Rekursion og eksperimental analyse

Dagens læringsmål

- Efter dagens forelæsning vil du være i stand til at
  - Forklare hvad en rekursiv algoritme er
  - Forklare forskellen mellem rekursion og iteration
  - Konstruere rekursive algoritmer
  - Analysere køretiden og argumentere for korrektheden af en rekursiv algoritme
  - Anvende mergesort, analysere dens køretid, samt argumentere for dens korrekthed.
  - Bruge den videnskabelige metode til at lave eksperimentel analyse af algoritmer.
- Pensum:
  - Reges og Stepp udleverede kopier (om rekursion)
  - KT 5.1 + s. 48-50 (om merge, mergesort og analysen af dem).
  - Sedgewick og Wayne udleverede kopier.

Rekursion

Rekursion

- Iteration
  - Aktioner der skal gentages beskrives ved hjælp af en løkke.
- Rekursion
  - Aktioner der skal gentages beskrives ved hjælp af en metode/algoritme der kalder sig selv.

Rekursion: Skriv ord ud i omvendt rækkefølge

- Input:

  er

  dette

  ikke

  sjovt

- Output:

  sjovt

  ikke

  dette

  er

```
public static void reverse(Scanner input){
  if (input.hasNextLine()) {
  String line = input.nextLine();
  reverse(input);
  System.out.println(line);
  }
}
```

Input:

er

dette

ikke

sjovt

line: er

environment

```
public static void reverse(Scanner input){
    if (input.hasNextLine()) {
    String line = input.nextLine();
    reverse(input);
    System.out.println(line);
    }
}
```

Input:

er

dette

ikke

sjovt

line: er

environment

```
public static void reverse(Scanner input){
    if (input.hasNextLine()) {
    String line = input.nextLine();
    reverse(input);
```

line: dette

environment

```
public static void reverse(Scanner input){
    if (input.hasNextLine()) {
    String line = input.nextLine();
    reverse(input);
```

line: ikke

environment

```
public static void reverse(Scanner input){
    if (input.hasNextLine()) {
    String line = input.nextLine();
    reverse(input);
```

line: sjovt

environment

```
public static void reverse(Scanner input){
    if (input.hasNextLine()) {
    String line = input.nextLine();
    reverse(input);
```

```
public static void reverse(Scanner input){
    if (input.hasNextLine()) {
    String line = input.nextLine();
    reverse(input);
    System.out.println(line);
    }
}
```

Input:

er

dette

ikke

sjovt

line: er

environment

```
public static void reverse(Scanner input){
    if (input.hasNextLine()) {
    String line = input.nextLine();
    reverse(input);
    System.out.println(line);
    }
}
```

Output:

sjovt

ikke

dette

er

Rekursion

- Basis-tilfælde. Simpelt/simple tilfælde der løses uden rekursivt kald

- Rekursions-tilfælde/skridt. Reducerer til simplere problem der kan løses ved et (eller flere) rekursive kald.

Eksempel

```
public static void reverse(Scanner input){
 if (input.hasNextLine()) {
 String line = input.nextLine();
 reverse(input);
 System.out.println(line);
 }
}
```

- Basis-tilfælde. Gør ingenting.
- Rekursions-tilfælde. reverse(input)

Rekursion og induktion

- Rekursion
  - Basis-tilfælde. Simpelt/simple tilfælde der løses uden rekursivt kald
  - Rekursions-tilfælde/skridt. Reducerer til simplere problem der kan løses ved et (eller flere) rekursive kald.

- Induktion (over *n*)
  - Basis-tilfælde. Simpelt/simple tilfælde der bevises for specifikke værdier af *n*.
  - Induktionsskridt. Antag at udsagnet gælder for alle heltal mindre end *n*, og brug dettte til at vise det er sandt for *n*.

Eksempel

```
public static void reverse(Scanner input){
 if (input.hasNextLine()) {
 String line = input.nextLine();
 reverse(input);
 System.out.println(line);
 }
}
```

- Korrekthed
    - Tom linie: Gør ingenting. ✓
    - *n* linier: Antag `reverse(input)` udskriver de sidste *n-1* linier i omvendt rækkefølge. Så udskrives de *n-1* sidste linier i omvendt rækkefølge først og bagefter udskrives den første linie. ✓

Eksempel

```
public static void reverse(Scanner input){
 if (input.hasNextLine()) {
 String line = input.nextLine();
 reverse(input);
 System.out.println(line);
 }
}
```

- Køretid T(n)

Hvert rekursivt kald tager worst-case O(1) tid $\Rightarrow$ T(n) = O(n).

Største fælles divisor (gcd)

- gcd(p,q). Find største heltal der går op i begge tal.

- Eksempel. gcd(4032,1272) = 24.

$$4032 = 2^6 \times 3^2 \times 7^1$$

$$1272 = 2^3 \times 3^1 \times 53^1$$

$$\text{gcd} = 2^3 \times 3^1 = 24$$

- Anvendelser
    - Simple brøker = 1272/4032 = 53/168
    - RSA kryptografisystem

Største fælles divisor

- gcd(p,q). Find største heltal der går op i både p og q.

- Euclids algoritme.

$$\gcd(p,q) = \begin{cases} p & \text{if } q = 0 \\ gcd(q, p \,\%\, q) & \text{otherwise} \end{cases}$$

← **Basistilfælde**

← **Rekursionsskridt**
Konvergerer mod basistilfældet.

$$\begin{aligned} \text{gcd}(4032, 1272) &= \text{gcd}(1272, 216) \\ &= \text{gcd}(216, 192) \\ &= \text{gcd}(192, 24) \\ &= \text{gcd}(24, 0) \\ &= 24. \end{aligned}$$

← `4032 = 3 × 1272 + 216`

Største fælles divisor

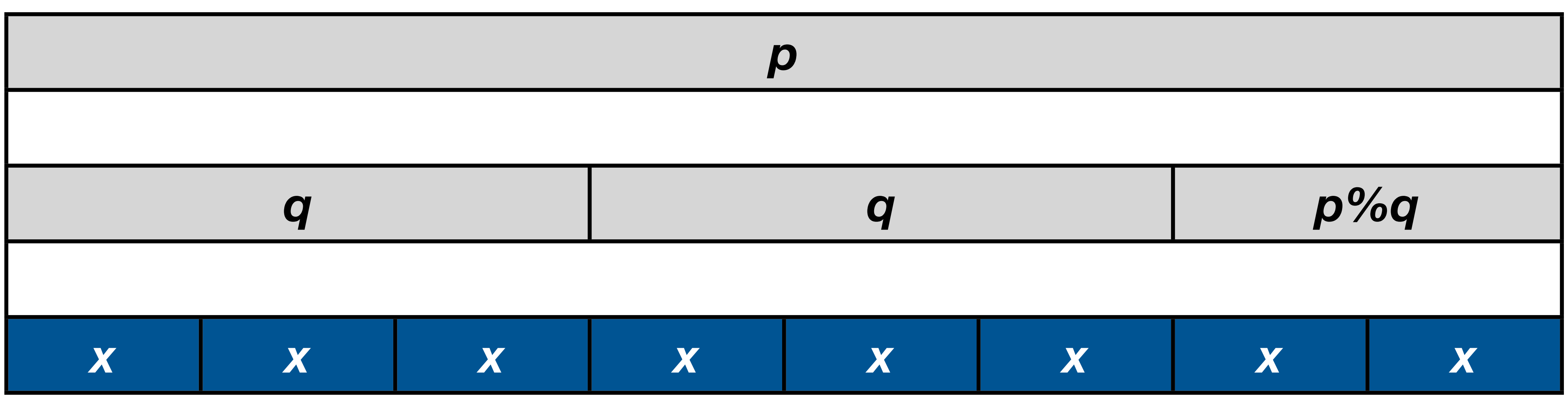
- gcd(p,q). Find største heltal der går op i både p og q.

- Euclids algoritme.

$$\gcd(p,q) = \begin{cases} p & \text{if } q = 0 \\ gcd(q, p \,\%\, q) & \text{otherwise} \end{cases}$$

← **Basistilfælde**

← **Rekursionsskridt**
Konvergerer mod basistilfældet.



gcd(p,q) = x, p = 8x, q = 3x

Største fælles divisor

- gcd(p,q). Find største heltal der går op i både p og q.

$$\gcd(p,q) = \begin{cases} p & \text{if } q = 0 \\ gcd(q, p \;\%\; q) & \text{otherwise} \end{cases}$$

← **Basistilfælde**

← **Rekursionsskridt**

- Pseudokode

```
gcd(p,q) {
  if (q = 0) return p;                 ← Basistilfælde
  else return gcd(q,p % q);            ← Rekursionsskridt
}
```

Towers of Hanoi

- Ryk alle plader fra den venstre pind til den højre pind.
  - Kun tilladt at rykke én plade ad gangen.
  - En plade kan enten placeres på en tom pind eller ovenpå en *større* plade.

start

slut

- Legende: Verdens ende når en gruppe af munke er færdige med at flytte pladerne i et tempel med 64 guldplader på 3 diamantnåle.
- http://mazeworks.com/hanoi/index.htm

Towers of Hanoi: Rekursiv løsning

Move n-1 smallest discs right.

Move largest disc left.

Move n-1 smallest discs right.

Towers of Hanoi: Rekursiv løsning

- TowersOfHanoi(*n*, *left*) udskriver de flytninger der skal til for at rykke *n* plader til venstre (hvis *left* = *true*) eller til højre (hvis *left* = *false*).

```
TowersOfHanoi(n, left){
   if (n=0) return;  // Basistilfælde
   TowersOfHanoi(n-1, !left)
   if (left) print(n + " left")
   else print(n + " right")
   TowersOfHanoi(n-1, !left)
}
```

Ryk n-1 øverste til modsat side

Ryk største til ønskede sted.

Ryk n-1 ovenpå den største

Towers of Hanoi: Rekursiv løsning

- TowersOfHanoi(*n*, *left*) udskriver de flytninger der skal til for at rykke *n* plader til venstre (hvis *left* = *true*) eller til højre (hvis *left* = *false*).

```
TowersOfHanoi(n, left){
   if (n=0) return;  // Basistilfælde
   TowersOfHanoi(n-1, !left)
   if (left) print(n + " left")
   else print(n + " right")
   TowersOfHanoi(n-1, !left)
}
```

Ryk n-1 øverste til modsat side

Ryk største til ønskede sted.

Ryk n-1 ovenpå den største

- Antal flytninger. $T(n) = 2 \cdot T(n-1) + 1$

Towers of Hanoi: Rekursiv løsning

- Antal flytninger: $T(n) = 2 \cdot T(n-1) + 1$
    - $T(1) = 1$, $T(2) = 3$, $T(3) = 7$, $T(4) = 15$
    - $T(n) = 2^n - 1$

- Bevis:
    - Hjælpefunktion $U(n)$: $U(n) = T(n) + 1$
    - $U(0) = T(0) + 1 = 1$
    - $U(n) = T(n) + 1 = (2 \cdot T(n-1) + 1) + 1 = 2 \cdot T(n-1) + 2 = 2 \cdot U(n-1)$
    - $U(n) = 2^n$
    - $T(n) = U(n) - 1 = 2^n - 1$

Towers of Hanoi: Rekursiv løsning

- Antal flytninger: $T(n) = 2 \cdot T(n-1) + 1 = 2^n - 1$

- Induktionsbevis:
  - Basis tilfælde: $T(0) = 2^0 - 1 = 0$.
  - Induktionsskridt:
    - Antag $T(n-1) = 2^{n-1} - 1$.
    - $T(n) = 2 \cdot (2^{n-1} - 1) + 1 = 2^n - 2 + 1 = 2^n - 1$.

Towers of Hanoi: Rekursiv løsning

- Antal flytninger: $T(n) = 2^n - 1$
  - For n = 64 tager det 585 milliarder år (hvis der rykkes en plade i sekundet).
  - Enhver løsning bruger mindst så mange skridt.
  - Køretiden af TowersOfHanoi: $T(n) = 2 \cdot T(n-1) + O(1) = \Theta(2^n)$
  - Pas på programmer der tager eksponentiel tid!!!

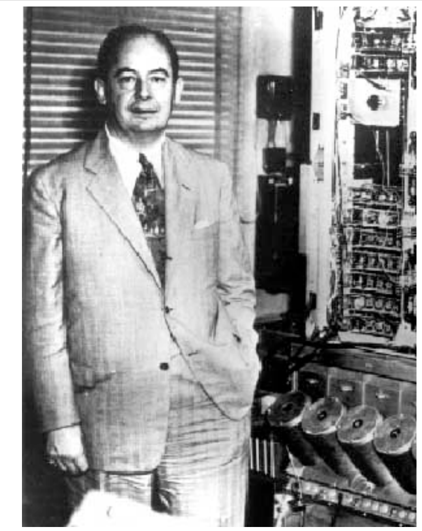
Mergesort

Sortering

- Sortering. Givet n elementer, omarranger dem i ikke-faldende orden.
- Oplagte anvendelser
  - Sorter en liste af navne, organiser et MP3 bibliotek, vis Google PageRank resultater, skriv RSS nyheder op i omvendt kronologisk orden.
- Nemme problemer for sorterede elementer
  - Find medianen, find nærmeste par, binær søgning i database, identificer statistiske “outliers”, find duplikater i mailingliste.
- Ikke oplagte anvendelser
  - Datakompression, computergrafik, computational biology, anbefaling af bøger på Amazon.

Mergesort

- Mergesort
  - Del tabellen i 2 halvdele.
  - Sorter rekursivt hver del
  - Flet de to halvdele sammen til én sorteret liste.



Jon von Neumann (1945)

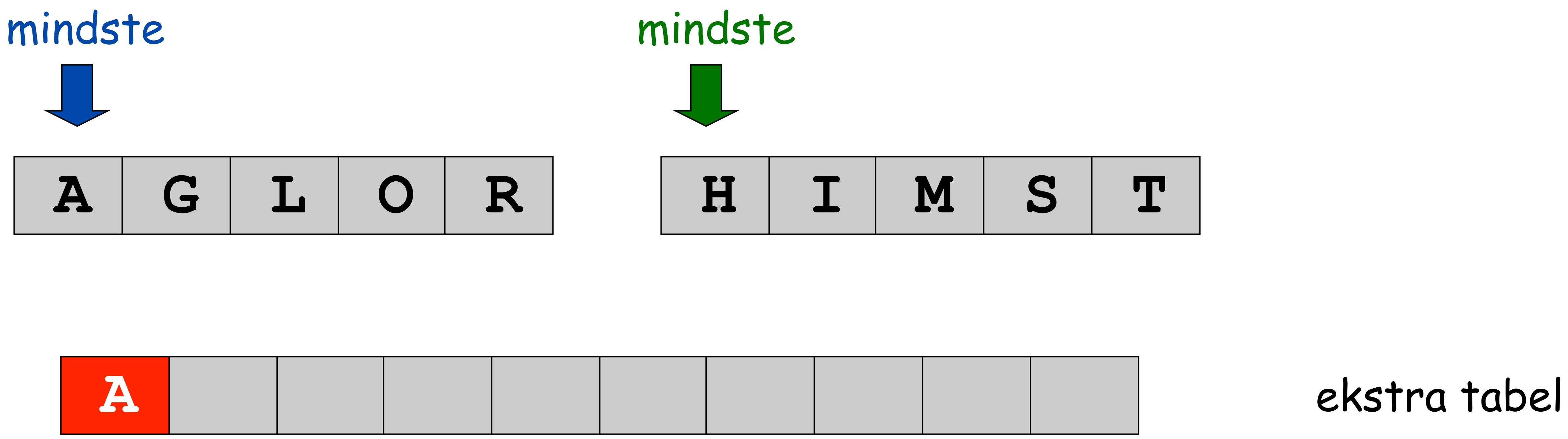
A	L	G	O	R	I	T	H	M	S		
A	L	G	O	R		I	T	H	M	S	Del
A	G	L	O	R		H	I	M	S	T	Sorter rekursivt
A	G	H	I	L	M	O	R	S	T		Flet

Fletning

- Mål. Kombiner to sorterede lister til én sorteret liste i lineær tid
- Idé.
    - Hold styr på mindste element i hver sorteret halvdel.
    - Indsæt mindste af de to elementer i en ekstra tabel.
    - Gentag indtil færdig.

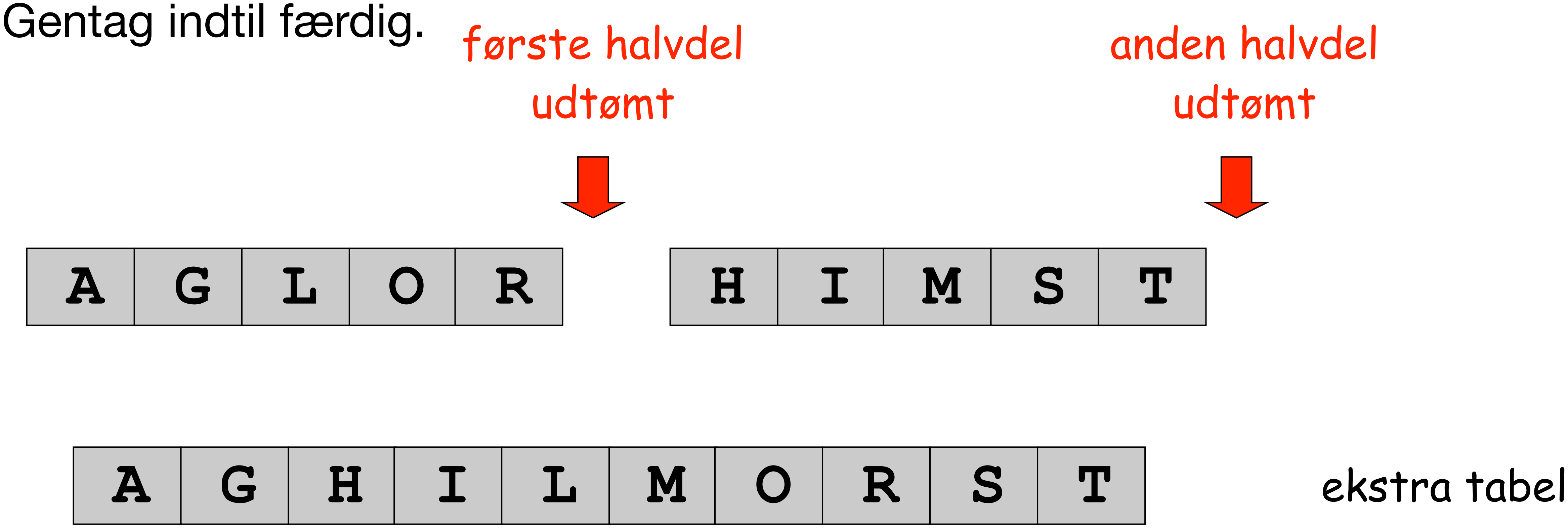
Fletning

- Flet.
    - Hold styr på mindste element i hver sorteret halvdel.
    - Indsæt mindste af de to elementer i en ekstra tabel.
    - Gentag indtil færdig.



Fletning

- Flet.
  - Hold styr på mindste element i hver sorteret halvdel.
  - Indsæt mindste af de to elementer i en ekstra tabel.
  - Gentag indtil færdig.

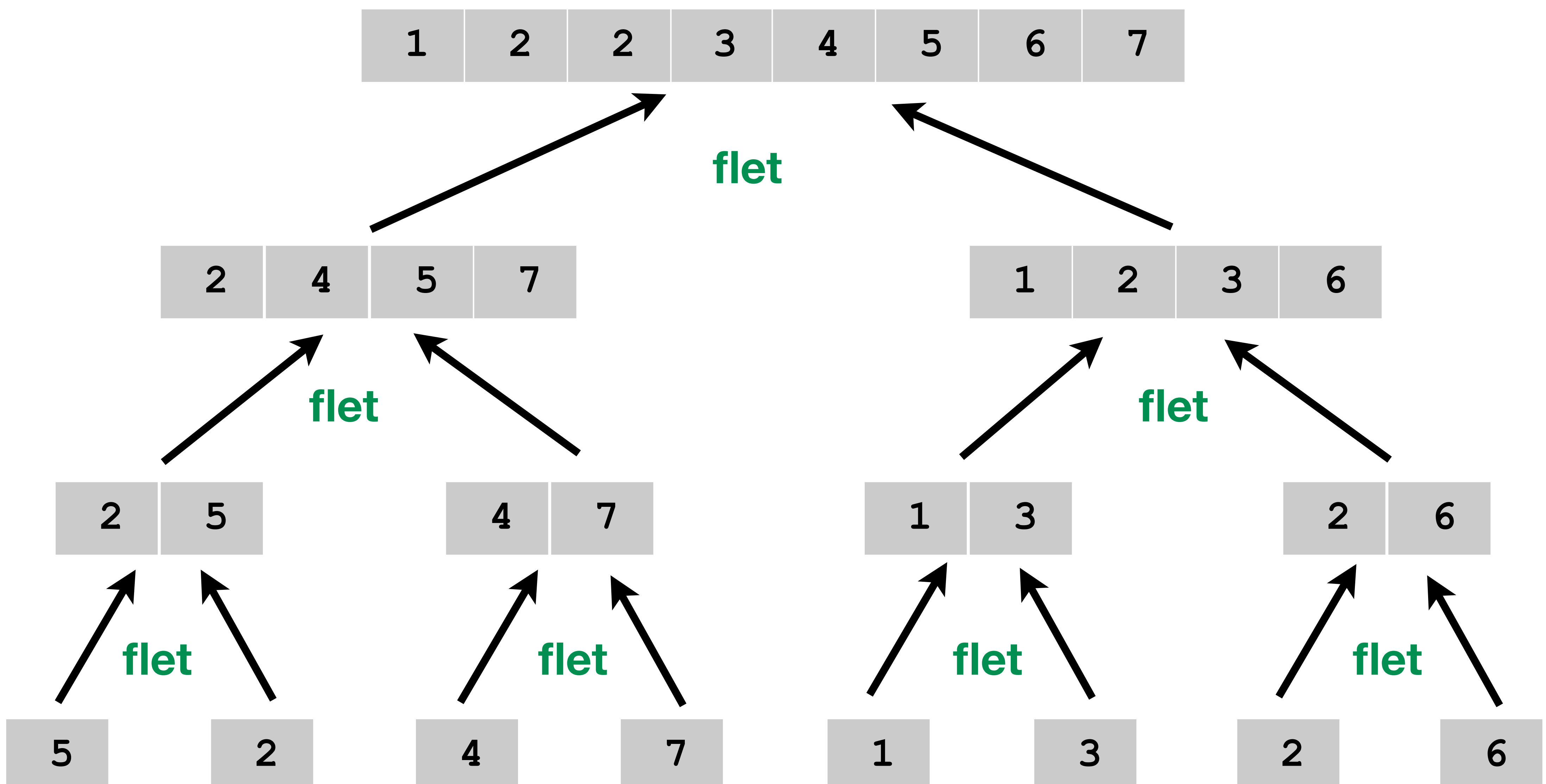


Mergesort

```
Mergesort(A, low, hi){
   if (low < hi)
     then {
       mid = ⌊(hi + low)/2⌋
       L = Mergesort(A, low, mid)
       R = Mergesort(A, mid+1, hi)
       return Merge(L,R)
     }
}
```

Mergesort: Eksempel

- A = < 5, 2, 4, 7, 1, 3, 2, 6 >



Mergesort: Korrekthed

- Påstand. Mergesort sorterer korrekt enhver tabel.

- Bevis: (induktion over n)
  - Basistilfælde. n=1. Allerede sorteret.
  - Induktionsskridt. Antag at påstanden er sand for alle tabeller af størrelse mindre end n.
    - Venstre og højre halvdel har begge størrelse mindre end n.
    - ⇒ det rekursive kald sorterer dem korrekt
    - Merge fletter de to sorterede tabeller korrekt.

Mergesort: Køretid

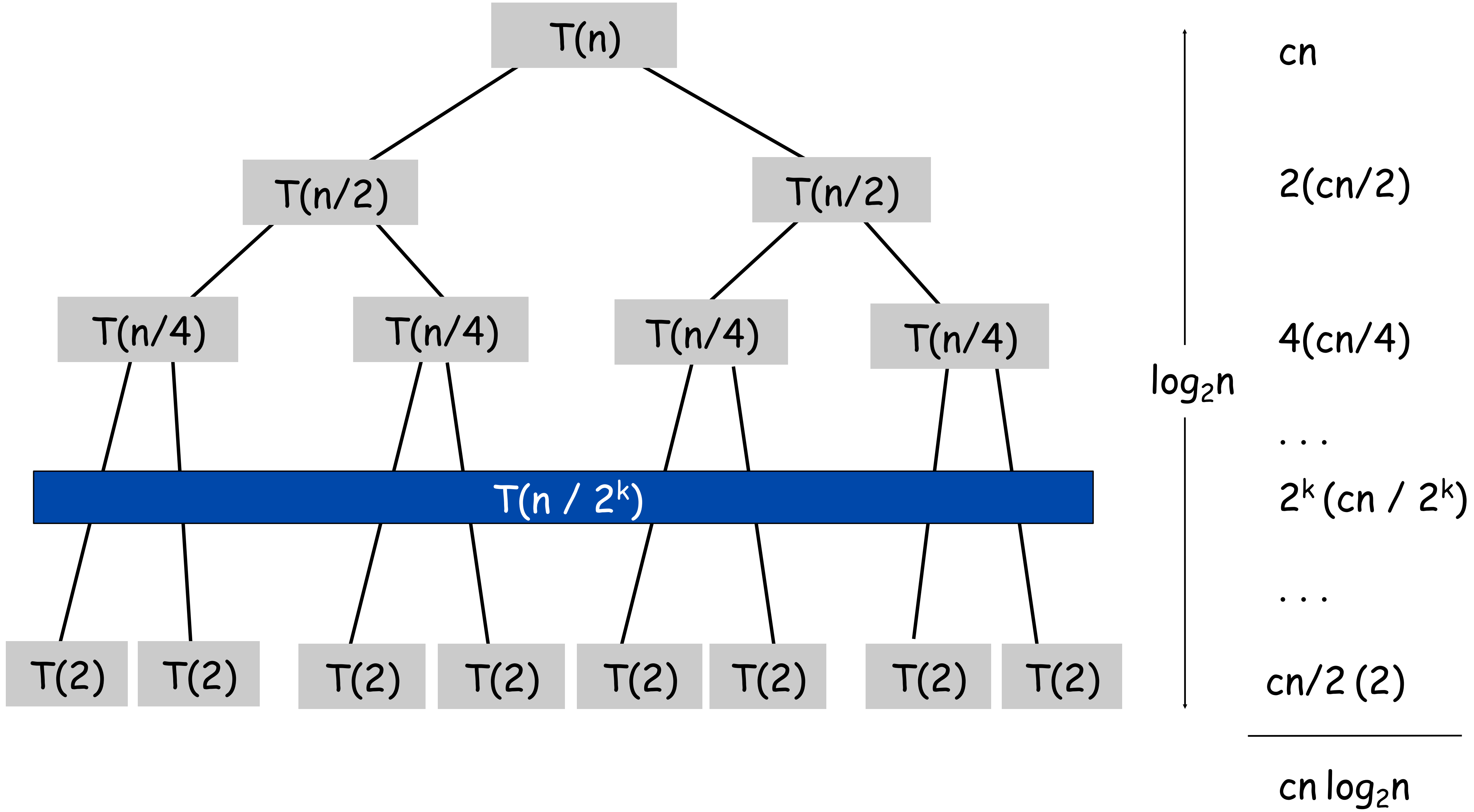
- Analyse
- Hvis $n = 0, 1$ eller 2: bruger vi konstant tid
- Hvis $n > 2$ bruger vi tid på
    - indeling af tabellen i 2 halvdele
    - rekursivt kald på venstre og højre halvdel
    - fletning af de sorterede tabeller
- Rekursionsligning
    - $T(n) \leq 2T(n/2) + cn,$ for $n > 2$ og en konstant c.
    - $T(2) \leq c$.

Mergesort: rekursionsligningen

- Rekursionsligning
  - $T(n) \le 2T(n/2) + cn$, for $n > 2$ og en konstant c.
  - $T(2) \le c$.
- Løsning
  - $T(n) = O(n \log n)$
- Forskellige beviser
  - Rekursionstræ
  - Substitution/induktion

Bevis: Rekursionstræ

- $T(n) \leq 2T(n/2) + cn$, for $n > 2$ og en konstant c, og $T(2) \leq c$.
- **Påstand.** $T(n) = O(n \log n)$



Bevis: Substitution

- Substitutionsmetoden.

  1. Gæt på løsningen.

  2. Brug matematisk induktion til at finde konstanterne og vise at løsningen virker.

Bevis: Substitution

- $T(n) \leq 2T(n/2) + cn$, for $n > 2$ og en konstant c, og $T(2) \leq c$.
- Påstand. $T(n) = O(n \log n)$
- Bevis: (induktion over n).
  - Basistilfælde: $n = 2$, $cn \log n = 2c \geq T(2)$.
  - Induktionshypotese: $T(m) \leq cm \log m$ for alle $m < n$.

$$
\begin{aligned}
T(n) &\leq 2T(n/2) + cn \\
&\leq 2c(n/2)\log(n/2) + cn \\
&= cn(\log n - 1) + cn \\
&= cn \log n - cn + cn = cn \log n.
\end{aligned}
$$

Rekursion: faldgruber

- Manglende basistilfælde
- Ingen konvergens mod basistilfældet
- Overdrevent pladsforbrug
- Overdreven genberegning

Rekursion: Q&A

- Er der situationer hvor iteration er den eneste mulige måde at angribe problemet på?
  - Nej. Enhver løkke kan erstattes af en rekursiv funktion.
- Er der situationer hvor rekursion er den eneste mulige måde at angribe problemet på?
  - Nej. Enhver rekursion kan erstattes af en tilsvarende iterativ metode.
- Hvad bør jeg foretrække?
  - Det der giver den simpleste, nemmest forståelige og mest effektive algoritme (afhængigt af det konkrete problem).

Opsummering

- Rekursive algoritmer
  - Basistilfælde, rekursionsskridt
  - Tæt knyttet til matematisk induktion.
  - Køretid: Rekursionstræ, substitutionsmetoden.
- Del-og-hersk. Elegant løsning til mange vigtige problemer.
  - Mergesort: O(n log n)

Eksperimentiel Analyse af Programmer

Philip Bille

Plan

- Teoretisk vs. eksperimentiel analyse
- Den videnskabelige metode
  - Bubblesort som eksempel

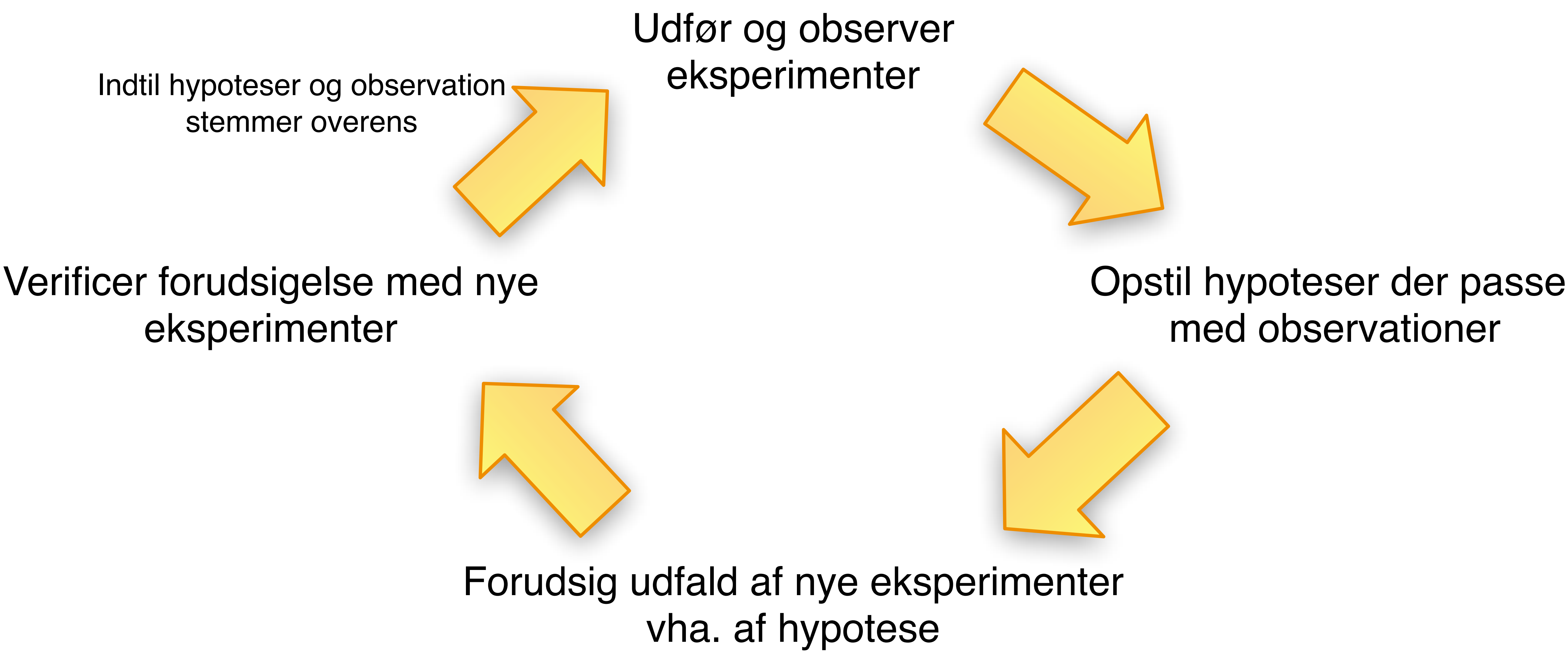
Teoretisk vs. Eksperimentiel Analyse

- Teoretisk analyse:
  - Analysere algoritmer for at estimere antal operationer som funktion af input størrelse.
  - Forsimplet model af virkelighed
  - Kan kræve avanceret matematik

- Eksperimentiel analyse:
  - Udføre eksperimenter for at måle køretid
  - Foregår i virkeligheden :-)
  - Nemt at gøre. Brug *videnskabelig metode*.

Den Videnskabelige Metode

Den Videnskabelige Metode



Eksempel: BubbleSort

BubbleSort

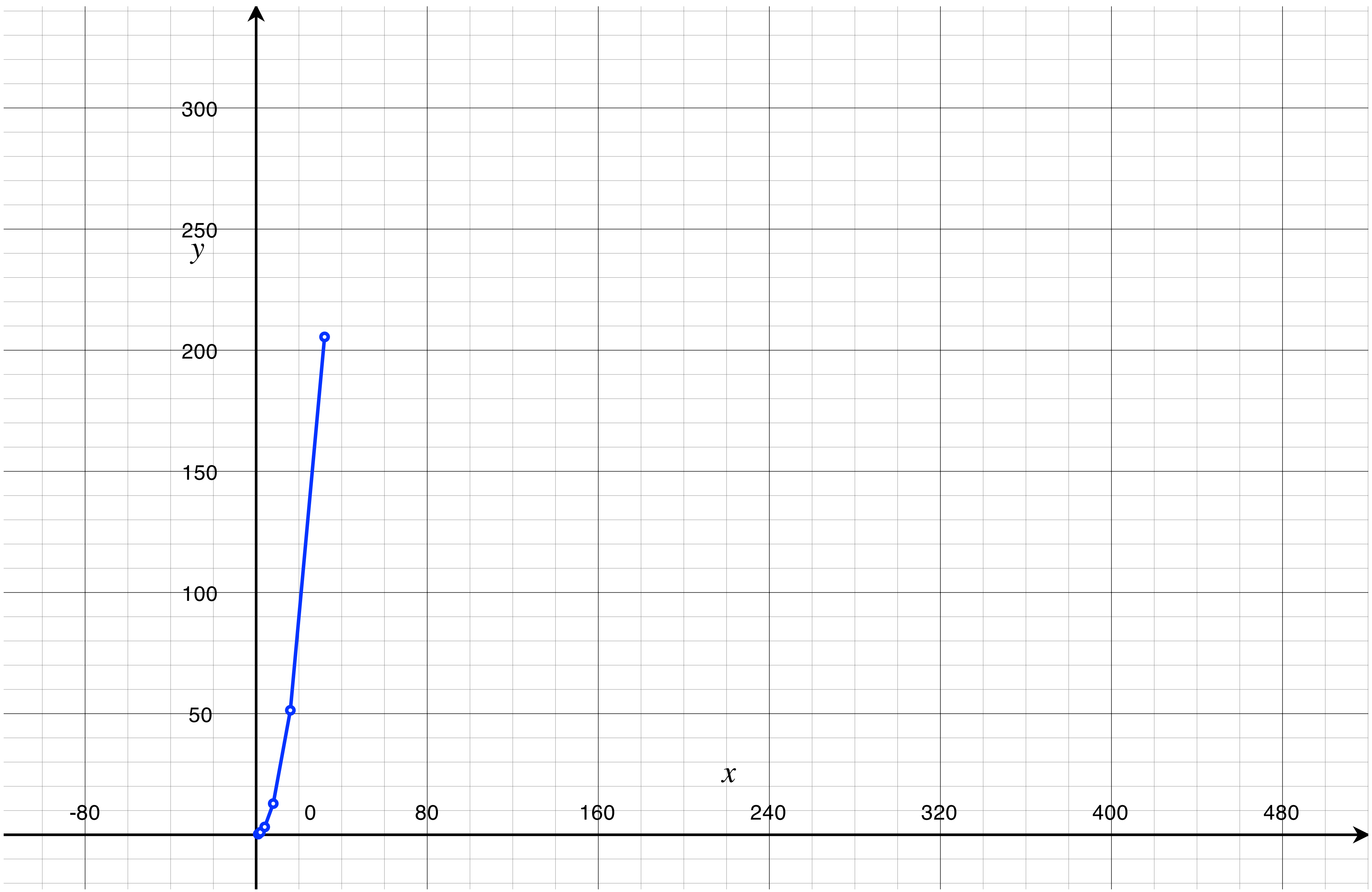
```
public static void bubbleSort(int[] A) {
    int tmp;
    for(int i = 1; i < A.length; i++) {
        for (int j = A.length - 1; j >= i; j--) {
            if(A[j] < A[j-1]) {
                tmp = A[j]; A[j] = A[j-1]; A[j-1] = tmp;
            }
        }
    }
}
```

BubbleSort

- Hvad ved vi om effektivitet af `bubbleSort`?
- Teori: Køretid for `bubbleSort` er cn^2 ($\Theta(n^2)$) for konstant $c > 0$
- Praksis: Hvordan passer den teoretiske køretid med praksis?
    - Er køretiden faktisk $\Theta(n^2)$ for standard input (f. eks. tilfældige tal)?
    - Hvad er konstanten c?
    - Er der andre faktorer der har væsentlig betydning for praktisk køretid?

Observer

n	tid
10000	0.27
20000	1.40
40000	3.23
80000	12.87
160000	51.38
320000	205.49



- Kunne ligne cn^2

Fordoblingshypotese

- Lad T(n) være tid for `bubbleSort` på tabel af størrelse n.
- Kig på forholdet mellem T(2n) og T(n).
- Hvis $T(n) = cn^2$ har vi:

$$\frac{T(2n)}{T(n)} = \frac{c(2n)^2}{cn^2} = \frac{c4n^2}{cn^2} = 4$$

- Hypotese 1 (*fordoblingshypothese*): $T(2n)/T(n) \approx 4$

Fordoblingshypotese

n	tid	T(2n)/T(n)
10000	0.27	
20000	1.40	5.18
40000	3.23	2.31
80000	12.87	3.98
160000	51.38	3.99
320000	205.49	4.00

Ser ud til at konvergere mod 4

Forudsigelse og Verifikation

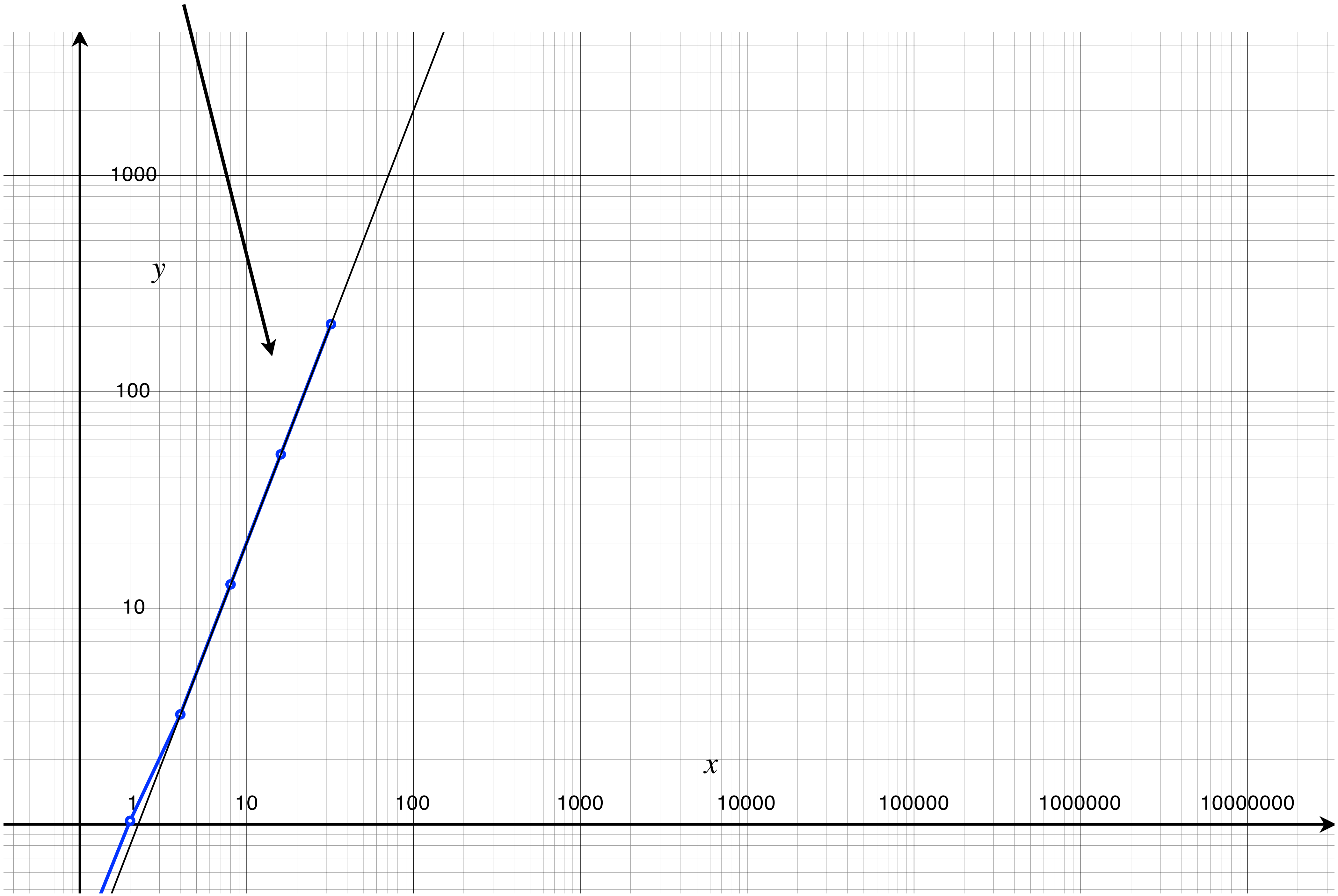
n	tid	T(2n)/T(n)
10000	0.27	
20000	1.40	5.18
40000	3.23	2.31
80000	12.87	3.98
160000	51.38	3.99
320000	205.49	4.00

- Forudsigelse: T(640000) = 205.49 · 4 = 821.96
- Observation: T(640000) = 822.01 !
- Yderligere observationer verificerer fordoblingshypotese.
- Overensstemmelse med hypotese og observationer :-)

Hvad Med c?

- Hvad nu hvis vi gerne vil finde konstanten c i køretid for cn^2 for `bubbleSort`?
- Metode:
  - Plot eksperimentielle målinger og tilpas c for kurve $f(n) = cn^2$.
  - Nemmest på dobbeltlogaritmisk skala da cn^d bliver ret linie.

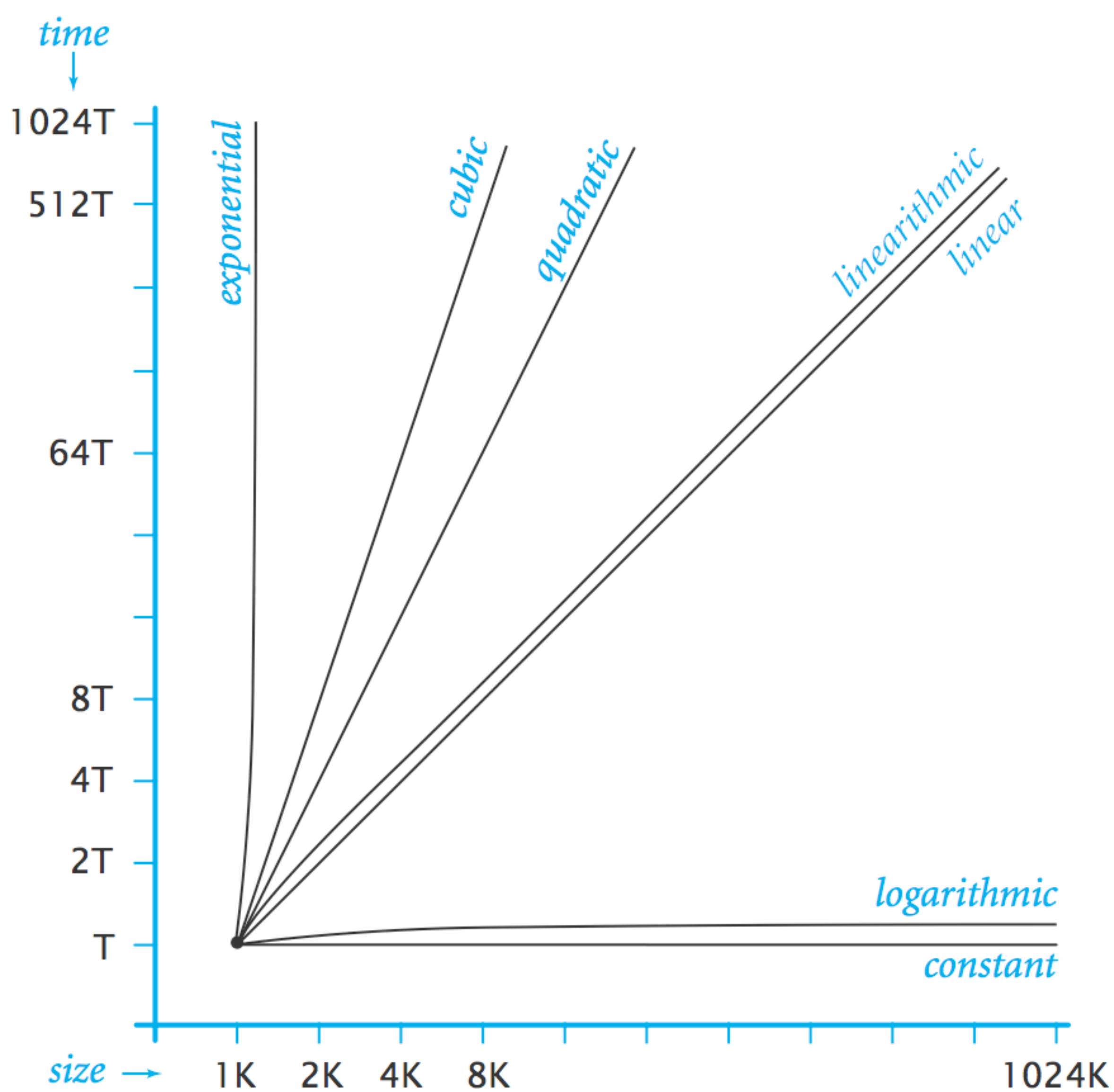
Målinger passer med $1/5 \cdot n^2$



Hypotese

- Hypotese 2: Køretid for `bubbleSort` er $1/5 \cdot n^2$.
- Kan ligeledes bruges til forudsigelse af køretider.

Klassifikation af Køretider



Orders of growth (log-log plot)

order of growth		factor for doubling hypothesis
description	*function*	
constant	1	1
logarithmic	$\log N$	1
linear	N	2
linearithmic	$N \log N$	2
quadratic	N^2	4
cubic	N^3	8
exponential	2^N	2^N

Teoretisk vs. Eksperimentiel Analyse

- Eksperimentiel analyse:
  - Mål køretider, opstil hypoteser
  - Nemt at udføre eksperimenter
  - Brugbar til at *forudsige* opførsel af programmer men ikke *forklare*
- Teoretisk Analyse:
  - Analysere algoritmer for at estimere antal operationer som funktion af input størrelse.
  - Kan kræve avanceret matematik
  - Brugbar til at forudsige og forklare

Resume

- Teoretisk vs. eksperimentiel analyse
- Den videnskabelige metode
  - Bubblesort som eksempel
  - Fordoblingshypoteser
  - Bestemmelse af konstanter vha. dobbeltlogaritmisk plot