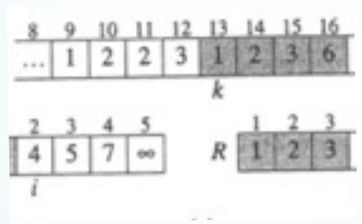
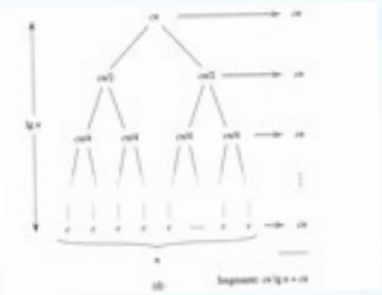
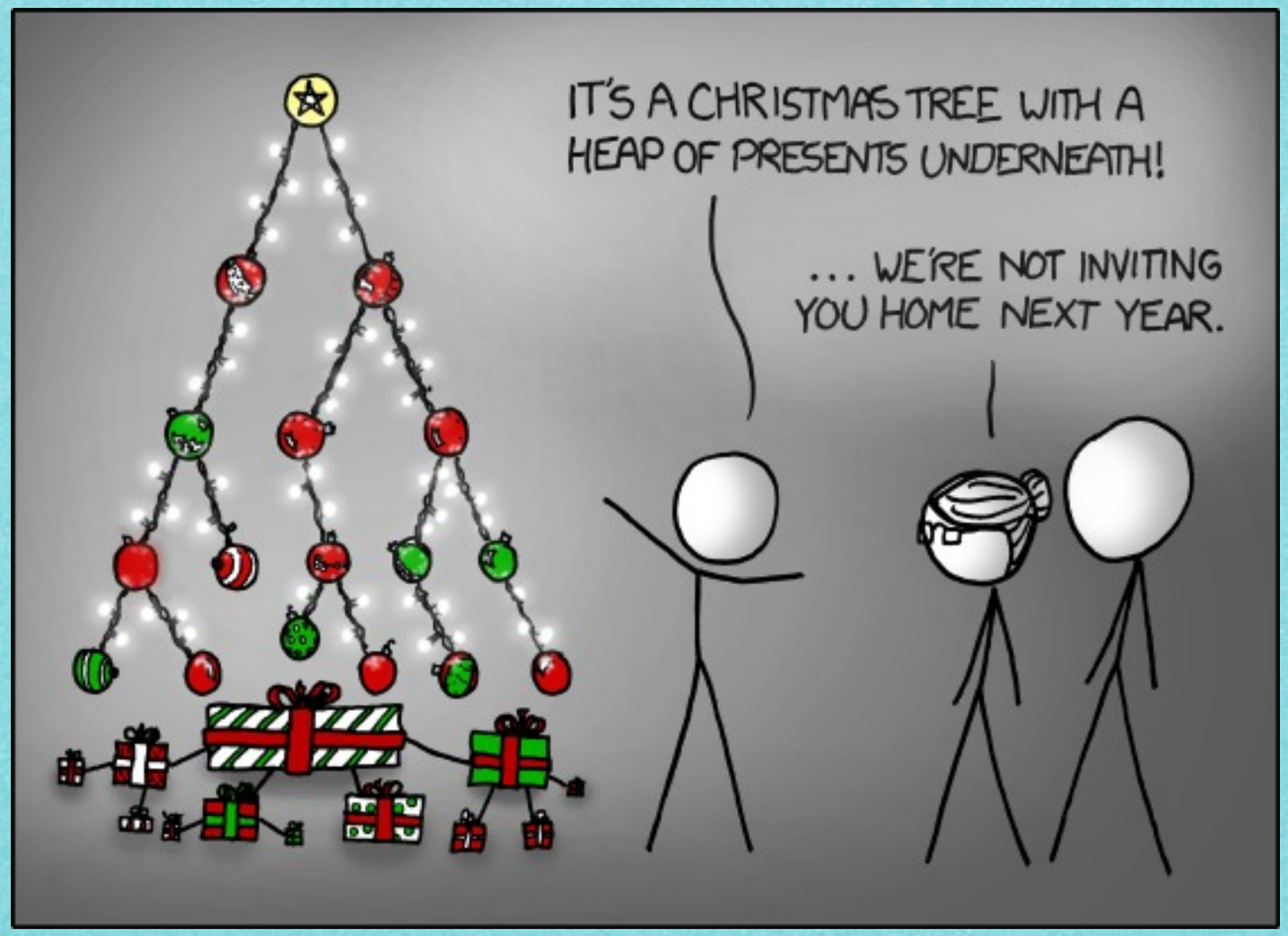




Kapitel 5: Sortieren

Algorithmen und Datenstrukturen
WS 2016/17

Prof. Dr. Sándor Fekete


IT'S A CHRISTMAS TREE WITH A HEAP OF PRESENTS UNDERNEATH!
... WE'RE NOT INVITING YOU HOME NEXT YEAR.


Vorsicht: KNUTH

THE CLASSIC WORK
NEWLY UPDATED AND REVISED

The Art of Computer Programming

VOLUME 3

Sorting and Searching

Second Edition

DONALD E. KNUTH

5.1 Aufgabenstellung

Sortieren von Objekten

Gegeben: n Objekte unterschiedlicher Größe

23 17 13 19 33 28 15

Gesucht: Eine Sortierung nach Größe

Sortieren von Objekten

Gegeben: n Objekte unterschiedlicher Größe

13 15 17 19 23 28 33

Gesucht: Eine Sortierung nach Größe

Sortieren von Objekten

Gegeben: n Objekte unterschiedlicher Größe

13 15 17 19 23 28 33

Zahl der Vergleiche: $O(n^2)$

„Geht‘s nicht noch etwas schneller?“

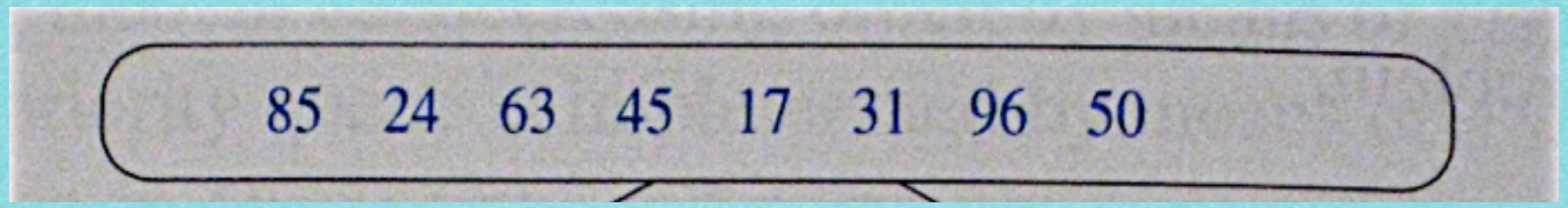
Sortieren von Objekten

Gegeben: n Objekte unterschiedlicher Größe

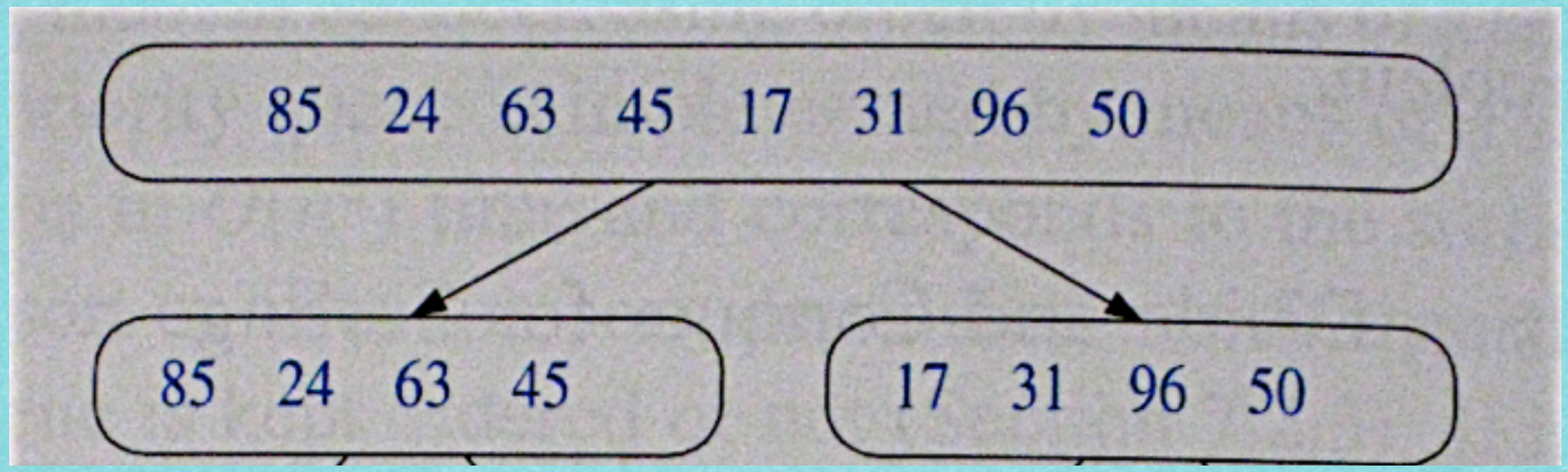
13 15 17 19 23 28 33

Zahl der Vergleiche: $O(n \log n)$

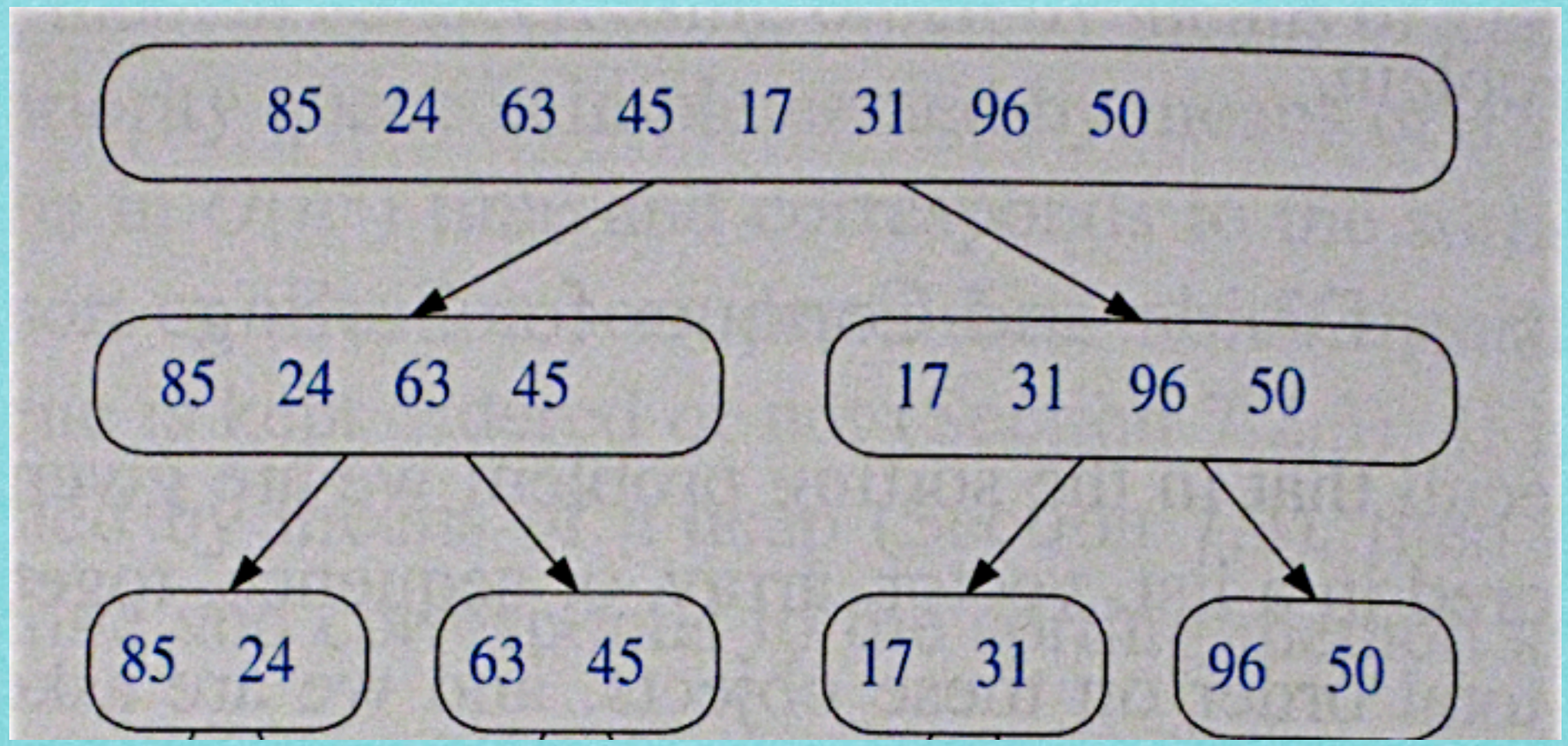
5.2 Mergesort



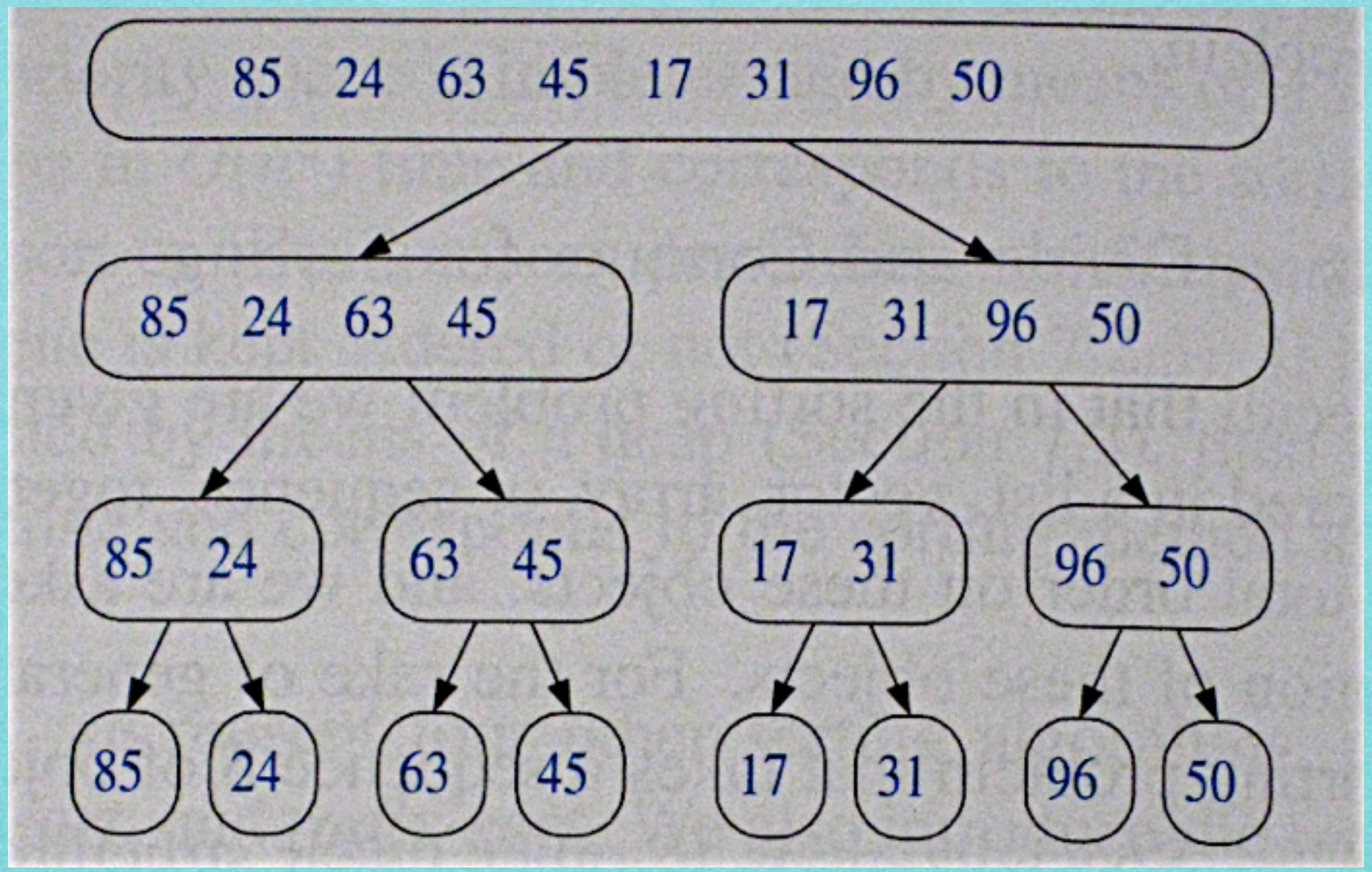
5.2 Mergesort



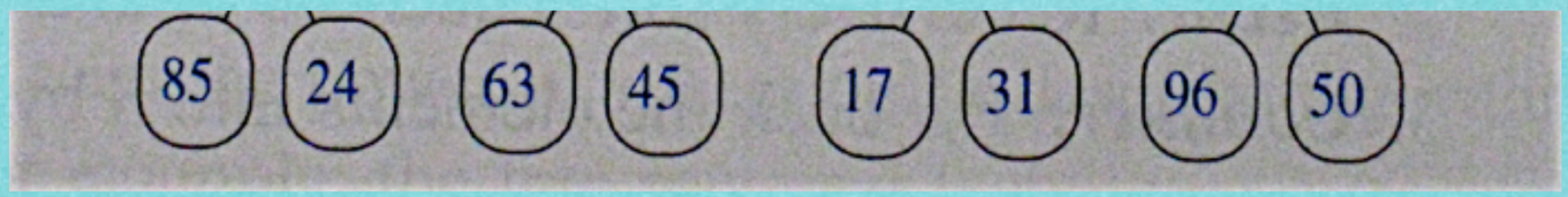
5.2 Mergesort



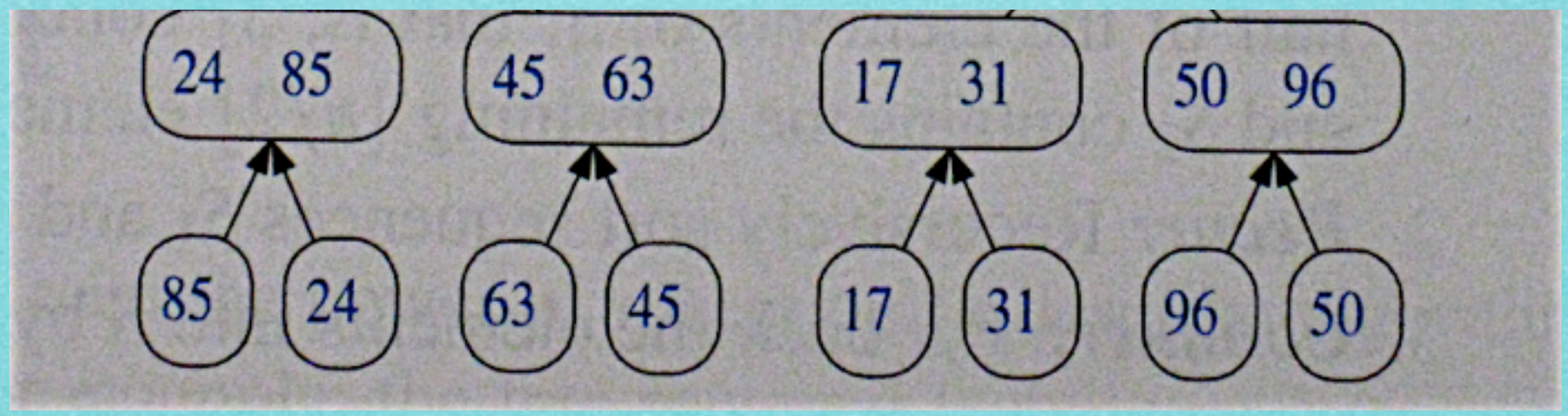
5.2 Mergesort



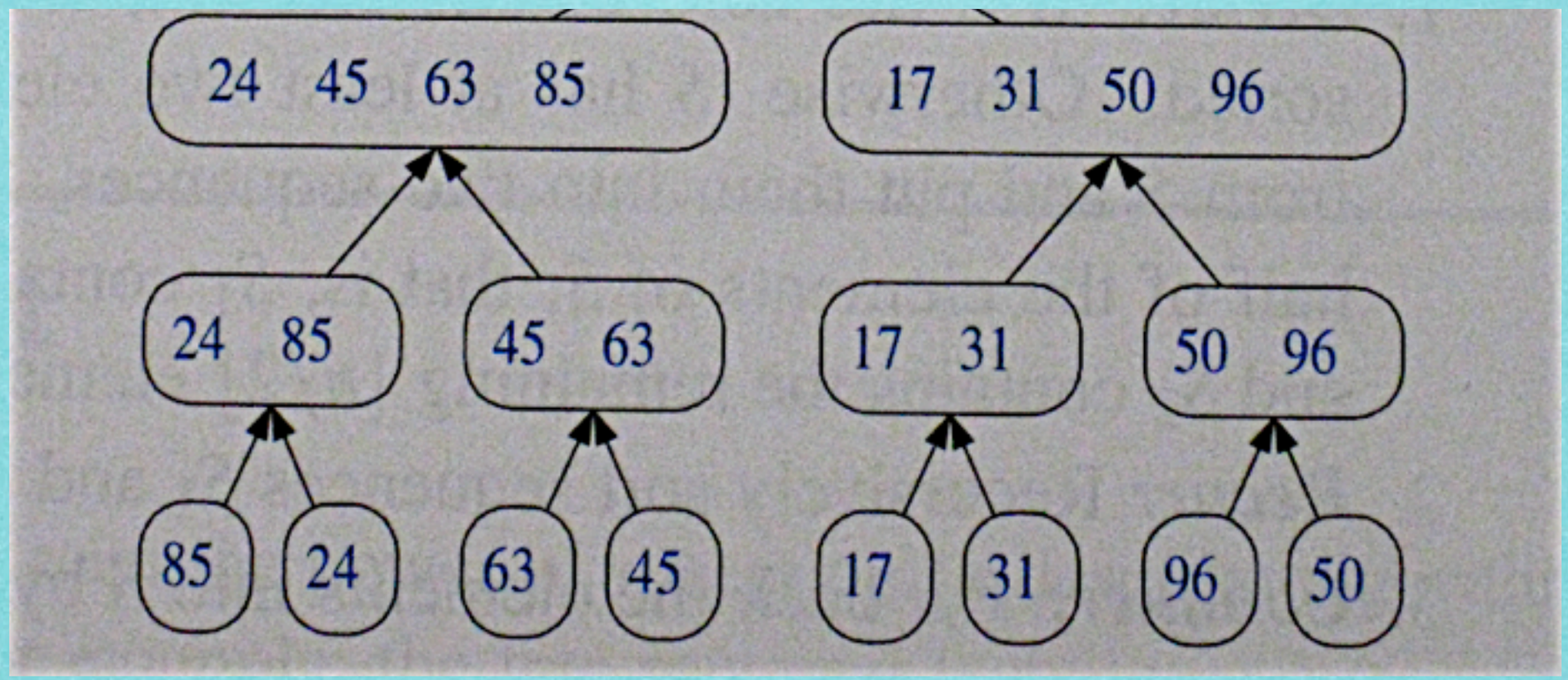
5.2 Mergesort



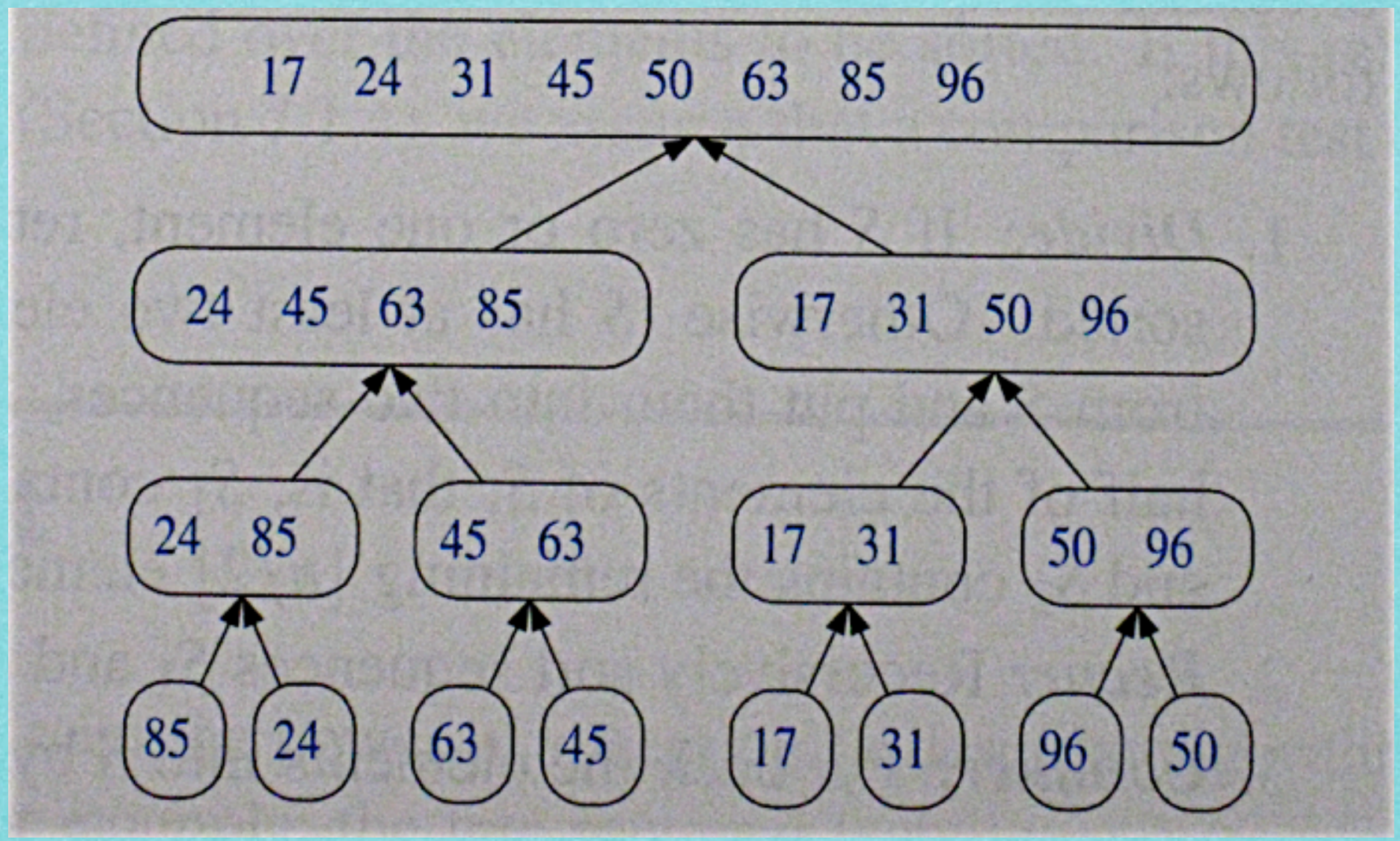
5.2 Mergesort



5.2 Mergesort



5.2 Mergesort



5.2.2 Algorithmische Beschreibung

Algorithmus 5.1

INPUT: Subarray von A=[1,...,n],
der bei Index p beginnt und bei Index r endet, d.h. A[p,...,r]

OUTPUT: Sortierter Subarray

MERGE-SORT(A,p,r)

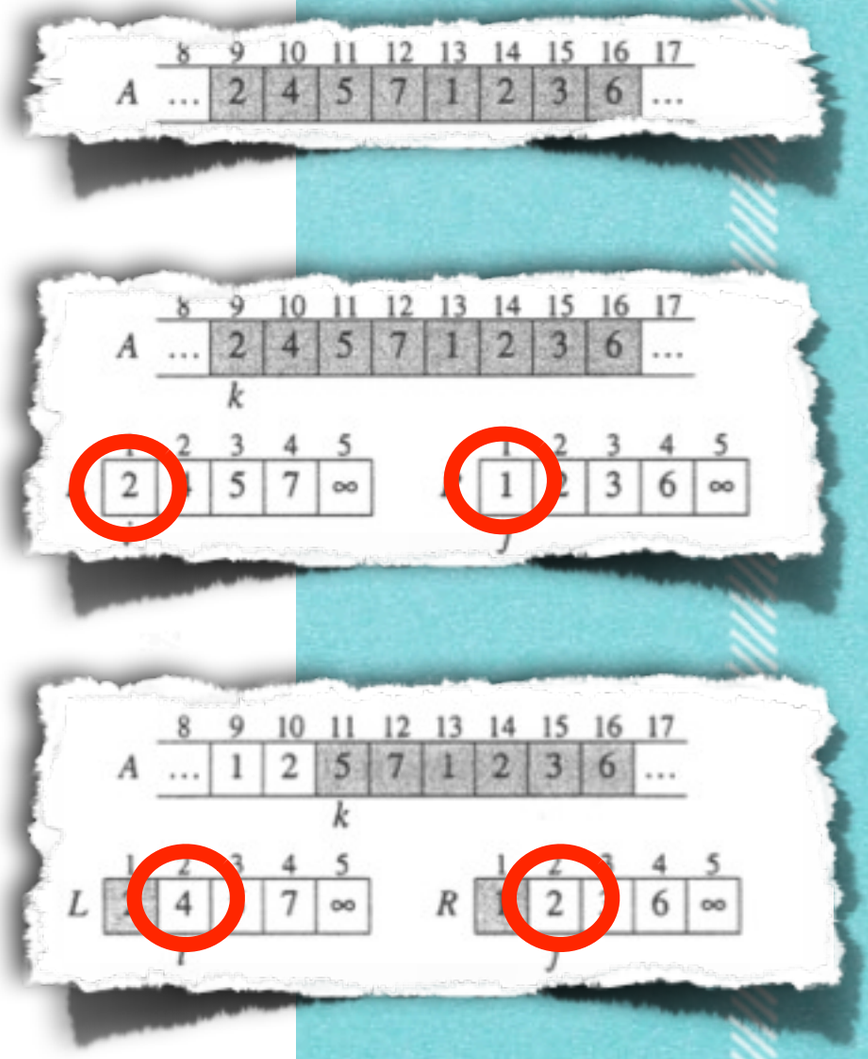
```
1  if p < r
2      then q ← ⌊(p + r)/2⌋
3           MERGE-SORT(A, p, q)
4           MERGE-SORT(A, q + 1, r)
5           MERGE(A, p, q, r)
```

Subroutine 5.2

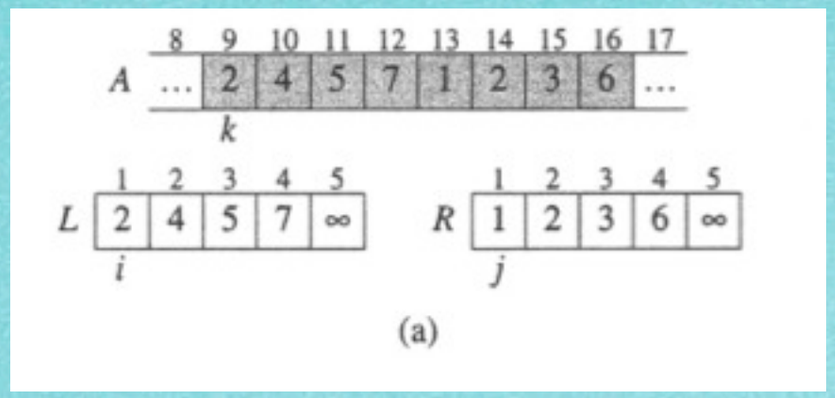
INPUT: Zwei sortierte Subarrays von A=[1,...,n], d.h. A[p,...,q] und A[q+1,...,r]

OUTPUT: Sortierter Subarray A[p,...,r]

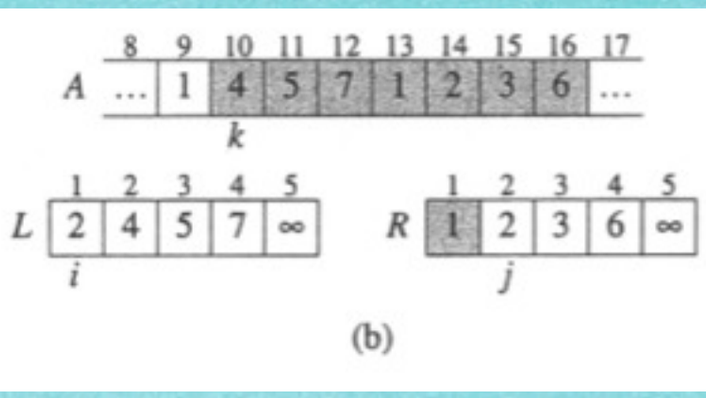
```
MERGE(A, p, q, r)
 1  n1 ← q − p + 1
 2  n2 ← r − q
 3  erzeuge die Felder L[1 .. n1 + 1] und R[1 .. n2 + 1]
 4  for i ← 1 to n1
 5        do L[i] ← A[p + i − 1]
 6  for j ← 1 to n2
 7        do R[j] ← A[q + j]
 8  L[n1 + 1] ← ∞
 9  R[n2 + 1] ← ∞
10  i ← 1
11  j ← 1
12  for k ← p to r
13        do if L[i] ≤ R[j]
14              then A[k] ← L[i]
15                   i ← i + 1
16              else A[k] ← R[j]
17                   j ← j + 1
```



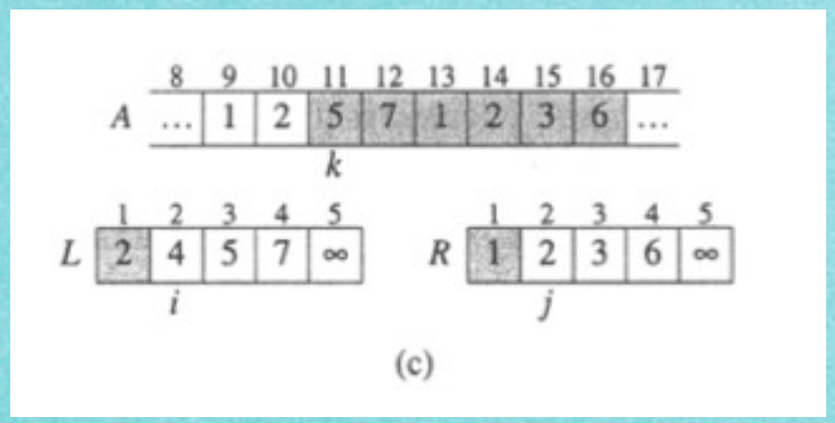
5.2 Mergesort



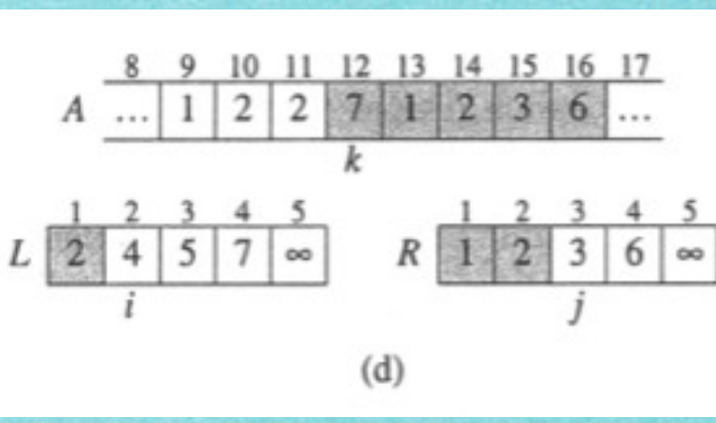
(a)



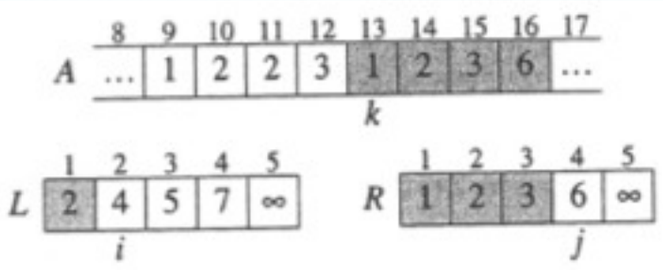
(b)



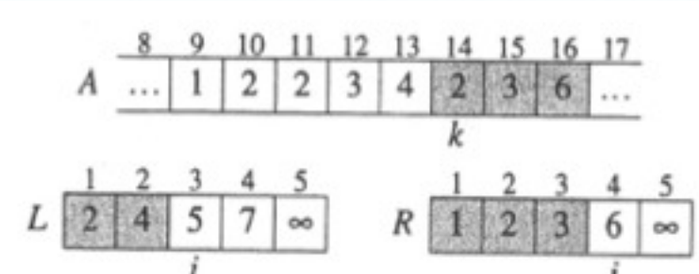
(c)



(d)

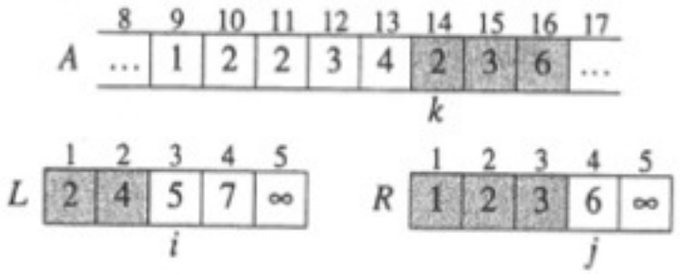


(e)

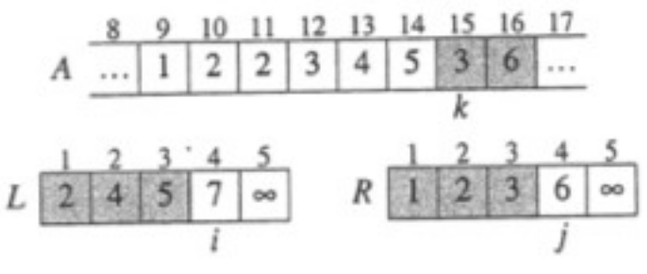


(f)

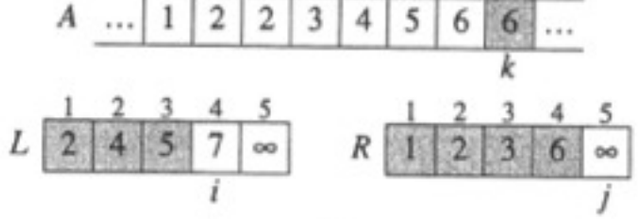
5.2 Mergesort



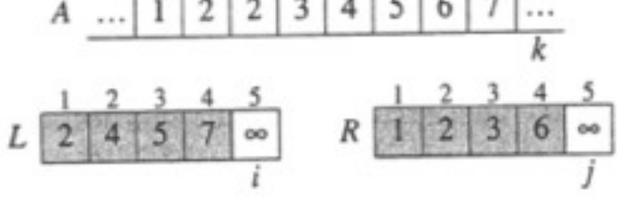
(f)



(g)



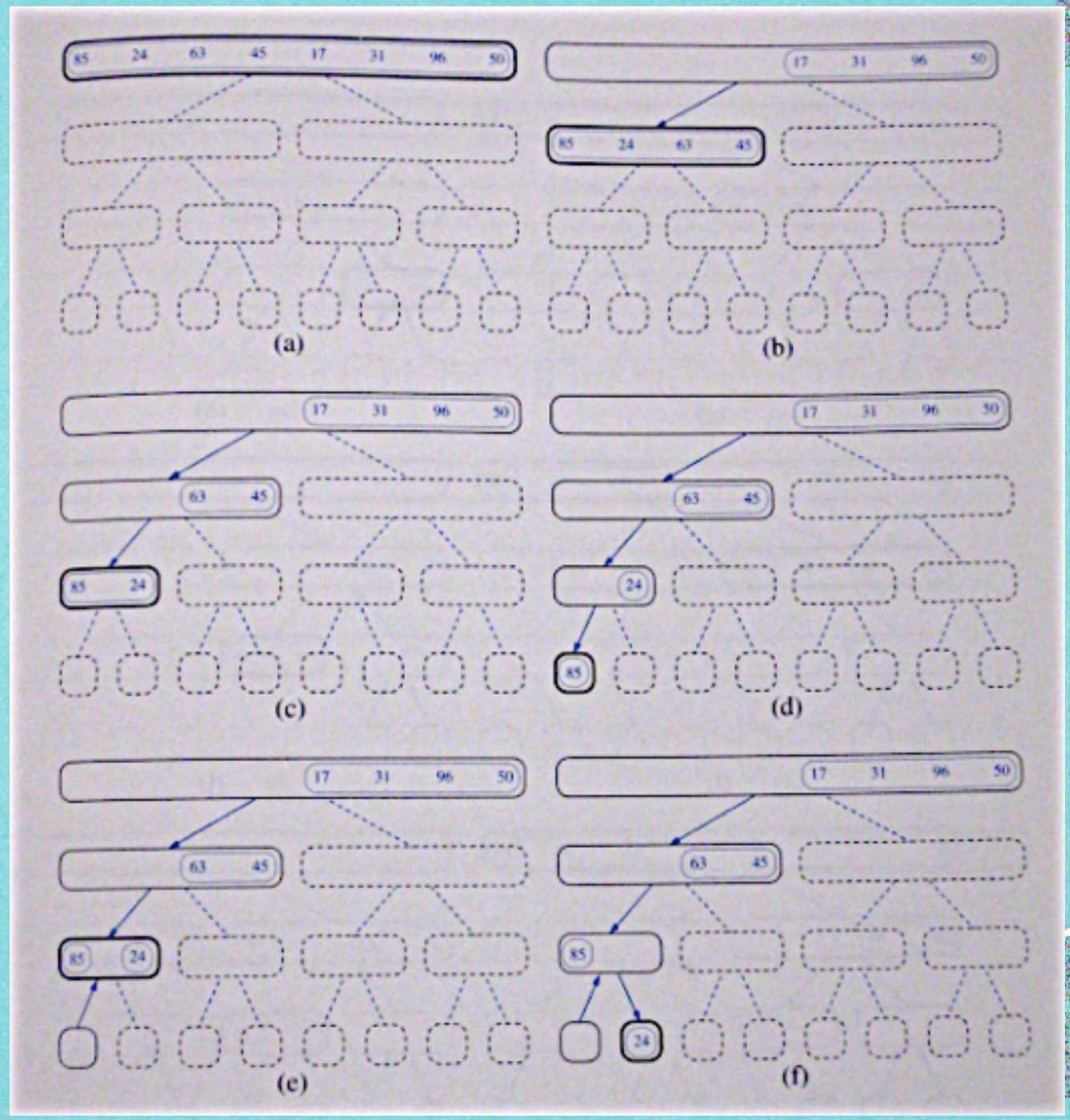
(h)



(i)

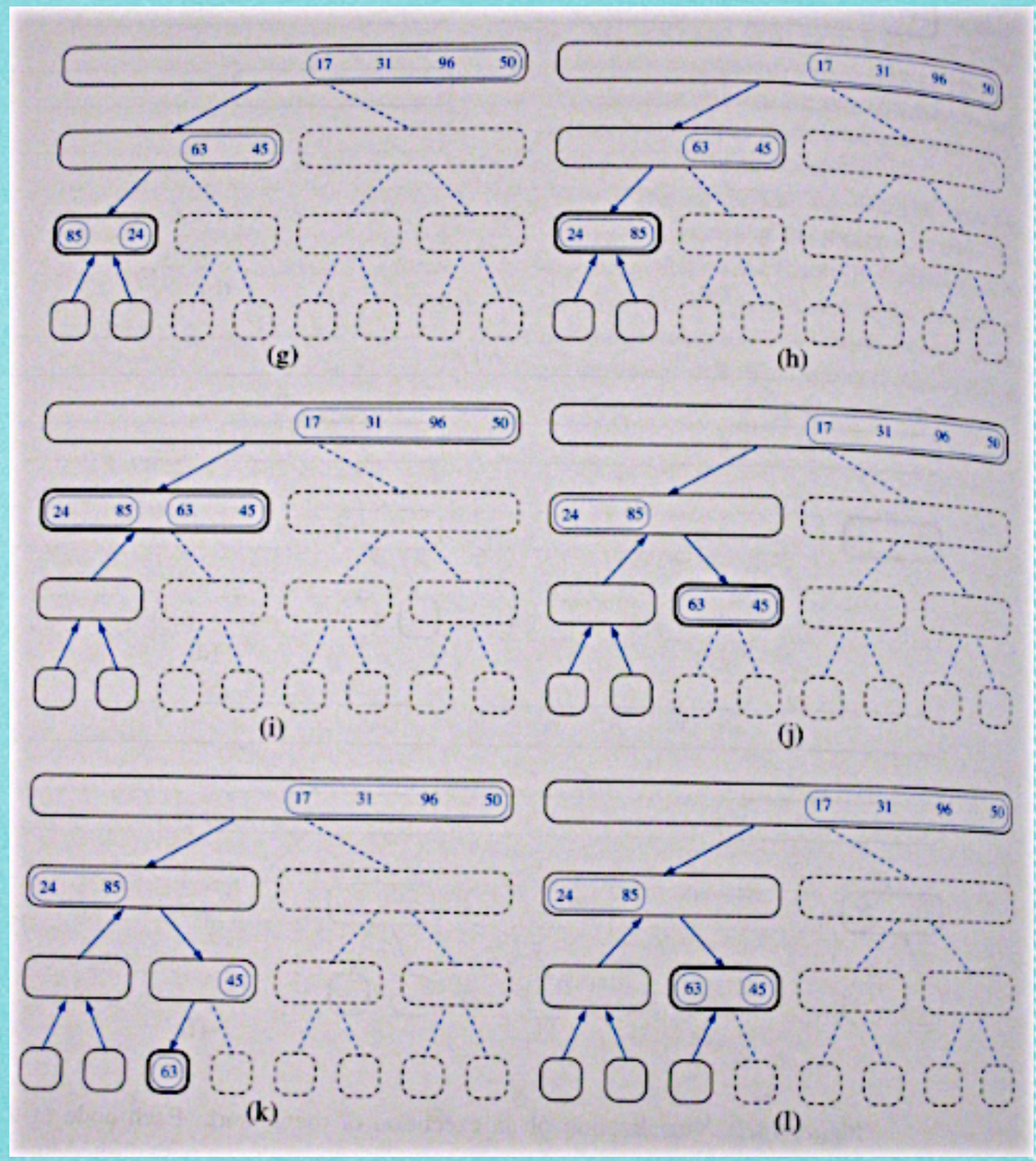
Ablauf makroskopisch

```
1  if p < r
2      then q ← ⌊(p + r)/2⌋
3           MERGE-SORT(A, p, q)
4           MERGE-SORT(A, q + 1, r)
5           MERGE(A, p, q, r)
```



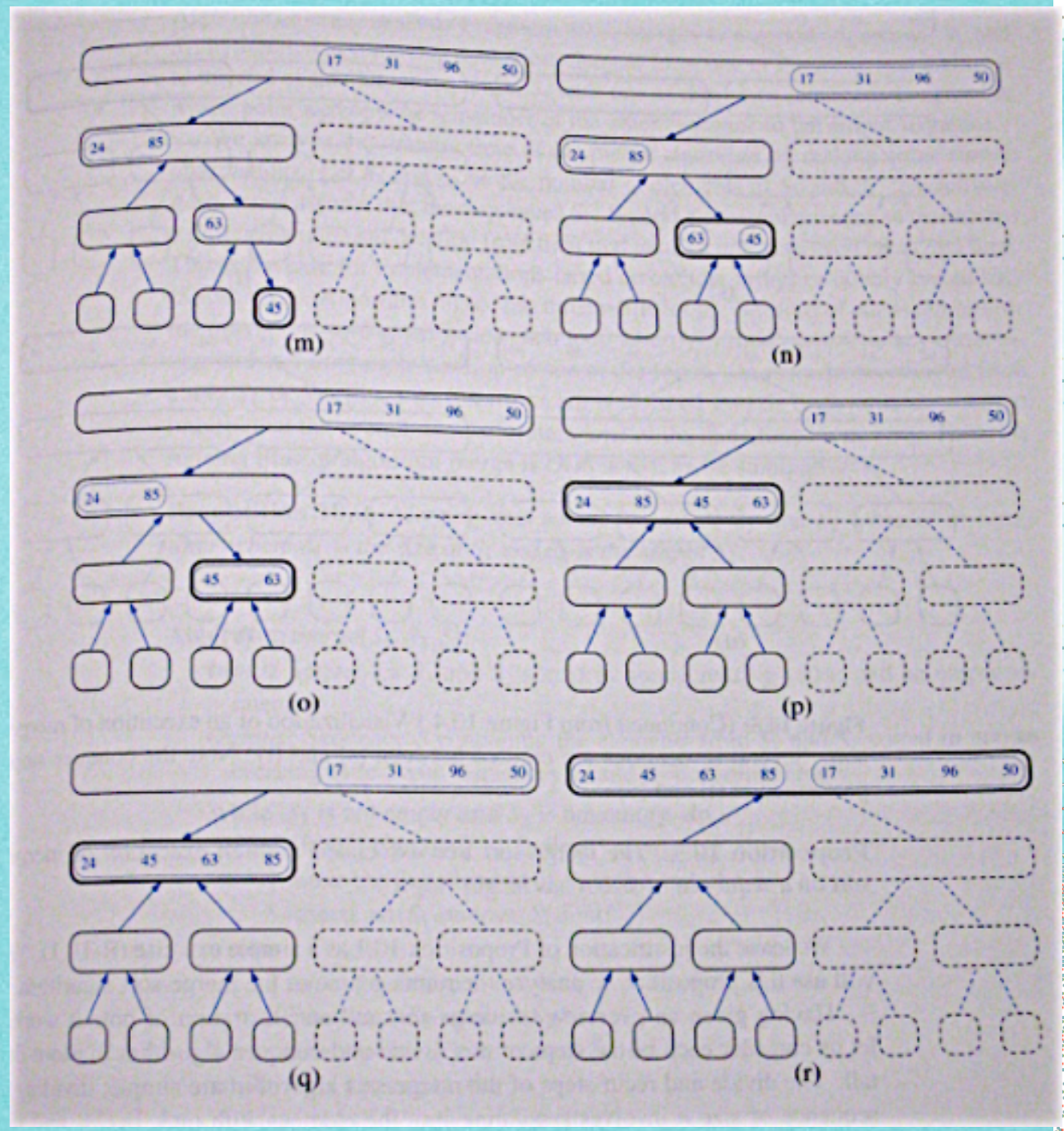
Ablauf makroskopisch

```
1  if p < r
2      then q ← ⌊(p + r)/2⌋
3           MERGE-SORT(A, p, q)
4           MERGE-SORT(A, q + 1, r)
5           MERGE(A, p, q, r)
```



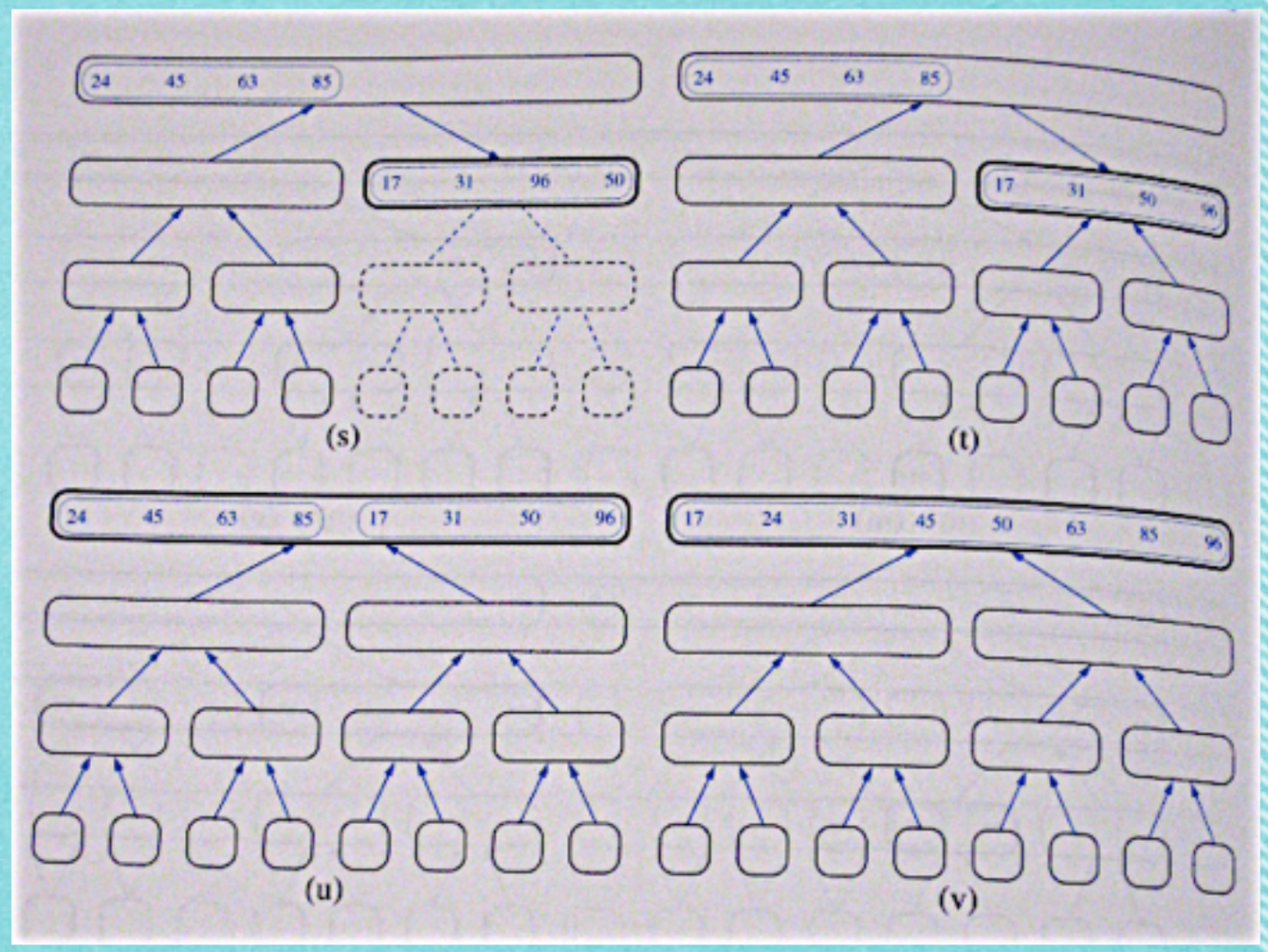
Ablauf makroskopisch

```
1  if p < r
2      then q ← ⌊(p + r)/2⌋
3           MERGE-SORT(A, p, q)
4           MERGE-SORT(A, q + 1, r)
5           MERGE(A, p, q, r)
```



Ablauf makroskopisch

```
1  if p < r
2      then q ← ⌊(p + r)/2⌋
3           MERGE-SORT(A, p, q)
4           MERGE-SORT(A, q + 1, r)
5           MERGE(A, p, q, r)
```



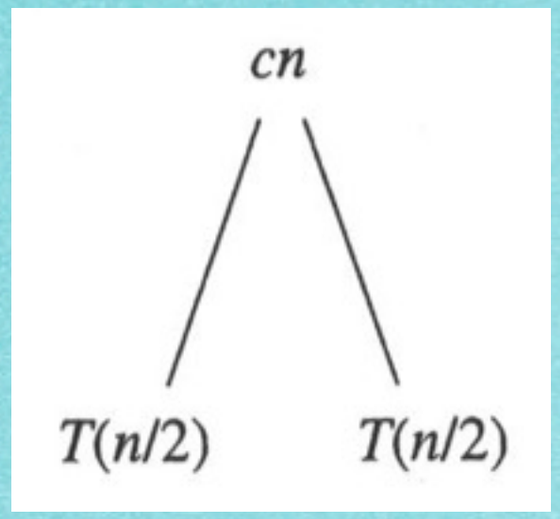
5.2.3 Laufzeit von Mergesort

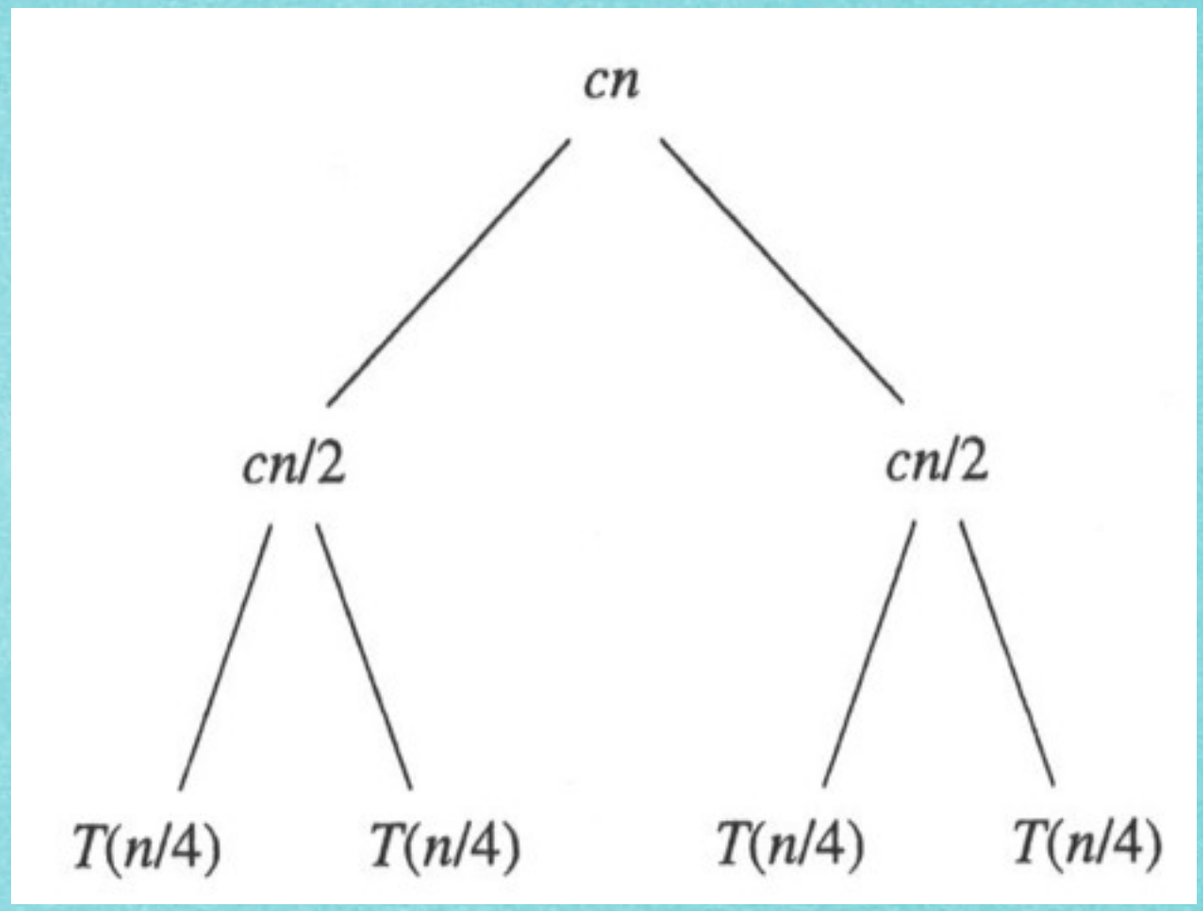
Wie viele Schritte benötigt Merge-Sort für einen Array der Länge n?

$T(n)$

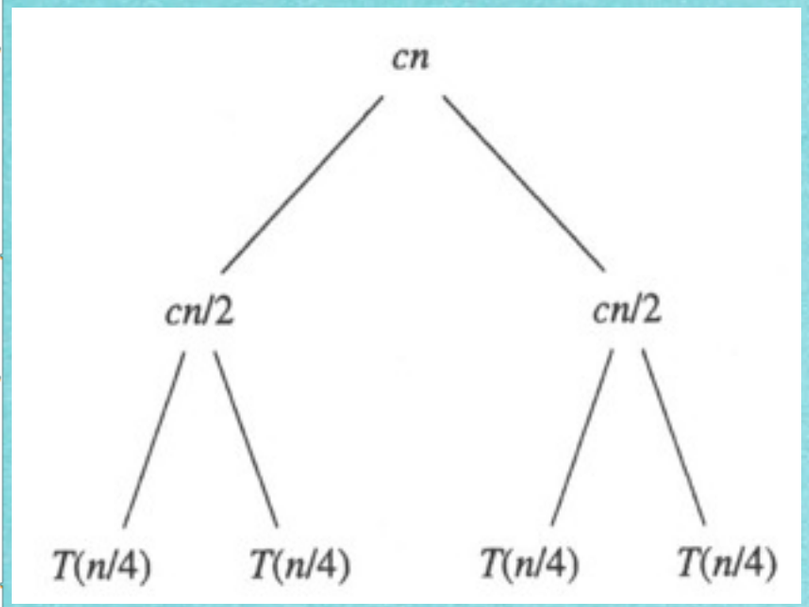
5.1.3 Laufzeit von Mergesort

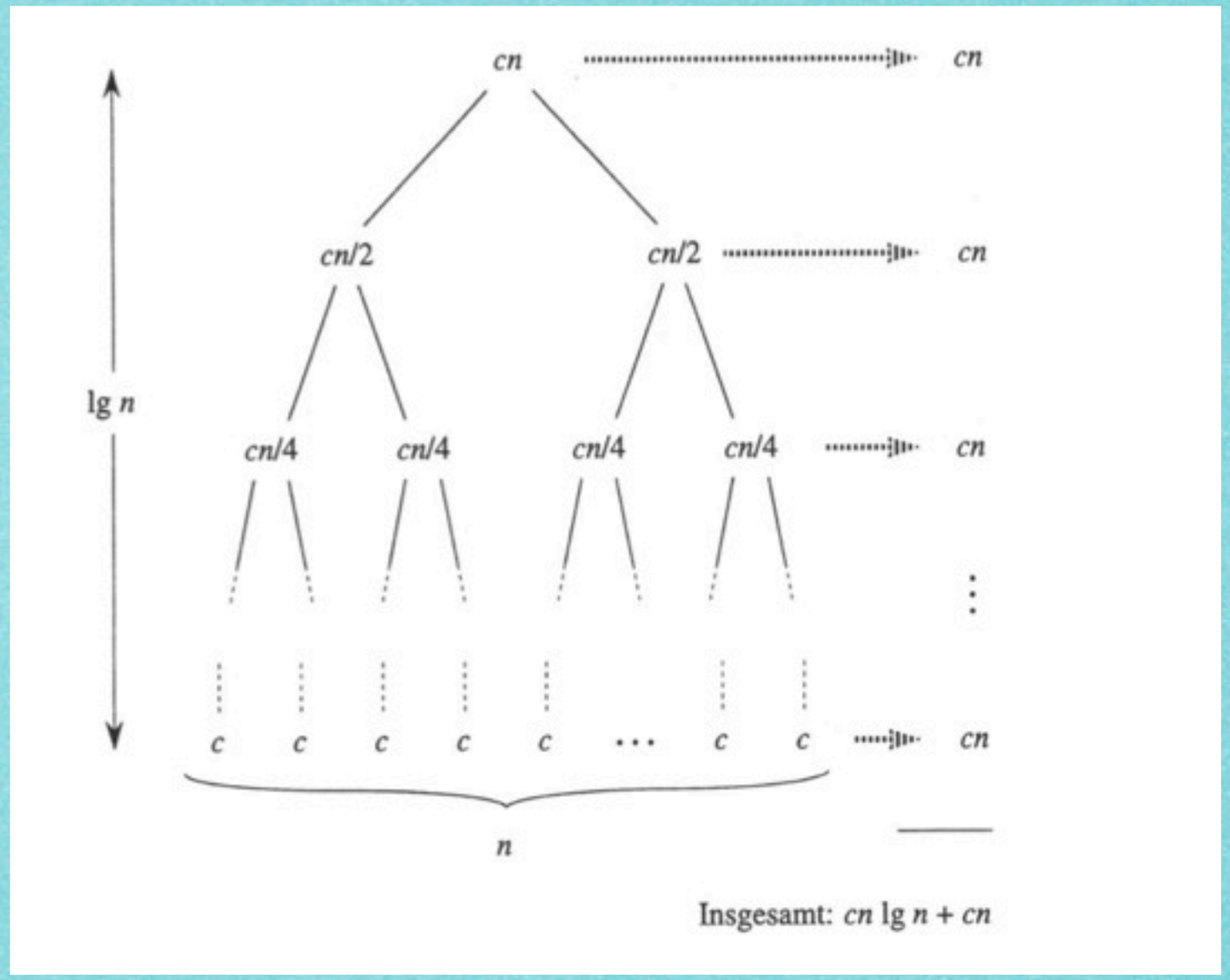
$T(n)$





5.2.3 Laufzeit von Mergesort





5.2.3 Laufzeit von Mergesort

<u>Satz 5.3 (Komplexität von Mergesort)</u>
Für einen n-elementigen Array A hat Mergesort eine Laufzeit von O(n log n).

Mehr an der Tafel!

s.fekete@tu-bs.de