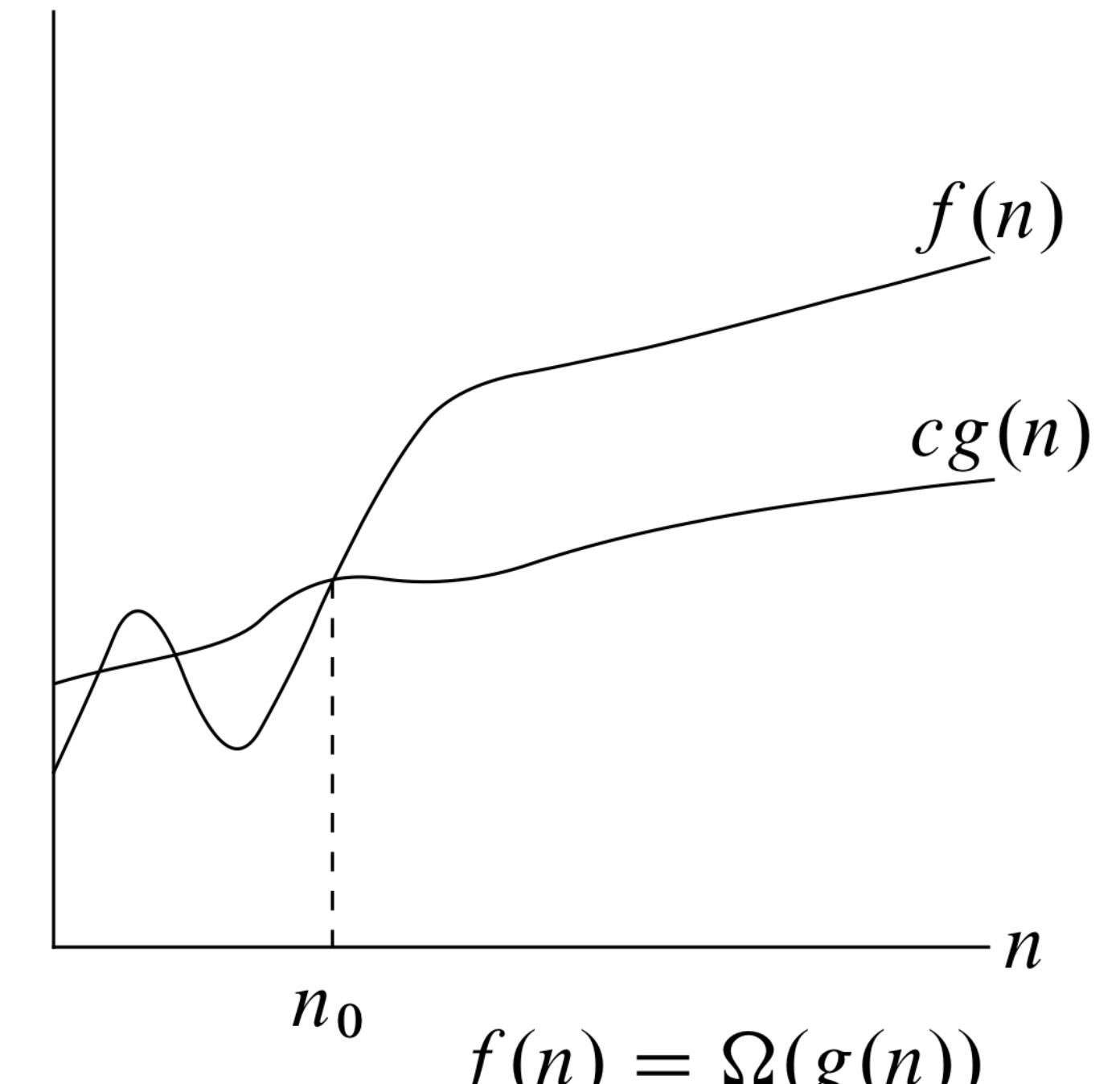
$$f(n) = \Theta(g(n))$$

“=”



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

“<”



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

“>”

# Formelle definitioner

- **Formelle definitioner:** For en ikke-negativ funktion  $g(n) : \mathbb{N} \rightarrow \mathbb{R}$  er

$$O(g(n)) = \{f(n) \mid \exists c > 0, n_0 > 0 : n \geq n_0 \Rightarrow 0 \leq f(n) \leq cg(n)\}$$

$$\Omega(g(n)) = \{f(n) \mid \exists c > 0, n_0 > 0 : n \geq n_0 \Rightarrow cg(n) \leq f(n)\}$$

$$\Theta(g(n)) = \{f(n) \mid \exists c_1, c_2 > 0, n_0 > 0 : n \geq n_0 \Rightarrow c_1g(n) \leq f(n) \leq c_2g(n)\}$$

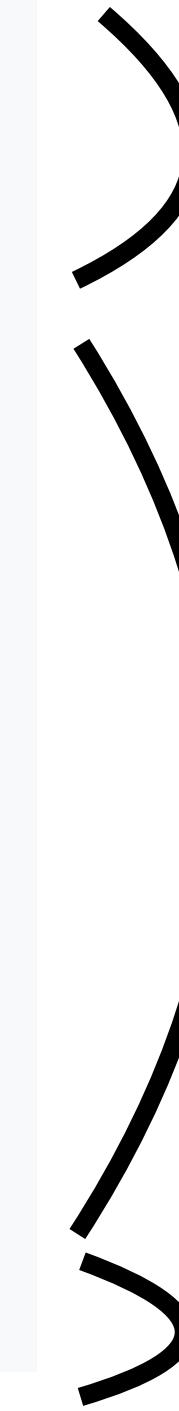
- I praksis bruges notationen oftest til at betegne en (ikke navngivet) funktion, fx:
  - $g(n) = O(n^2)$  betyder  $g(n) = f(n)$  for en eller anden  $f(n) \in O(n^2)$
  - $g(n) = n^2 + \Omega(n)$  betyder  $g(n) = n^2 + f(n)$  for en eller anden  $f(n) \in \Omega(n)$

# Tilbage til insertion sort

- Værstefalds køretiden er  $T(n) \leq a_2 n^2 + a_1 n + a_0$  for konstanter  $a_0, a_1, a_2$
- Det betyder at  $T(n) \leq an^2$  for  $n \geq 1$  hvis vi vælger  $a = |a_0| + |a_1| + |a_2|$
- Så  $T(n) \in O(n^2)$
- ... eller som man oftere skriver,  $T(n) = O(n^2)$
- **NB!** Det er også sandt at  $T(n) = O(n^3)$ , eller at  $T(n) = O(2^n)$ , eller ..., men det er ikke nogen præcis analyse

# Tilbage til binær søgning

```
function binary_search(A, n, T) is
    L := 1
    R := n
    while L ≤ R do
        m := floor((L + R) / 2)
        if A[m] < T then
            L := m + 1
        else if A[m] > T then
            R := m - 1
        else:
            return m
    return unsuccessful
```



Tid  $O(1)$ , kører 1 gang

Tid  $O(1)$   
/iteration

Tid  $O(1)$ , kører højst 1 gang

Kører højst  $\lg(n) + 1$  gange fordi  
 $R - L + 1$  bliver bliver (mindst)  
halveret i hver iteration og løkken  
terminerer når  $R - L + 1 < 1$

- Kan vi konkludere at køretiden er  $O(\log n)$ ?

# Store-O og addition

- Antag at  $f(n) = O(g(n))$  og  $f'(n) = O(g'(n))$  er ikke-negative, dvs.
  - Der findes  $n_0, c > 0$  så  $n \geq n_0 \Rightarrow f(n) \leq cg(n)$
  - Der findes  $n'_0, c' > 0$  så  $n \geq n'_0 \Rightarrow f'(n) \leq c'g'(n)$
  - Hvad kan vi sige om  $f(n) + f'(n)$ ?
  - $n \geq \max(n_0, n'_0) \Rightarrow f(n) + f'(n) \leq cg(n) + c'g(n) \leq \max(c, c') \cdot (g(n) + g'(n))$
  - Det vil sige at  $f(n) + f'(n) = O(g(n) + g'(n))$

# Øvelse: Store-O og multiplikation

- Antag at  $f(n) = O(g(n))$  og  $f'(n) = O(g'(n))$  er ikke-negative, dvs.
  - Der findes  $n_0, c > 0$  så  $n \geq n_0 \Rightarrow f(n) \leq cg(n)$
  - Der findes  $n'_0, c' > 0$  så  $n \geq n'_0 \Rightarrow f'(n) \leq c'g'(n)$
  - Hvad kan vi sige om  $f(n) \cdot f'(n)$ ?

# Simplere analyse af insertion sort

- **for-løkken** laver  $O(n)$  iterationer, der hver tager  $O(1)$  tid, bortset fra tiden i **while-løkken**
- I hver iteration af **for-løkken** laver **while-løkken**  $O(n)$  iterationer, der hver tager  $O(1)$  tid, totalt  $O(1) + O(n) \cdot O(1)$  tid
- Den totale tid for alle kørsler af **while-løkken** er  $O(n) \cdot O(n) = O(n^2)$
- For *omvendt sorteret* input tager brugeren algoritmen tid  $\Omega(n) \cdot \Omega(n) = \Omega(n^2)$

```
INSERTION-SORT( $A, n$ )
```

```
  for  $j = 2$  to  $n$ 
     $key = A[j]$ 
    // Insert  $A[j]$  into the sorted sequence  $A[1 \dots j - 1]$ .
     $i = j - 1$ 
    while  $i > 0$  and  $A[i] > key$ 
       $A[i + 1] = A[i]$ 
       $i = i - 1$ 
     $A[i + 1] = key$ 
```

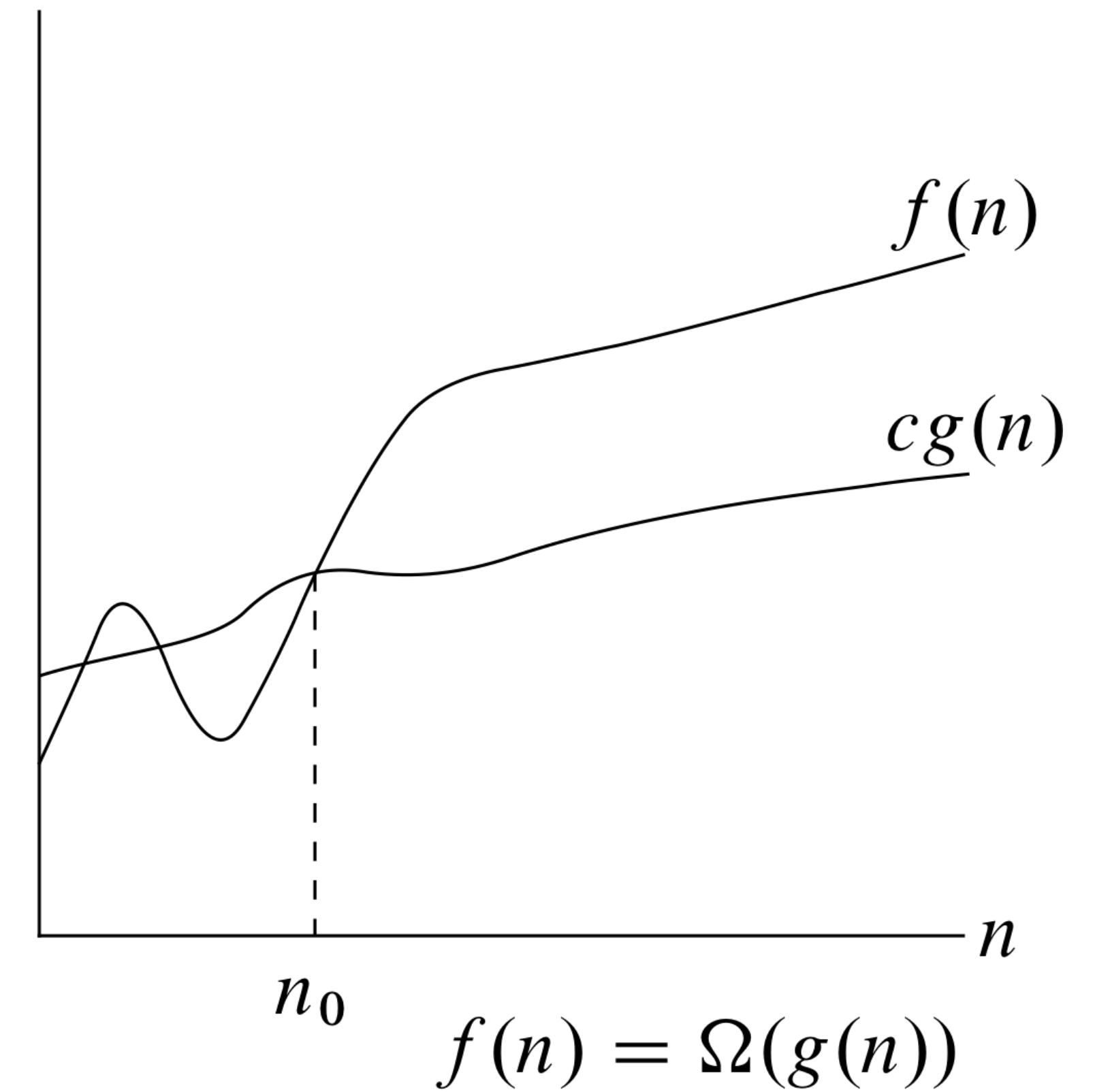
# Store-O og subtraktion, division

- Antag at  $f(n) = O(g(n))$  og  $f'(n) = O(g'(n))$  er ikke-negative
- Hvad kan vi sige om  $f(n) - f'(n)$ ?
- ... næsten ingenting
- Hvad kan vi sige om  $f(n)/f'(n)$ ?
- ... næsten ingenting



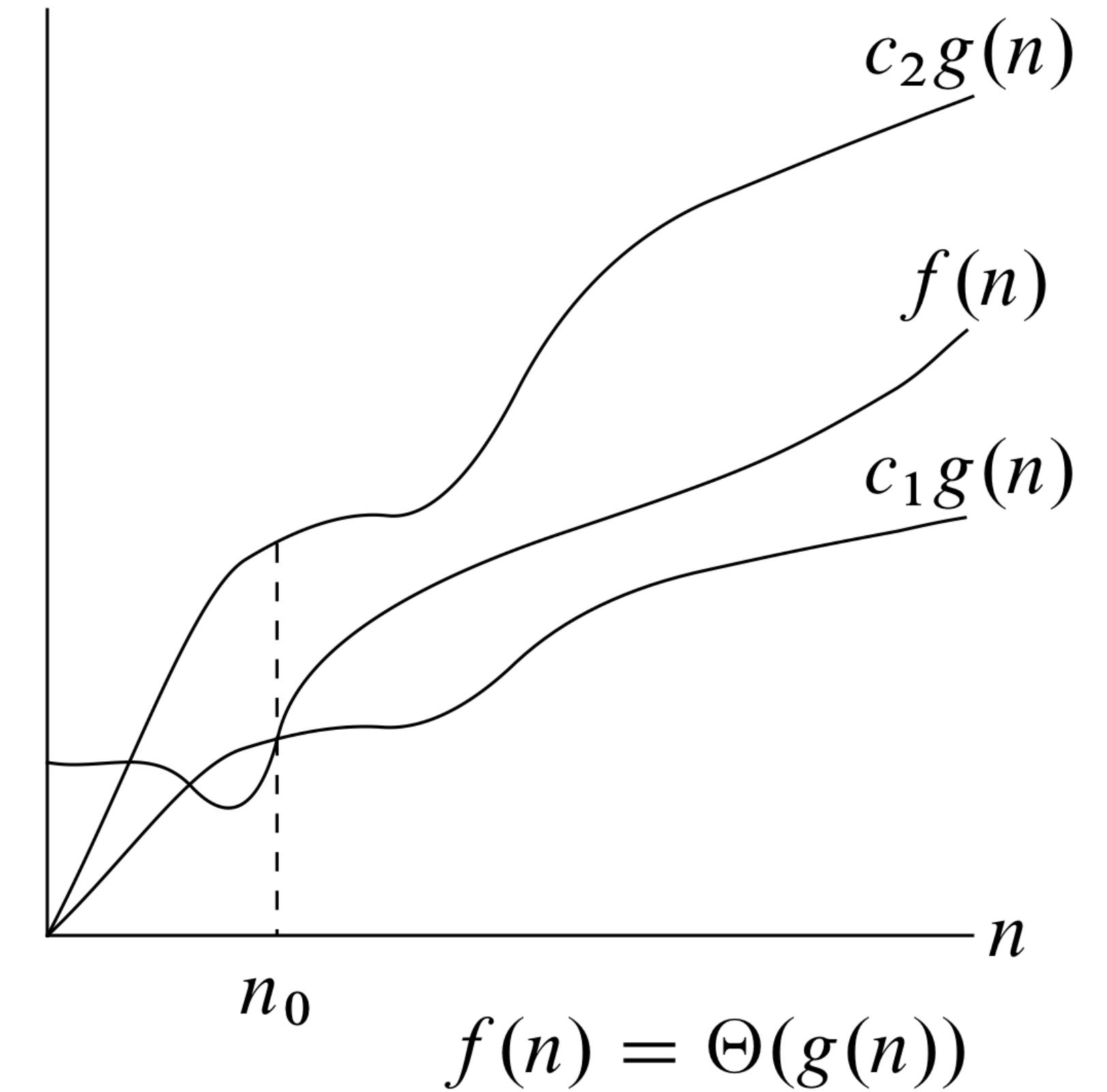
# Store-Omega

- Begrænser funktioner *nedefra*
- I øvelsen tidligere så vi, at insertion sort kører i tid  $\Omega(n^2)$  på et værste-falds input
- Det betyder *ikke* at algoritmen *altid* er så langsom – faktisk kører den i tid  $O(n)$  hvis input allerede er sorteret



# Store-Theta

- Begrænser funktioner *nedefra og oppefra*
- Vi har  $T(n) = \Theta(g(n))$  hvis og kun hvis  $T(n) = O(g(n))$  og  $T(n) = \Omega(g(n))$  (øvelse)
- Insertion sort har værstefalds køretid  $T(n) = \Theta(n^2)$
- **NB!** Det er også sandt insertion sort har værstefalds køretid  $T(n) = \Omega(n \log n)$ , men det er ikke en præcis analyse



# Eksempel

- Antag at  $a$  og  $b$  er positive heltal.
- Hvad kan vi sige om  $(n + a)^b$ ?
- $(n + a)^b = n^b + \binom{b}{1} an^{b-1} + \binom{b}{2} a^2 n^{b-2} + \dots + \binom{b}{b-1} a^{b-1} n + \binom{b}{0} a^b$
- Vælg  $c = 1 + \binom{b}{1} a + \binom{b}{2} a^2 + \dots + \binom{b}{b-1} a^{b-1} + \binom{b}{0} a^b$ , så er  
 $n^b \leq (n + a)^b \leq cn^b$  for  $n \geq 1$
- Dvs.  $(n + a)^b = \Theta(n^b)$

# Regler for asymptotisk notation

**Theorem 5.** Let  $f, g, h, p : \mathbb{R}^+ \rightarrow \mathbb{R}$  be asymptotically positive functions.

(R1) **“Overall constant factors can be ignored”**

If  $c > 0$  is a constant then  $cf(x)$  is  $\Theta(f(x))$ .

(R2) **“For polynomials only the highest-order term matters”**

If  $p(x)$  is a polynomial of degree  $d$ , then  $p(x)$  is  $\Theta(x^d)$ .

(R3) **“The fastest growing term determines the growth rate”**

If  $f(x)$  is  $o(g(x))$  then  $c_1g(x) + c_2f(x)$  is  $\Theta(g(x))$ , where  $c_1 > 0$  and  $c_2 \in \mathbb{R}$  are constants.

(R4) **“Logarithms grow faster than constants”**

If  $c > 0$  is a constant then  $c$  is  $o(\log_a(x))$  for all  $a > 1$ .

(R5) **“Powers (and polynomials) grow faster than logarithms”**

$\log_a(x)$  is  $o(x^b)$  for all  $a > 1$  and  $b > 0$ .

(R6) **“Exponentials grow faster than powers (and polynomials)”**

$x^a$  is  $o(b^x)$  for all  $a$  and all  $b > 1$ .

(R7) **“Larger powers grow faster”**

$x^a$  is  $o(x^b)$  if  $a < b$ .

(R8) **“Exponentials with a bigger base grow faster”**

$a^x$  is  $o(b^x)$  if  $0 < a < b$ .

**Theorem 7.** Let  $f, g, h : \mathbb{R}^+ \rightarrow \mathbb{R}$  be asymptotically positive functions.

(R9) **Transitivity**

If  $f(x)$  is  $O(g(x))$  and  $g(x)$  is  $O(h(x))$  then  $f(x)$  is  $O(h(x))$ .

If  $f(x)$  is  $o(g(x))$  and  $g(x)$  is  $o(h(x))$  then  $f(x)$  is  $o(h(x))$ .

If  $f(x)$  is  $\Theta(g(x))$  and  $g(x)$  is  $\Theta(h(x))$  then  $f(x)$  is  $\Theta(h(x))$ .

(R10) **Chaining**

If  $f(x)$  is  $O(g(x))$  and  $g(x)$  is  $\Theta(h(x))$  then  $f(x)$  is  $O(h(x))$ .

If  $f(x)$  is  $\Theta(g(x))$  and  $g(x)$  is  $O(h(x))$  then  $f(x)$  is  $O(h(x))$ .

If  $f(x)$  is  $o(g(x))$  and  $g(x)$  is  $O(h(x))$  then  $f(x)$  is  $o(h(x))$ .

If  $f(x)$  is  $O(g(x))$  and  $g(x)$  is  $o(h(x))$  then  $f(x)$  is  $o(h(x))$ .

If  $f(x)$  is  $o(g(x))$  and  $g(x)$  is  $\Theta(h(x))$  then  $f(x)$  is  $o(h(x))$ .

If  $f(x)$  is  $\Theta(g(x))$  and  $g(x)$  is  $o(h(x))$  then  $f(x)$  is  $o(h(x))$ .

# Lille-o og lille-omega

“ $>$ ”

“ $\geq$ ”

“ $=$ ”

“ $\leq$ ”

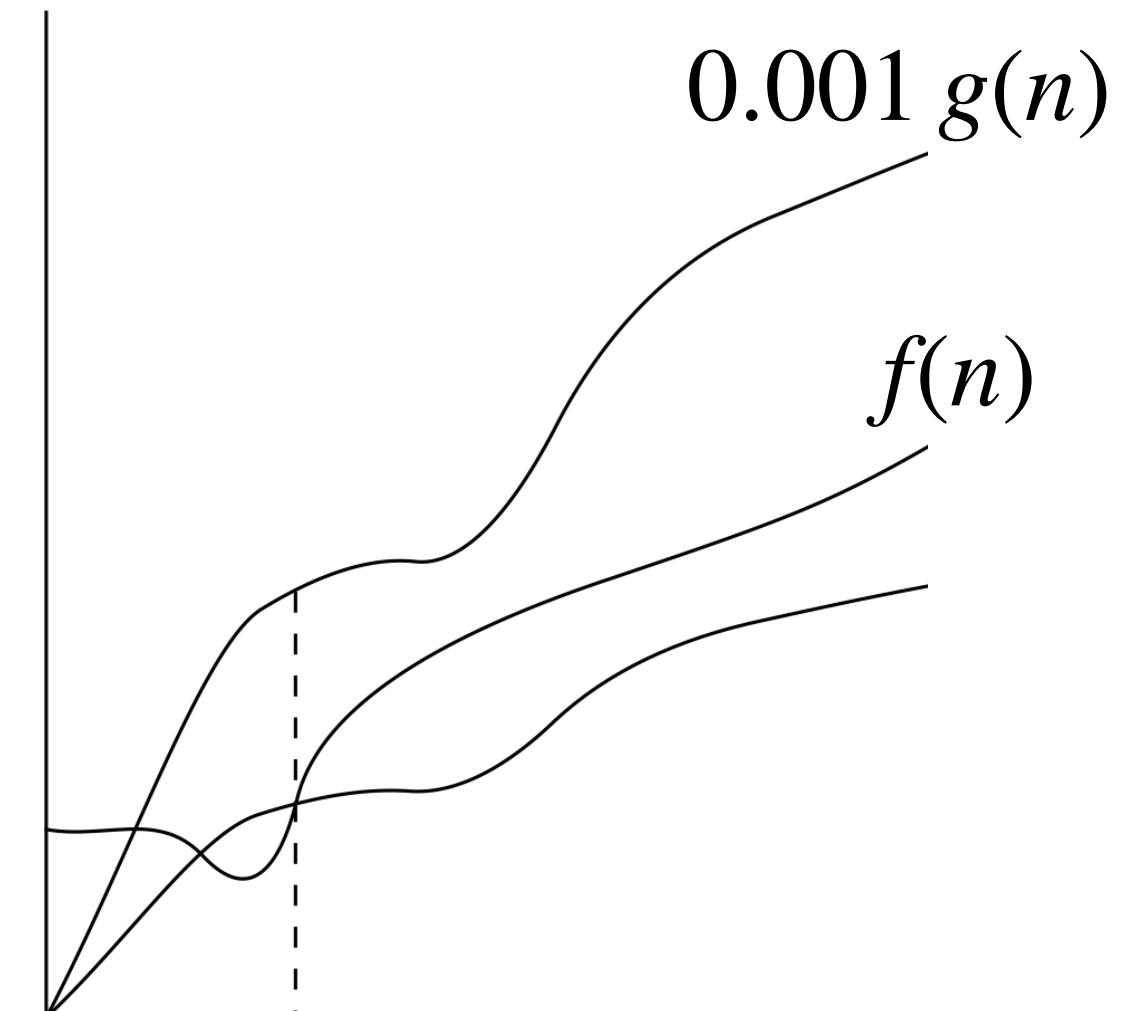
“ $<$ ”

$$f(n) = \omega(g(n)) \quad f(n) = \Omega(g(n)) \quad f(n) = \Theta(g(n)) \quad f(n) = O(g(n)) \quad f(n) = o(g(n))$$

- **Formelle definitioner:** For en ikke-negativ funktion  $g(n) : \mathbb{N} \rightarrow \mathbb{R}$  er

$$o(g(n)) = \{f(n) \mid \forall c > 0 : \exists n_0 > 0 : n \geq n_0 \Rightarrow 0 \leq f(n) < cg(n)\}$$

$$\omega(g(n)) = \{f(n) \mid \forall c > 0 : \exists n_0 > 0 : n \geq n_0 \Rightarrow cg(n) < f(n)\}$$



# Regler og ikke-regler for lille-o og lille-omega

“ $>$ ”

“ $\geq$ ”

“ $=$ ”

“ $\leq$ ”

“ $<$ ”

$$f(n) = \omega(g(n)) \quad f(n) = \Omega(g(n)) \quad f(n) = \Theta(g(n)) \quad f(n) = O(g(n)) \quad f(n) = o(g(n))$$

## Regler:

- $f(n) = \omega(g(n)) \Rightarrow f(n) = \Omega(g(n))$   
“ $a > b \Rightarrow a \geq b$ ”
- $f(n) = o(g(n)) \Rightarrow f(n) = O(g(n))$   
“ $a < b \Rightarrow a \leq b$ ”
- $f(n) = \omega(g(n)) \Rightarrow f(n) \notin \Theta(g(n))$   
“ $a > b \Rightarrow \neg(a = b)$ ”

## Ikke-regler:

- $f(n) = \omega(g(n)) \vee f(n) = O(g(n))$   
“ $a > b \vee a \leq b$ ”
- $f(n) = o(g(n)) \vee f(n) = \Omega(g(n))$   
“ $a < b \vee a \geq b$ ”



# Nogle grundlæggende funktioner brugt i kurset

Detaljer i CLRS 3.3

- Oprunding  $\lceil x \rceil$  og nedrunding  $\lfloor x \rfloor$
- Fakultet:  $n! = n \cdot (n - 1) \cdot \dots \cdot 2 \cdot 1$
- Polynomier af grad  $d$ :

$$f(n) = a_d n^d + a_{d-1} n^{d-1} + \dots + a_1 n + a_0 = \sum_{i=0}^d a_i n^i$$

- Logaritmer:
  - $\ln(n)$  er tallet der opfylder  $e^{\ln(n)} = n$ , hvor  $e = \sum_{i=0}^{\infty} (1/i!) = 2,718\dots$
  - $\lg(n) = \log_2(n)$  er tallet der opfylder  $2^{\lg(n)} = n$

# Nogle nytige regneregler

Detaljer i CLRS 3.3

For  $n, m > 0$  og  $a, b \in \mathbb{R}$ :

- $\lceil n/2 \rceil + \lfloor n/2 \rfloor = n$  for  $n \in \mathbb{N}$
- $n^a \cdot n^b = n^{a+b}$
- $1/n^a = n^{-a}$
- $\log(n \cdot m) = \log n + \log m$
- $\log(n^a) = a \log n$

# Store-O notation med flere variable

- Hvad betyder  $f(n, m) = O(g(n, m))$ , f.eks.  $T(n, m) = O(m + n \log n)$ ?
- Opgave 3.2-7 i CLRS: Vi siger at  $f(n, m) = O(g(n, m))$  hvis og kun hvis der findes  $c > 0, k > 0$  så  $(n \geq k \vee m \geq k) \Rightarrow 0 \leq f(n, m) \leq cg(n, m)$
- Med andre ord:
  - For ethvert fast  $m = m_0$  er  $f(n, m_0) = O(g(n, m_0))$ , og
  - For ethvert fast  $n = n_0$  er  $f(n_0, m) = O(g(n_0, m))$
- NB! Ikke den eneste mulige definition, men den mest brugte  
(se også diskussion af Rodney Howell)

## Quiz om asymptotisk notation



# Næste skridt

- Øvelser i store-O notation
  - Kom i gang med det samme ved at se på opgaverne til mandagens øvelser
- **Næste forelæsning, mandag:**
  - Del-og-hersk paradigmet
  - Rekursionsligninger
  - Analyse af *merge sort*
  - Nedre grænse for tidskompleksiteten af enhver sorteringsalgoritme (sammenligningsbaseret)