

<https://xkcd.com/688/>

# Del-og-hersk, rekursionsligninger, og nedre grænse for sortering

Algoritmer og Datastrukturer  
Københavns Universitet, forår 2024

Rasmus Pagh



# Plan for i dag

- **Del 1:** Del-og-hersk paradigmet via eksempler:
  - Mergesort
  - Quicksort
  - Maximum del tabel
  - Heltalsmultiplikation (skolemetoden og Karatsuba)
- **Del 2:** Rekursionsligninger og løsningsmetoder:
  - Rekursionstræer
  - “Master theorem”
  - Substitutionsmetoden
- **Del 3:** Nedre grænse for antal sammenligninger i en sorteringsalgoritme

# Del og hersk-metoden

---

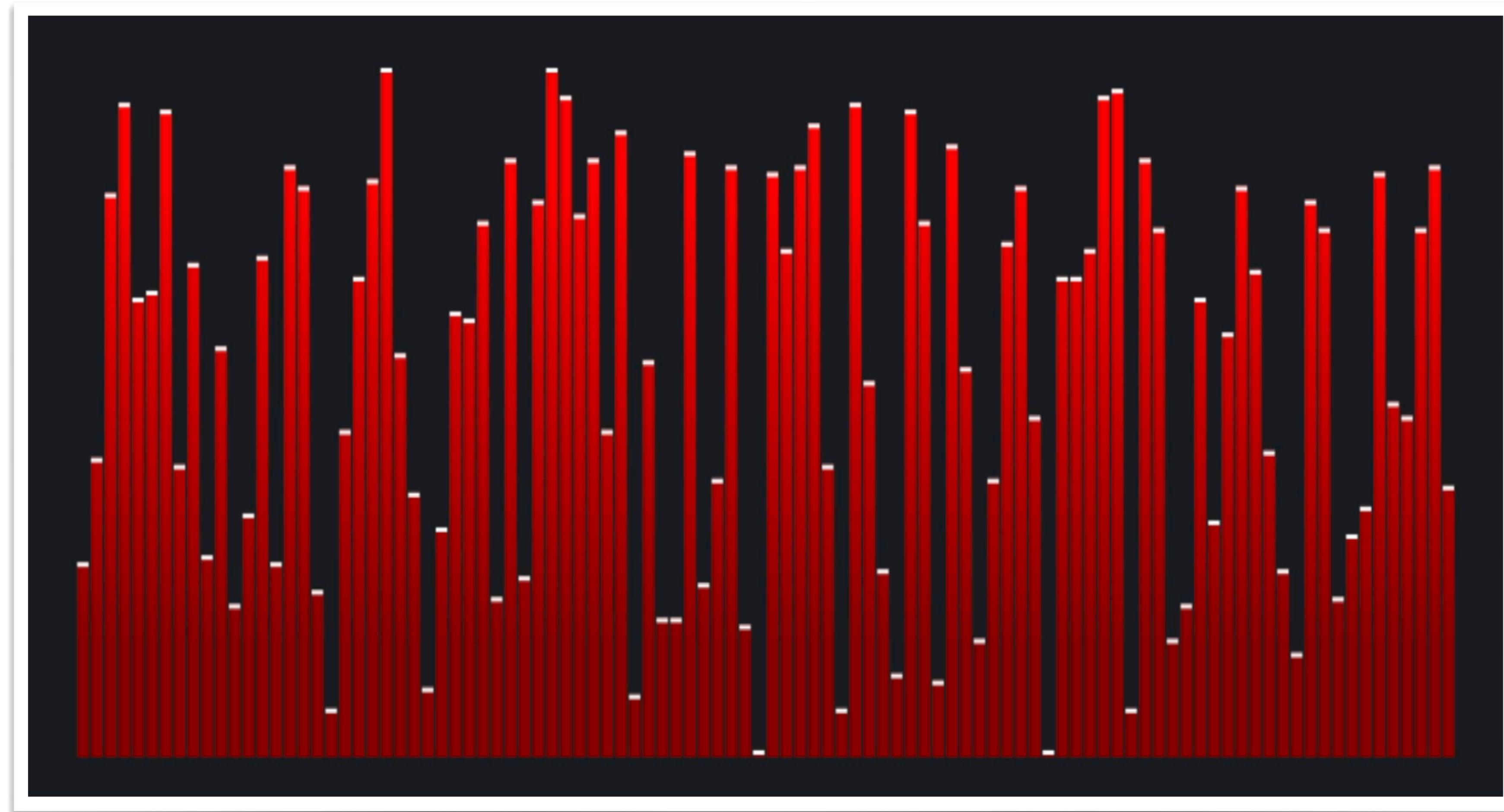
Fra Wikipedia, den frie encyklopædi

**Del og hersk** (divide et impera) er et udtryk kendt fra den [romerske](#) administration. Romerne brugte principippet til at kontrollere deres erobrede [provinser](#) ved at spille på interne konflikter, således at de slap for at bruge unødvendige ressourcer på at holde revolutionære personer nede. De tillod lokale skikke på den betingelse, at de lokale underlagde sig romersk kontrol.

# Del-og-hersk paradigmet i algoritmik

- En del-og-hersk algoritme er en rekursiv metode der virker ved at:
  - **Dele** problemet op i delproblemer af samme type som det oprindelige
  - Løse hvert delproblem rekursivt
  - **Kombinere** løsningerne til en løsning af det oprindelige problem
- Kaldes på dansk også “del-og-kombinér” (eng. “divide-and-conquer”)

# Del-og-hersk eksempel: Merge sort



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

# Pseudocode for merge

MERGE( $A, p, q, r$ )

$$n_1 = q - p + 1$$

$$n_2 = r - q$$

let  $L[1..n_1 + 1]$  and  $R[1..n_2 + 1]$  be new arrays

**for**  $i = 1$  **to**  $n_1$

$L[i] = A[p + i - 1]$

**for**  $j = 1$  **to**  $n_2$

$R[j] = A[q + j]$

$L[n_1 + 1] = \infty$

$R[n_2 + 1] = \infty$

$i = 1$

$j = 1$

**for**  $k = p$  **to**  $r$

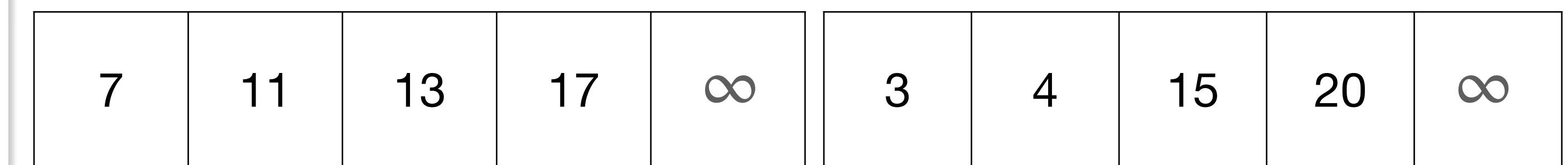
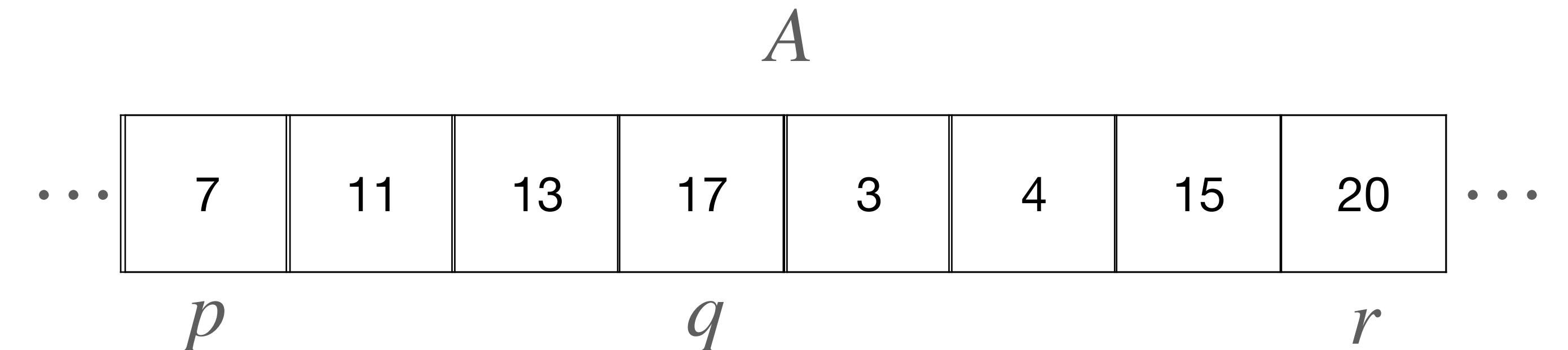
**if**  $L[i] \leq R[j]$

$A[k] = L[i]$

$i = i + 1$

**else**  $A[k] = R[j]$

$j = j + 1$



PRACTICAL IN-PLACE MERGESORT\*

JYRKI KATAJAINEN

Department of Computer Science, University of Copenhagen  
Universitetsparken 1, DK-2100 Copenhagen East, DENMARK  
Electronic mail: [jyrki@diku.dk](mailto:jyrki@diku.dk)

TOMI PASANEN†

Turku Centre for Computer Science,  
Lemminkäisenkatu 14A, FIN-20520 Turku, FINLAND  
Electronic mail: [Tomi.Pasanen@utu.fi](mailto:Tomi.Pasanen@utu.fi)

JUKKA TEUHOLA‡

University of Turku

# Pseudokode for mergesort

MERGE( $A, p, q, r$ )

```
n1 = q - p + 1
n2 = r - q
let L[1 .. n1 + 1] and R[1 .. n2 + 1] be new arrays
for i = 1 to n1
    L[i] = A[p + i - 1]
for j = 1 to n2
    R[j] = A[q + j]
L[n1 + 1] = ∞
R[n2 + 1] = ∞
i = 1
j = 1
for k = p to r
    if L[i] ≤ R[j]
        A[k] = L[i]
        i = i + 1
    else A[k] = R[j]
        j = j + 1
```

MERGE-SORT( $A, p, r$ )

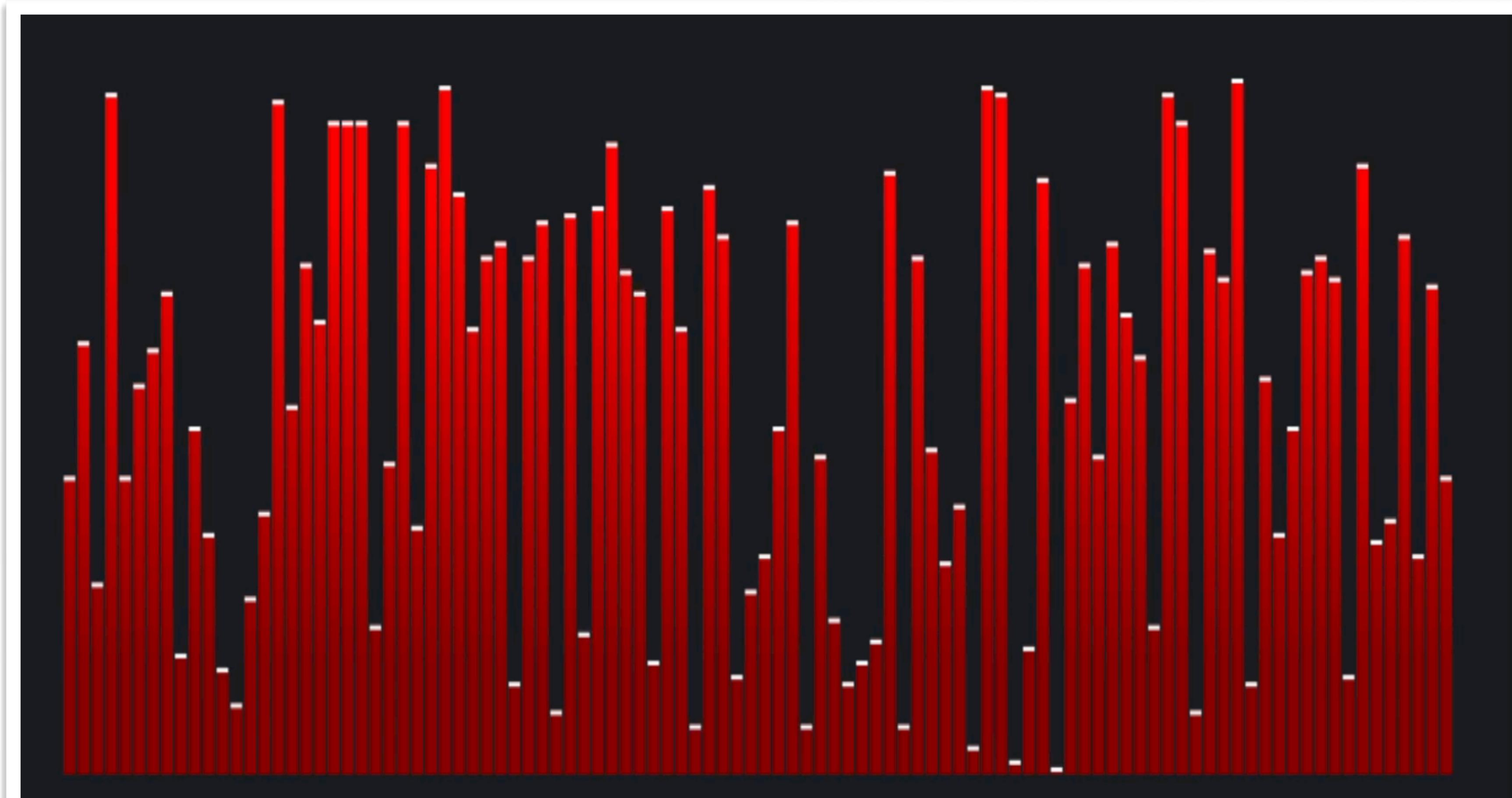
```
if p < r
    q = ⌊(p + r)/2⌋
    MERGE-SORT(A, p, q)           // check for base case
    MERGE-SORT(A, q + 1, r)       // divide
    MERGE(A, p, q, r)            // conquer
                                // conquer
                                // combine
```

- Antal elementer, der skal sorteres:

$$n = r - p + 1$$

$O(n)$  tid plus 2 rekursive problemer,  
størrelse  $\lfloor n/2 \rfloor$  og  $\lceil n/2 \rceil$

# Del-og-hersk eksempel: Quicksort



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

# Del-og-hersk eksempel: Quicksort

Pivot, der deler  
små og store

- **Idé:** Opdel input i “små” og “store”...

**PARTITION( $A, p, r$ )**

$x = A[r]$

$i = p - 1$

**for**  $j = p$  **to**  $r - 1$

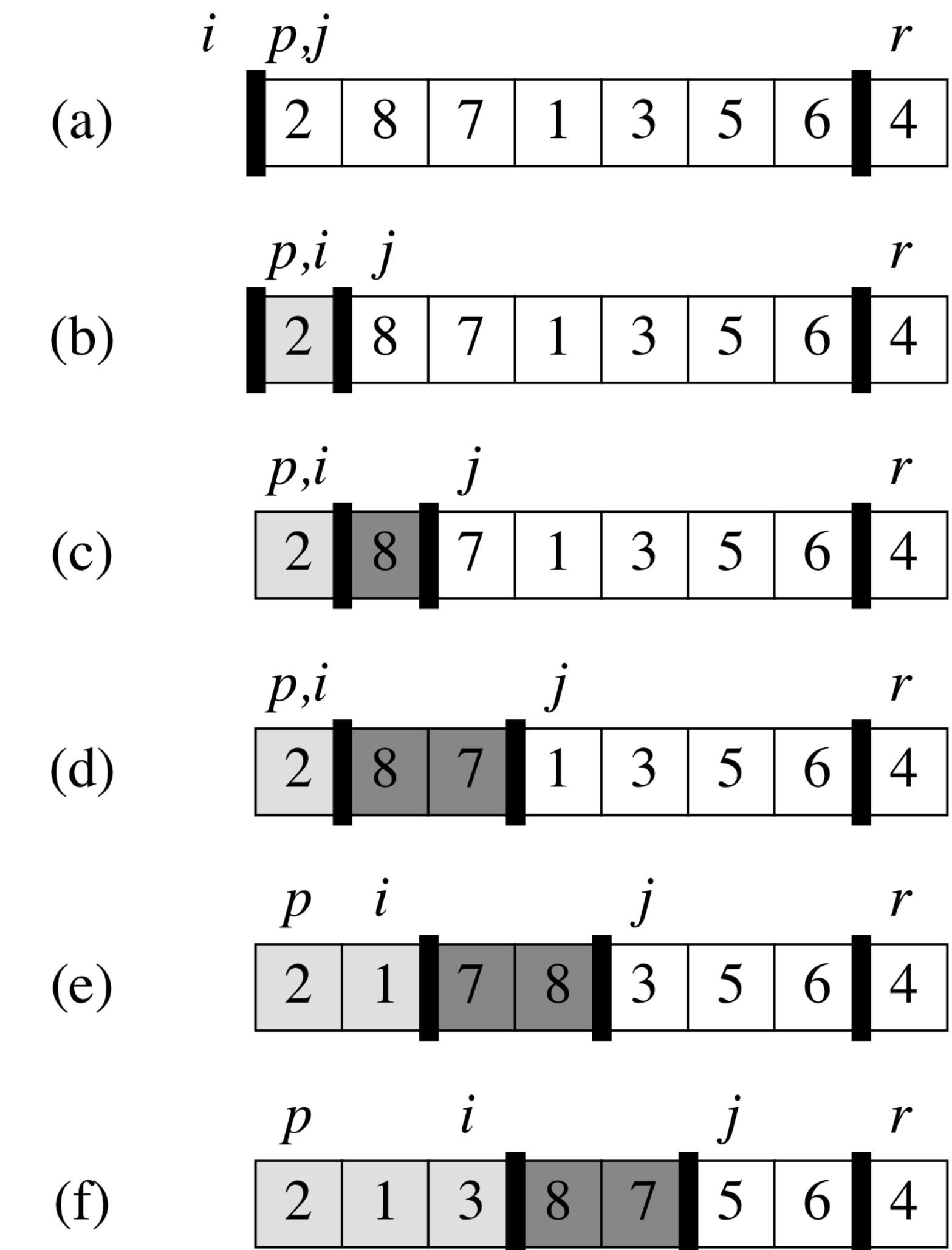
**if**  $A[j] \leq x$

$i = i + 1$

        exchange  $A[i]$  with  $A[j]$

    exchange  $A[i + 1]$  with  $A[r]$

**return**  $i + 1$



# Del-og-hersk eksempel: Quicksort

- **Idé:** Opdel input i “små” og “store”, og sortér hver del rekursivt

PARTITION( $A, p, r$ )

$x = A[r]$

$i = p - 1$

**for**  $j = p$  **to**  $r - 1$

**if**  $A[j] \leq x$

$i = i + 1$

        exchange  $A[i]$  with  $A[j]$

    exchange  $A[i + 1]$  with  $A[r]$

**return**  $i + 1$

QUICKSORT( $A, p, r$ )

**if**  $p < r$

$q = \text{PARTITION}(A, p, r)$

        QUICKSORT( $A, p, q - 1$ )

        QUICKSORT( $A, q + 1, r$ )

- Antal elementer der skal sorteres:

$$n = r - p + 1$$

$O(n)$  tid plus 2 rekursive problemer af *total* størrelse  $n$

# Del-og-hersk eksempel: Maksimum deltabel

- Du bestyrer en iskiosk og har en tabel  $A[0..n]$  der for hver dato indeholder den forventede fortjeneste (positiv eller negativ) hvis du holder åbent.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
A	13	-3	-25	20	-3	-16	-23	18	20	-7	12	-5	-22	15	-4	7

maximum subarray

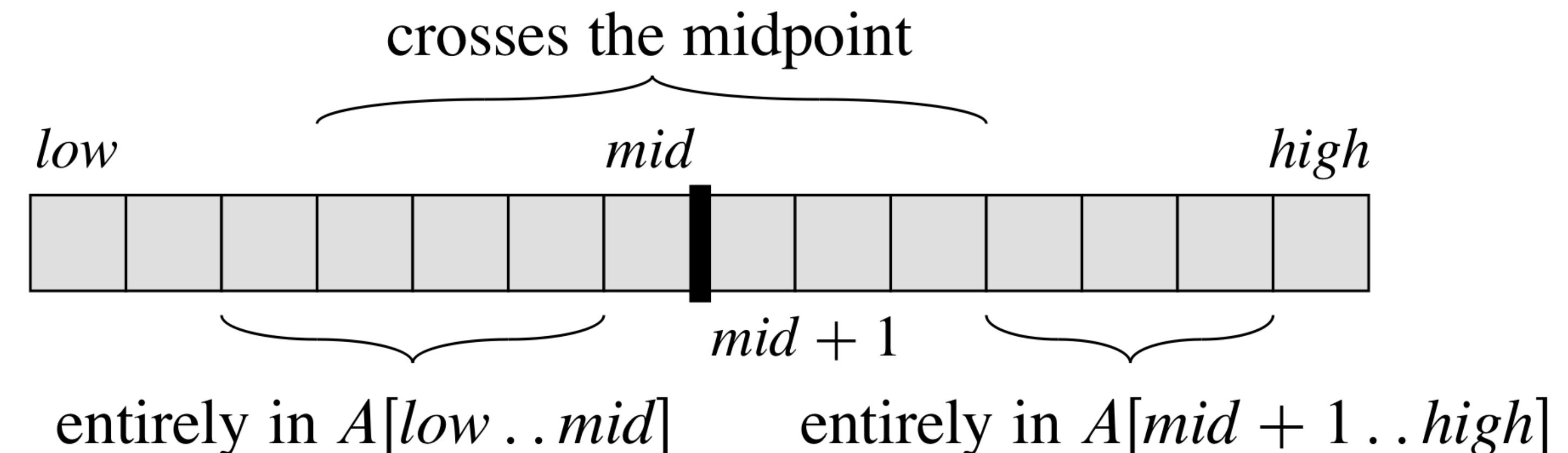
- Maksimum deltabel problemet:** Find et bedste tidsinterval  $[i, j]$  at have din isvogn åben.

Dvs. værdier  $i \leq j$  der maksimerer  $z_{ij} = \sum_{k=i}^j A[k]$



# Del-og-hersk eksempel: Maksimum deltabel

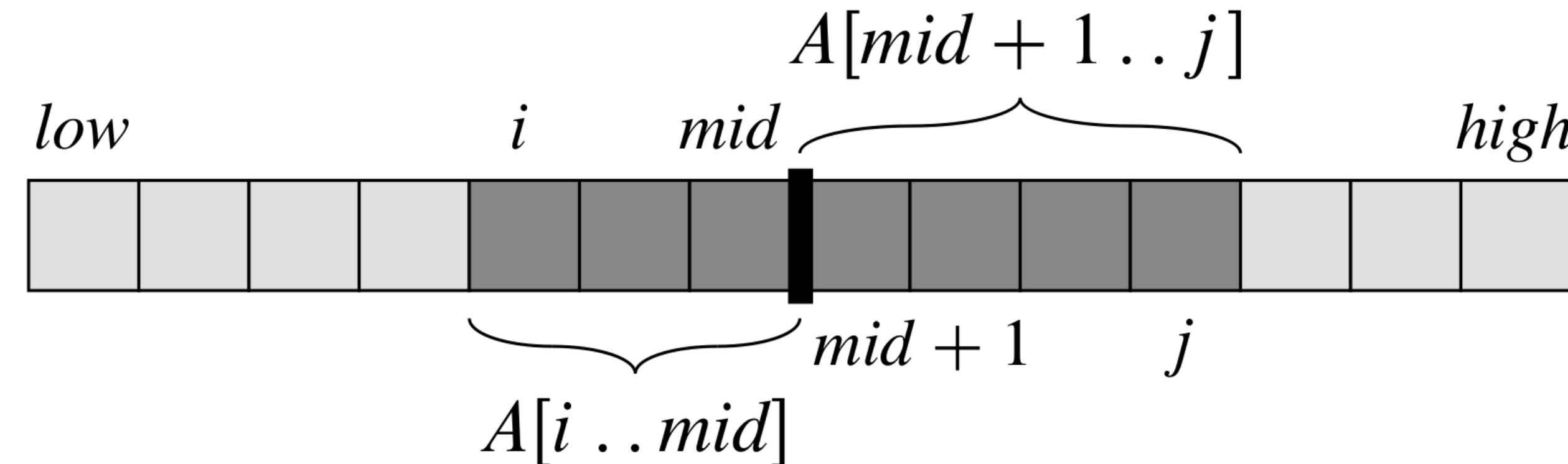
- Find værdier  $i \leq j$  der maksimerer  $z_{ij} = \sum_{k=i}^j A[k]$
- **Naiv algoritme:** Beregn  $z_{ij}$  for alle  $1 \leq i \leq j \leq n$ .  
Tidsforbrug  $O(n^2) \cdot O(n) = O(n^3)$ .
- Tre typer af optimal løsning:
  1.  $i \leq n/2 < j$
  2.  $i \leq j \leq n/2$
  3.  $n/2 < i \leq j$



# Øvelse: Maksimum del tabel

- Hvordan finder vi en optimal løsning af type 1?

- Dvs. find værdier  $i \leq n/2 < j$  der maksimerer  $z_{ij} = \sum_{k=i}^j A[k]$



# Del-og-hersk eksempel: Maksimum deltabel

```
FIND-MAX-CROSSING-SUBARRAY( $A, low, mid, high$ )
```

// Find a maximum subarray of the form  $A[i \dots mid]$ .

$left-sum = -\infty$

$sum = 0$

**for**  $i = mid$  **downto**  $low$

$sum = sum + A[i]$

**if**  $sum > left-sum$

$left-sum = sum$

$max-left = i$

// Find a maximum subarray of the form  $A[mid + 1 \dots j]$ .

$right-sum = -\infty$

$sum = 0$

**for**  $j = mid + 1$  **to**  $high$

$sum = sum + A[j]$

**if**  $sum > right-sum$

$right-sum = sum$

$max-right = j$

// Return the indices and the sum of the two subarrays.

**return** ( $max-left, max-right, left-sum + right-sum$ )

```
FIND-MAXIMUM-SUBARRAY( $A, low, high$ )
```

**if**  $high == low$

**return** ( $low, high, A[low]$ )

// base case: only one element

**else**  $mid = \lfloor (low + high)/2 \rfloor$

( $left-low, left-high, left-sum$ ) =

FIND-MAXIMUM-SUBARRAY( $A, low, mid$ )

( $right-low, right-high, right-sum$ ) =

FIND-MAXIMUM-SUBARRAY( $A, mid + 1, high$ )

( $cross-low, cross-high, cross-sum$ ) =

FIND-MAX-CROSSING-SUBARRAY( $A, low, mid, high$ )

**if**  $left-sum \geq right-sum$  and  $left-sum \geq cross-sum$

**return** ( $left-low, left-high, left-sum$ )

**elseif**  $right-sum \geq left-sum$  and  $right-sum \geq cross-sum$

**return** ( $right-low, right-high, right-sum$ )

**else return** ( $cross-low, cross-high, cross-sum$ )

$O(n)$  tid plus 2 rekursive problemer,  
størrelse  $\lfloor n/2 \rfloor$  og  $\lceil n/2 \rceil$

# Multiplikation af store tal

- **Input:**  $A$  og  $B$ ,  $n$ -bit heltal
- **Output:** Produktet  $C = A \cdot B$  (et  $2n$ -bit heltal)
- **Skolemetoden (binær version):**
  - Tag én bit af  $A$  ad gangen, gang med  $B$
- **Eksempel:**
  - $9 \cdot 11$  beregnes som  $1001_2 \cdot 1011_2$
  - Tidsskompleksitet  $O(n^2)$

# Multiplikation af store tal med del-og-hersk

- **Input:**  $A$  og  $B$ ,  $n$ -bit heltal
- **Output:** Produktet  $C = A \cdot B$  (et  $2n$ -bit heltal)
- **Rekursiv formulering:**
  - Antag at  $n = 2^i$
  - Find  $A_1, A_0, B_1, B_0$  ( $\frac{n}{2}$ -bit heltal) så  $A = A_1 2^{n/2} + A_0$  og  $B = B_1 2^{n/2} + B_0$
  - Kan beregne

$$C = (A_1 2^{n/2} + A_0) \cdot (B_1 2^{n/2} + B_0) = Z_2 \cdot 2^n + Z_1 \cdot 2^{n/2} + Z_0$$

hvor  $Z_2 = A_1 \cdot B_1$ ,  $Z_1 = A_1 \cdot B_0 + A_0 \cdot B_1$ ,  $Z_0 = A_0 \cdot B_0$  beregnes rekursivt

# Multiplikation af store tal med del-og-hersk

- **Rekursiv formulering:**

$$C = (A_1 2^{n/2} + A_0) \cdot (B_1 2^{n/2} + B_0) = Z_2 \cdot 2^n + Z_1 \cdot 2^{n/2} + Z_0$$

hvor  $Z_2 = A_1 \cdot B_1$ ,  $Z_1 = A_1 \cdot B_0 + A_0 \cdot B_1$ ,  $Z_0 = A_0 \cdot B_0$  beregnes rekursivt

- **Naivt:** 4 rekursive multiplikationer
- **Karatsuba:** 3 rekursive multiplikationer

$O(n)$  tid plus 3 rekursive problemer af samme størrelse,  $n/2$

- Beregn  $Z_2 = A_1 \cdot B_1$  og  $Z_0 = A_0 \cdot B_0$

$$\begin{aligned} \bullet \quad & \text{Beregn } Z_1 = Z_2 + Z_0 - (A_0 - A_1)(B_0 - B_1) \\ &= Z_2 + Z_0 - |A_0 - A_1| \cdot |B_0 - B_1| \text{ sign}((A_0 - A_1)(B_0 - B_1)) \end{aligned}$$

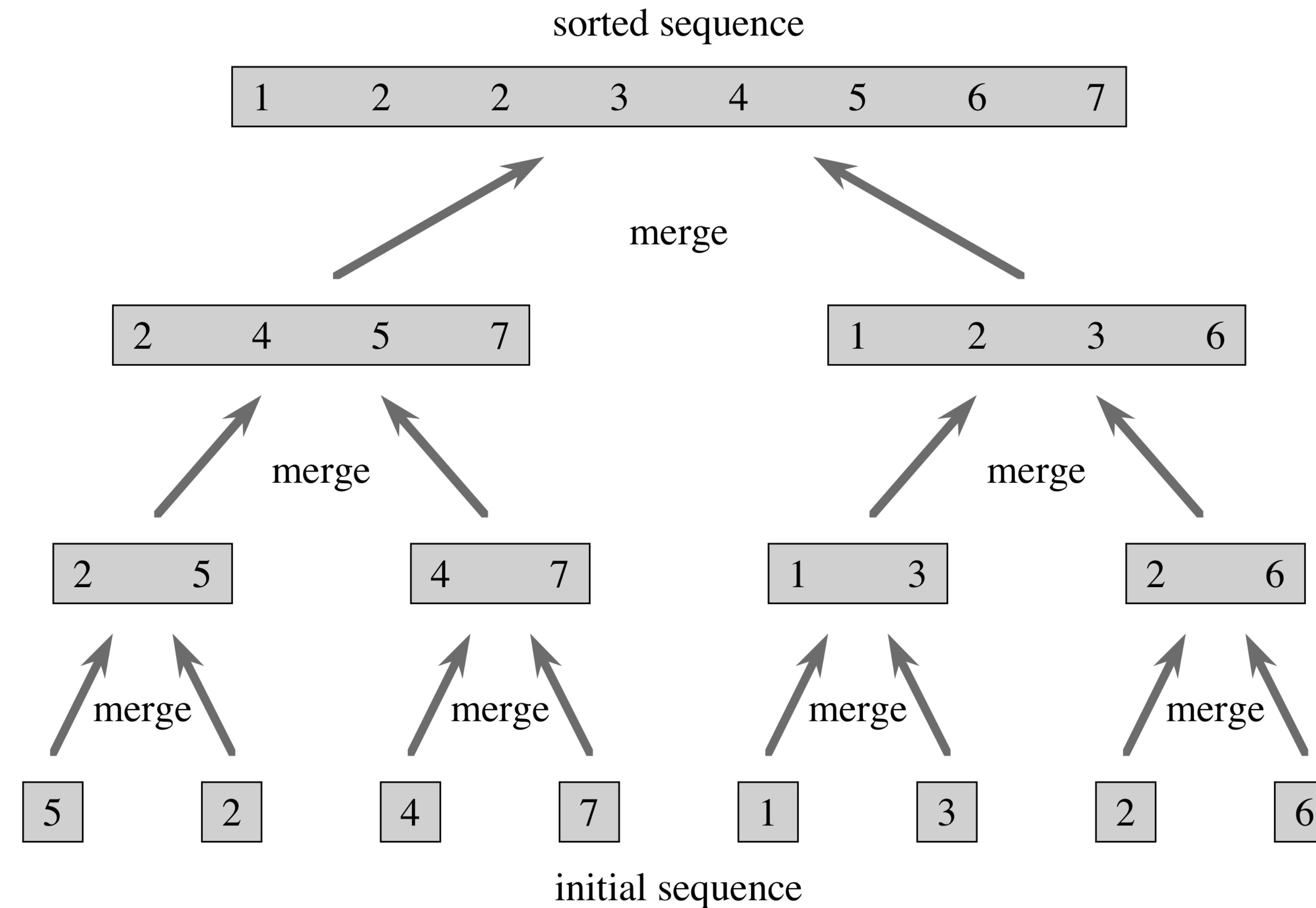
sign(.) beregner fortegnet,  $-1$  eller  $+1$

# Del 2: Tidsanalyse af del-og-hersk?

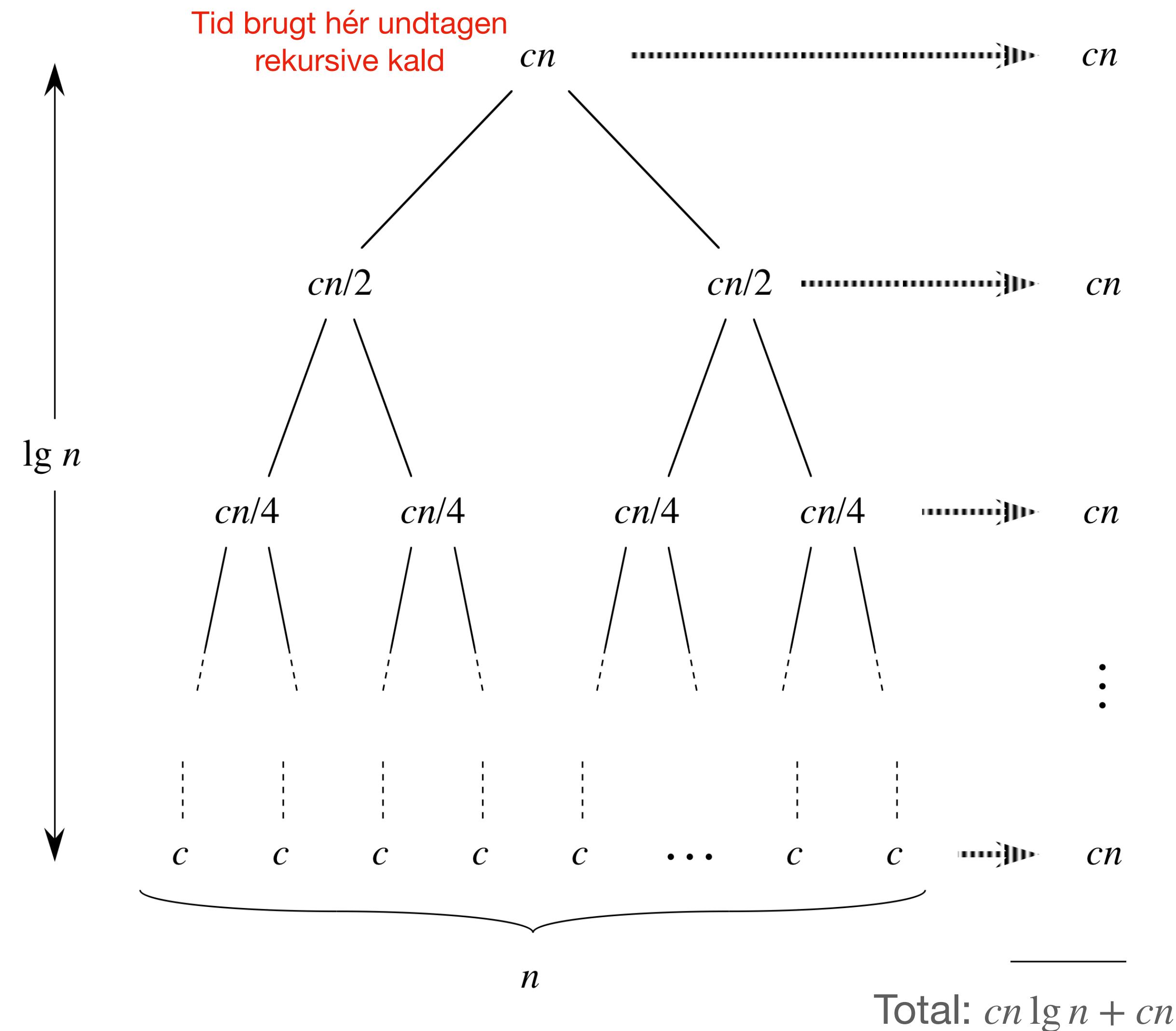
Tidsanalyserne kan skrives på lignende form

- Mergesort:  $O(n)$  tid plus 2 rekursive problemer af størrelse  $\approx n/2$
- Quicksort:  $O(n)$  tid plus 2 rekursive problemer af *total* størrelse  $n$
- Maksimal del tabel:  $O(n)$  tid plus 2 rekursive problemer af størrelse  $\approx n/2$
- Karatsuba multiplikation:  $O(n)$  tid plus 3 rekursive problemer af størrelse  $n/2$

# Rekursionstræ for mergesort



# “Tegn og gæt” tidsanalyse med rekursionstræ



# Rekursionsligning for mergesort

- Definér  $T(n)$  som tiden for mergesort på et værstefalds input
- Hvis  $n$  er *lige* har vi at  $T(n) = 2 T(n/2) + \Theta(n)$ , og desuden er  $T(1) = \Theta(1)$
- Mere generelt er  $T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n)$ , men lad os ignorere afrundingen (er uden betydning når  $n$  er en 2-potens, og også generelt)
- En lighed (eller en ulighed), der udtrykke  $T(n)$  som funktion af værdier af  $T(n')$  for  $n' < n$  kalder vi en *rekursionsligning*.

# Øvelse

- Lav rekursionstræer der svarer til disse rekursionsligninger, og forsøg at gætte en løsning.

$$1. T(n) \leq 4 T(n/2) + n$$

$$2. T(n) \leq 4 T(n/4) + n$$

$$3. T(n) \leq 2 T(n/2) + n^2$$

- I alle tre tilfælde gælder at  $T(n) \leq 1$  for  $n \leq 1$

# Sammenfatning af øvelse

1.  $T(n) \leq 4 T(n/2) + n$

Rekursionstræet har dybde  $\log_2 n$ , sum  $4^i$  på niveau  $i$ , totalt  $4^{\log_2 n} = O(n^2)$

*Det totale arbejde **stiger** med rekursionsniveauet*

2.  $T(n) \leq 4T(n/4) + n$ :

Rekursionstræet har dybde  $\log_4 n$ , sum  $n$  på hvert niveau, totalt  $O(n \log n)$

*Det totale arbejde er **det samme** på alle rekursionsniveauerne*

3.  $T(n) \leq 2 T(n/2) + n^2$

Rekursionstræet har dybde  $\log_2 n$ , sum  $n^2/2^i$  på niveau  $i$ , totalt  $O(n^2)$

*Det totale arbejde **falder** med rekursionsniveauet*

# Master method

Fungerer også hvis  $T(n/b)$  erstattes af  $T(\lceil n/b \rceil)$  eller  $T(\lfloor n/b \rfloor)$

- Giver løsning til de hyppigst forekommende rekursionsligninger, på formen:

$$T(n) = aT(n/b) + \Theta(f(n)), \text{ for konstanter } a \geq 1 \text{ og } b > 1$$

- Lad  $\varepsilon > 0$  være en konstant og antag at  $f(n)$  er ikke-aftagende. Så gælder:

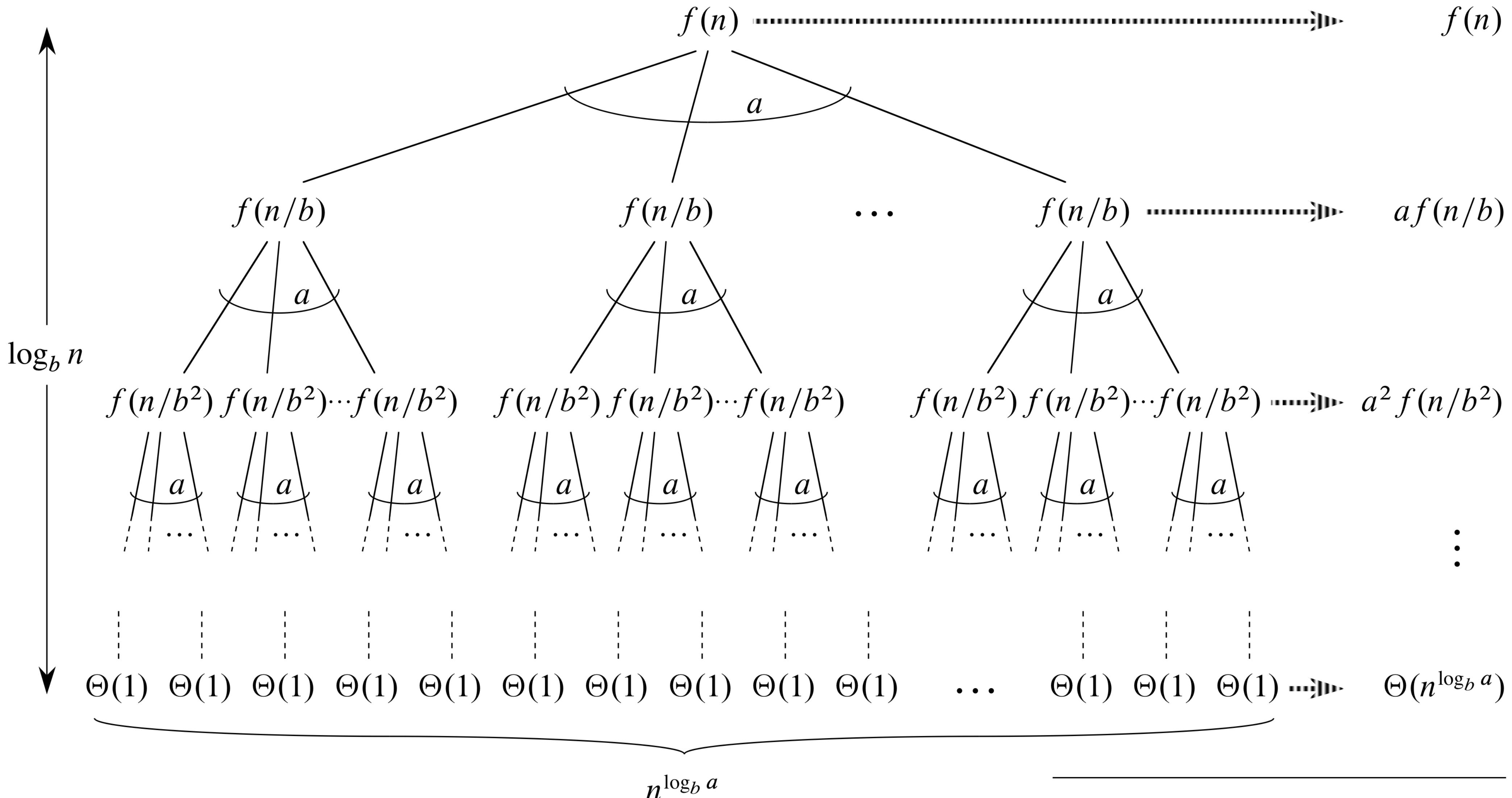
$$1. f(n) = O(n^{\log_b(a)-\varepsilon}) \implies T(n) = \Theta(n^{\log_b a})$$

Karatsuba multiplikation passer i dette tilfælde med  $a = 3, b = 2$

$$2. f(n) = \Theta(n^{\log_b(a)}) \implies T(n) = \Theta(n^{\log_b a} \log n)$$

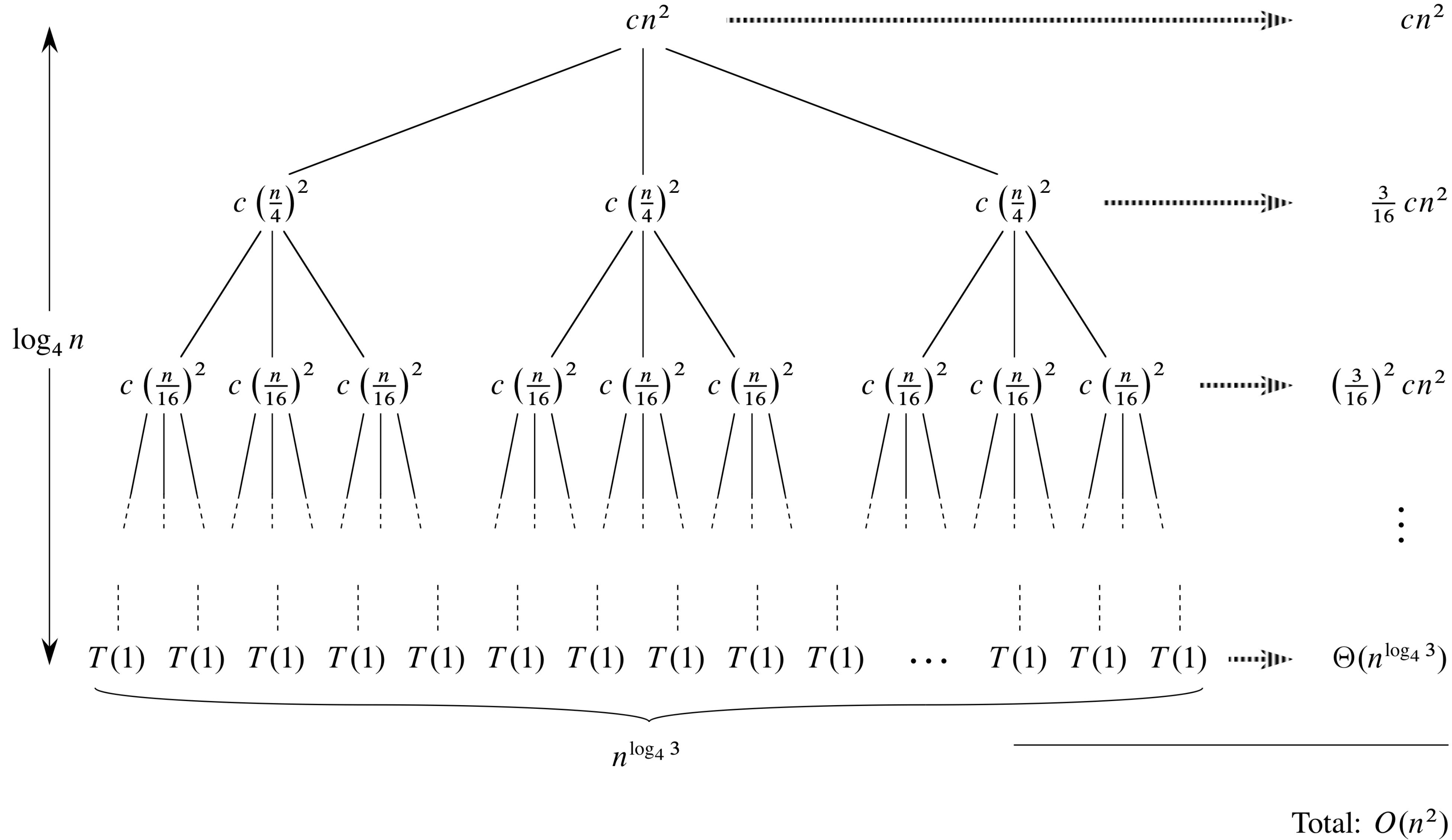
Mergesort og maximum del tabel passer i dette tilfælde med  $a = b = 2$

$$3. f(n) = \Omega(n^{\log_b(a)+\varepsilon}) \implies T(n) = \Theta(f(n))$$



# Illustration af tilfælde 1

$$\text{Total: } \Theta(n^{\log_b a}) + \sum_{j=0}^{\log_b n - 1} a^j f(n/b^j)$$



# Illustration af tilfælde 3

$a = 3, b = 4$

# Substitutionsmetoden

Teknik til at bekræfte en løsning til en rekursionsligning

- Start med et gæt på en løsning, fx  $T(n) = O(f(n))$
- Antag at  $T(n) \leq 1$  når  $n \leq k$   
(højst en konstant faktor forkert for konstant  $k$ )
- Vis ved induktion på  $n$  at  $T(n) \leq cf(n)$ , for  $n > k$  og passende konstant  $c$

# Eksempel på substitutionsmetoden

Lad os se på rekursionsligningen  $T(n) \leq 2 T(n/2) + cn$ , hvor  $c \geq 1$

- Gæt på en løsning:  $T(n) = O(n \lg n)$
- Antag at  $T(1) = 1$
- Vil gerne vise at  $T(n) \leq c n \lg(n) + cn$  ved induktion:
  - **Basis:**  $T(1) \leq c 1 \log(1) + c 1$  er sandt
  - **Induktionshypoteze:** Antag  $T(n') \leq c n' \lg(n') + cn'$  for  $n' < n$
  - **Induktionsskridt:** 
$$\begin{aligned} T(n) &\leq 2 T(n/2) + cn \\ &\leq 2(c(n/2)\lg(n/2) + c(n/2)) + cn \\ &= cn \lg(n) + cn \end{aligned}$$

Lægger  $cn$  til for at få uligheden til at holde i basistilfældet

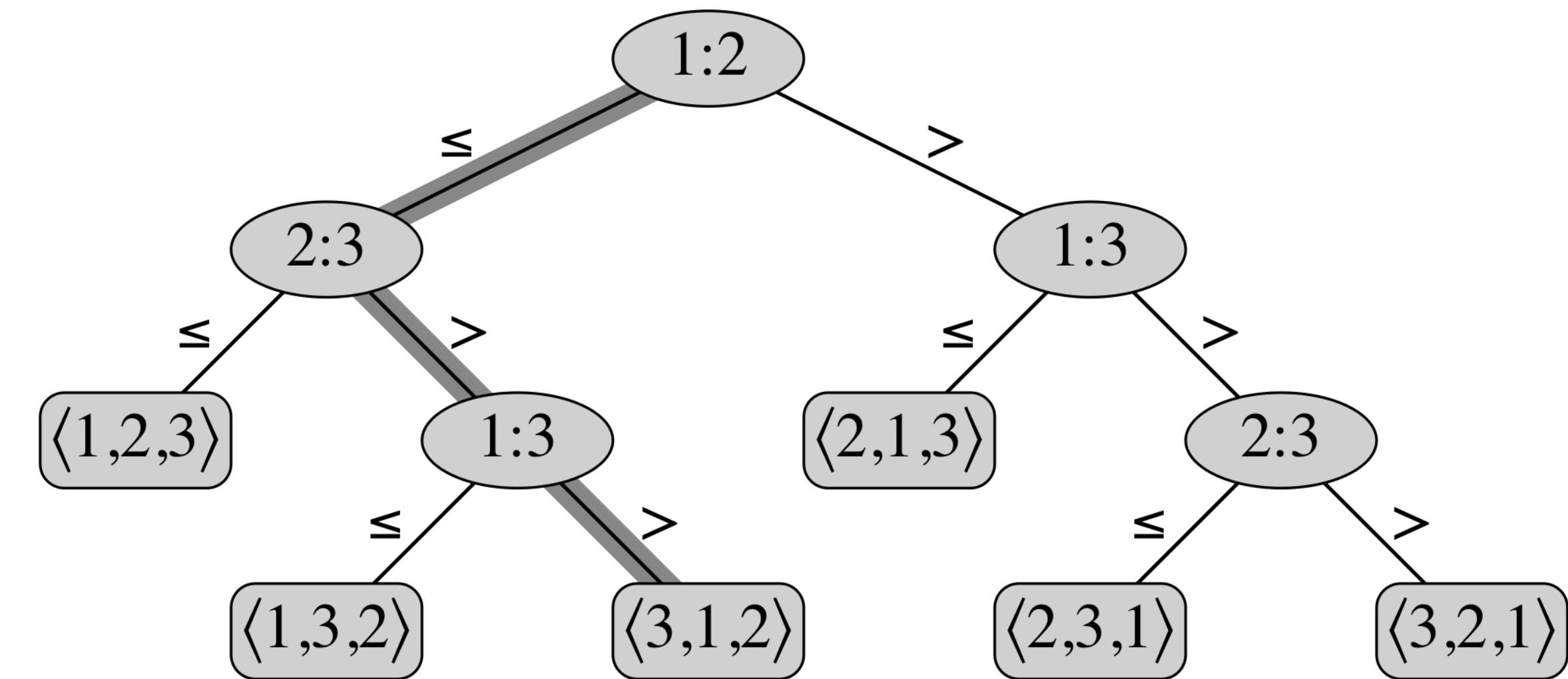
# Flere eksempler

1. For  $T(n) \leq 4T(n/2) + n$ , kan vise at  $T(n) \leq 2n^2 - n$  (gæt fra før:  $O(n^2)$ )
2. For  $T(n) \leq 2T(n/2) + 1$ , kan vise at  $T(n) \leq 2n - 1$  (gæt fra før:  $O(n)$ )
3. For  $T(n) \leq T(n - 1) + n$ , kan vise at  $T(n) \leq n^2$  (gæt:  $O(n^2)$ )

# Del 3: Nedre grænse for sorteringsalgoritmer

Sammenligningsbaseret model:

- I hvert skridt sammenligner vi ét par  $i : j$  af elementer  $A[i]$  og  $A[j]$  fra input
- Valget af  $i$  og  $j$  afhænger af resultaterne af de tidlige sammenligninger
- Et *beslutningstræ* kan vise alle måder programmet kan køre på — bladene svarer til outputs (permutationer af input)

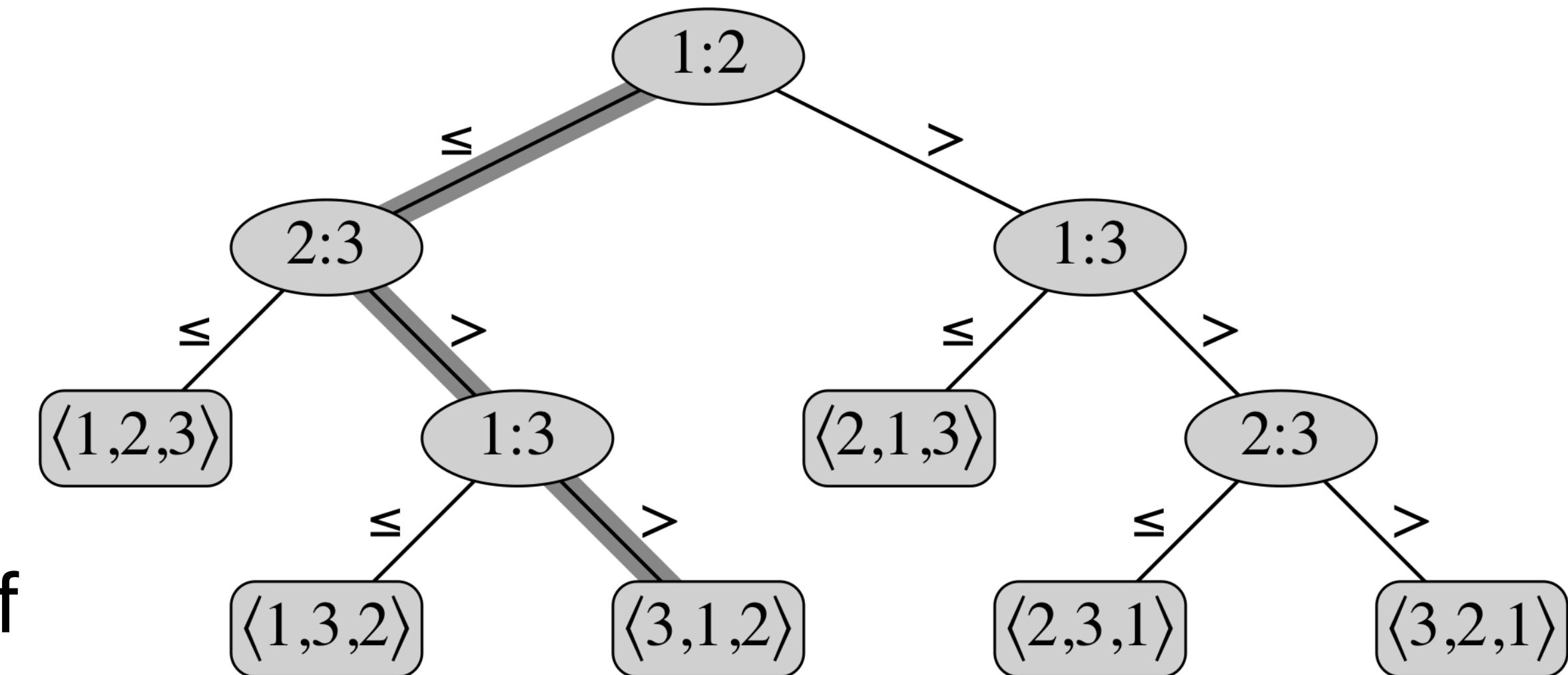


Modellen fanger de fleste sorteringsalgoritmer, som fx mergesort og quicksort, men ikke specialiserede algoritmer til heltal som fx radix sort

# Nedre grænse for sorterings

## Observationer

- Beslutningstræet er *binært*, dvs. antal blade på niveau  $1, \dots, i$  er højest  $2^i$ .
- Lad  $\sigma_1, \sigma_2, \dots, \sigma_n$  være en permutation af  $\{1, \dots, n\}$ , og betragt et input hvor  $A[\sigma_i] = i$ : Output skal være  $\langle \sigma_1, \dots, \sigma_n \rangle$
- Dvs. der er  $n! = n \cdot (n - 1) \cdot \dots \cdot 2 \cdot 1$  forskellige outputs, og derfor  $\geq n!$  blade.



$$2^{\text{dybde}} \geq \#\text{blade} \geq n!$$

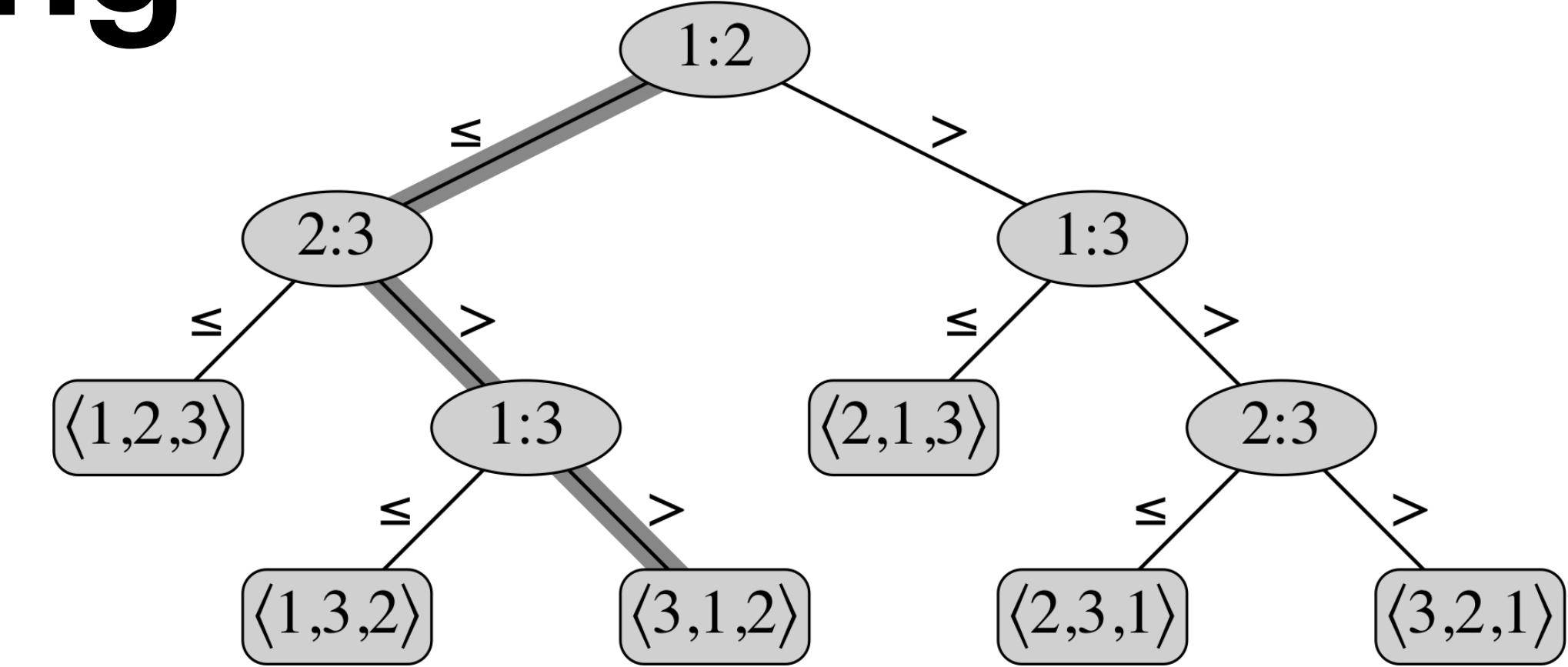
$$\Rightarrow \text{dybde} \geq \log_2(n!)$$

$$\Rightarrow \text{antal sammenligninger} \geq \log_2(n!)$$

# Nedre grænse for sorterings

## Værstefalds antal sammenligninger:

- $\log_2(n!)$   
 $= \log_2(n) + \log_2(n - 1) + \dots + \log_2(1)$   
 $\geq \log_2(n) + \log_2(n - 1) + \dots + \log_2(n/2)$   
 $= \frac{n}{2} \log_2(n/2)$   
 $= \frac{\tilde{n}}{2} \log_2(n) - n/2$   
 $= \Omega(n \log n)$
- En mere omhyggelig udregning viser  
 $\log_2(n!) \geq n \log_2 n - \log_2(e)n$



Lower Bounds for Sorting 16, 17, and 18 Elements

Florian Stober\*

Armin Weiß\*

### Abstract

It is a long-standing open question to determine the minimum number of comparisons  $S(n)$  that suffice to sort an array of  $n$  elements. Indeed, before this work,  $S(n)$  has been known only for  $n \leq 22$  with the exception of  $n = 16$ , 17, and 18.

In this work, we fill that gap by proving that sorting  $n = 16$ , 17, and 18 elements requires 46, 50, and 54 comparisons respectively. This fully determines  $S(n)$  for these values and disproves a conjecture by Knuth that  $S(16) = 45$ . Moreover, we show that for sorting 28 elements at least 99 comparisons are needed.

We obtain our result via an exhaustive computer search which extends previous work by Wells (1965) and Peczarski (2002, 2004, 2007, 2012). Our progress is both based on advances in hardware and on novel algorithmic ideas such as applying a bidirectional search to this problem.

the Ford-Johnson algorithm was optimal. Manacher [8] has been the first to show that this is not the case, and there are infinitely many  $n$  for which  $F(n) > S(n)$ . Shortly after, it has been shown that the Ford-Johnson algorithm is not optimal for  $n = 47$  [14]. It is not known whether there is any  $n < 47$  for which  $F(n) > S(n)$ .

**Lower Bounds via Computer Search.** The first work to tackle the lower bound problem using an exhaustive computer search dates back to 1964, when Wells showed that  $S(12) > C(12) = 29$  – implying that  $S(12) = F(12) = 30$  [15, 17].

His work is the foundation of many subsequent papers including the present. The algorithm explores the full search space of possible comparisons to sort some array applying some clever pruning techniques based on the number of linear extensions of intermediate partially ordered sets (posets) – for details see Section 3.

In 1994, Kasai, Sawato, and Iwata succeeded to

# Næste skridt

- Husk ugeopgave 1, deadline næste mandag **kl. 13**
- Øvelser onsdag om del og hersk, rekursionsligninger
- **Næste forelæsning, onsdag:**
  - Amortiseret analyse