ALGORITHMISCHE METHODE TEILE & HERRSCHE BEISPIEL: MERGESORT

Algorithmische Methode: Teile & Herrsche

Teile & Herrsche (Divide & Conquer)

- Teile Eingabe in mehrere Teile auf
- Löse das Problem rekursiv auf den Teilen
- Füge die Teillösungen zu einer Gesamtlösung zusammen

Teile & Herrsche: Beispiel Sortieren

Teile & Herrsche (Divide & Conquer)

- Teile Eingabe in mehrere Teile auf
- Löse das Problem rekursiv auf den Teilen
- Füge die Teillösungen zu einer Gesamtlösung zusammen

Beispiel(Sortieren)

15	**7**	**6**	**13**	**25**	**4**	**9**	**12**

Teile & Herrsche: Beispiel Sortieren

Teile & Herrsche (Divide & Conquer)

- Teile Eingabe in mehrere Teile auf
- Löse das Problem rekursiv auf den Teilen
- Füge die Teillösungen zu einer Gesamtlösung zusammen

Beispiel(Sortieren)

Schritt 1:
Aufteilen der
Eingabe

Teile & Herrsche: Beispiel Sortieren

Teile & Herrsche (Divide & Conquer)

- Teile Eingabe in mehrere Teile auf
- Löse das Problem rekursiv auf den Teilen
- Füge die Teillösungen zu einer Gesamtlösung zusammen

Beispiel(Sortieren)

Schritt 2:
Rekursiv Sortieren

Teile & Herrsche: Beispiel Sortieren

Teile & Herrsche (Divide & Conquer)

- Teile Eingabe in mehrere Teile auf
- Löse das Problem rekursiv auf den Teilen
- Füge die Teillösungen zu einer Gesamtlösung zusammen

Beispiel(Sortieren)

Schritt 3:
Zusammenfügen

Teile & Herrsche: Beispiel Sortieren

Teile & Herrsche (Divide & Conquer)

- Teile Eingabe in mehrere Teile auf
- Löse das Problem rekursiv auf den Teilen
- Füge die Teillösungen zu einer Gesamtlösung zusammen

Beispiel(Sortieren)

Schritt 3:
Zusammenfügen

Teile & Herrsche: Beispiel Sortieren

Teile & Herrsche (Divide & Conquer)

- Teile Eingabe in mehrere Teile auf
- Löse das Problem rekursiv auf den Teilen
- Füge die Teillösungen zu einer Gesamtlösung zusammen

Beispiel(Sortieren)

Schritt 3:
Zusammenfügen

Teile & Herrsche: Beispiel Sortieren

Teile & Herrsche (Divide & Conquer)

- Teile Eingabe in mehrere Teile auf
- Löse das Problem rekursiv auf den Teilen
- Füge die Teillösungen zu einer Gesamtlösung zusammen

Beispiel(Sortieren)

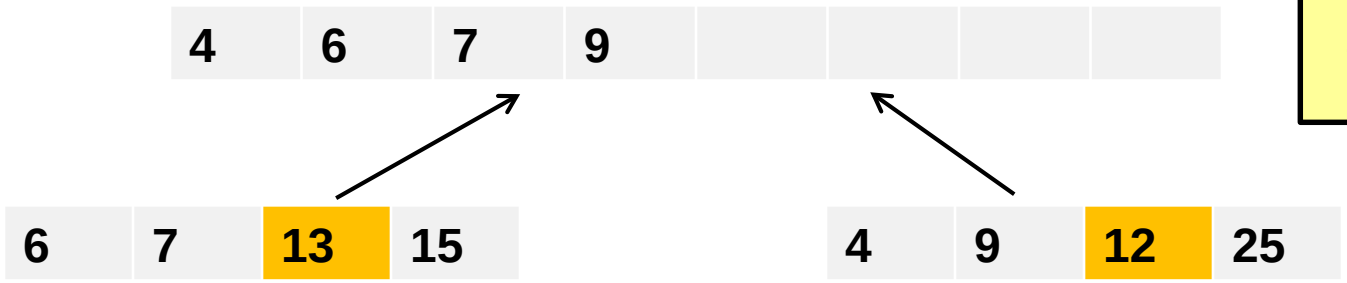
Schritt 3:
Zusammenfügen

Teile & Herrsche: Beispiel Sortieren

Teile & Herrsche (Divide & Conquer)

- Teile Eingabe in mehrere Teile auf
- Löse das Problem rekursiv auf den Teilen
- Füge die Teillösungen zu einer Gesamtlösung zusammen

Beispiel(Sortieren)



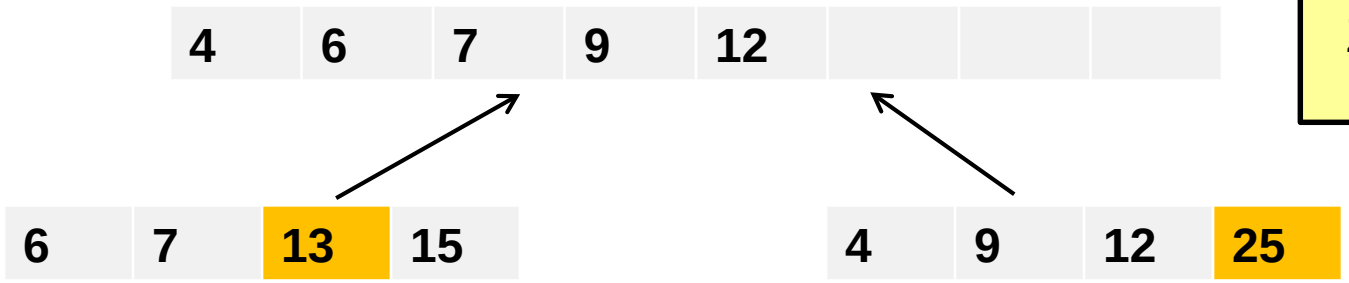
Schritt 3:
Zusammenfügen

Teile & Herrsche: Beispiel Sortieren

Teile & Herrsche (Divide & Conquer)

- Teile Eingabe in mehrere Teile auf
- Löse das Problem rekursiv auf den Teilen
- Füge die Teillösungen zu einer Gesamtlösung zusammen

Beispiel(Sortieren)



Schritt 3:
Zusammenfügen

Teile & Herrsche: Beispiel Sortieren

Teile & Herrsche (Divide & Conquer)

- Teile Eingabe in mehrere Teile auf
- Löse das Problem rekursiv auf den Teilen
- Füge die Teillösungen zu einer Gesamtlösung zusammen

Beispiel(Sortieren)

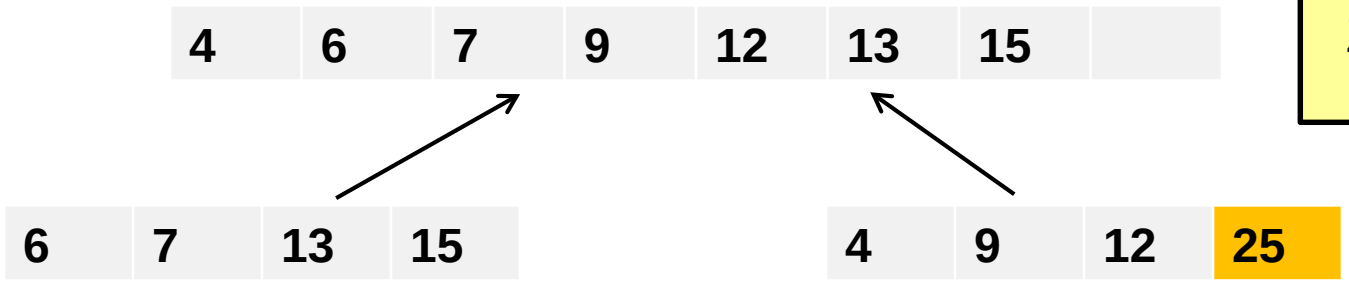
Schritt 3:
Zusammenfügen

Teile & Herrsche: Beispiel Sortieren

Teile & Herrsche (Divide & Conquer)

- Teile Eingabe in mehrere Teile auf
- Löse das Problem rekursiv auf den Teilen
- Füge die Teillösungen zu einer Gesamtlösung zusammen

Beispiel(Sortieren)



Schritt 3:
Zusammenfügen

Teile & Herrsche: Beispiel Sortieren

Teile & Herrsche (Divide & Conquer)

- Teile Eingabe in mehrere Teile auf
- Löse das Problem rekursiv auf den Teilen
- Füge die Teillösungen zu einer Gesamtlösung zusammen

Beispiel(Sortieren)

Schritt 3:
Zusammenfügen

Teile & Herrsche: Beispiel Sortieren

Teile & Herrsche (Divide & Conquer)

- Teile Eingabe in mehrere Teile auf
- Löse das Problem rekursiv auf den Teilen
- Füge die Teillösungen zu einer Gesamtlösung zusammen

- *Wichtig*
- Wir benötigen Rekursionabbruch
- Sortieren: Folgen der Länge 1 sind sortiert

MERGESORT

Teile & Herrsche: MergeSort

```
MergeSort(Array A, p, r)
1.    if p< r then
2.       q ← ⌊(p+r)/2⌋
3.       MergeSort(A,p,q)
4.       MergeSort(A,q+1,r)
5.       Merge(A,p,q,r)
```

Teile & Herrsche: MergeSort

MergeSort(Array A, p, r)

1. **if** p< r **then**
2. q ← ⌊(p+r)/2⌋
3. MergeSort(A,p,q)
4. MergeSort(A,q+1,r)
5. Merge(A,p,q,r)

- Sortiere A[p,…,r]

Teile & Herrsche: MergeSort

MergeSort(Array A, p, r)

1. **if** p< r **then**
2. q ← ⌊(p+r)/2⌋
3. MergeSort(A,p,q)
4. MergeSort(A,q+1,r)
5. Merge(A,p,q,r)

- Sortiere A[p,…,r]

Teile & Herrsche: MergeSort

MergeSort(Array A, p, r)

1. **if** p< r **then**
2. q ← ⌊(p+r)/2⌋
3. MergeSort(A,p,q)
4. MergeSort(A,q+1,r)
5. Merge(A,p,q,r)

- Sortiere A[p,…,r]
- p≥r, dann nichts zu tun

Teile & Herrsche: MergeSort

MergeSort(Array A, p, r)

1. **if** p< r **then**
2. $q \leftarrow \lfloor (p+r)/2 \rfloor$
3. MergeSort(A,p,q)
4. MergeSort(A,q+1,r)
5. Merge(A,p,q,r)

- Sortiere A[p,…,r]
- p≥r, dann nichts zu tun

Teile & Herrsche: MergeSort

MergeSort(Array A, p, r)

1. **if** p< r **then**
2. $q \leftarrow \lfloor (p+r)/2 \rfloor$
3. MergeSort(A,p,q)
4. MergeSort(A,q+1,r)
5. Merge(A,p,q,r)

- Sortiere A[p,…,r]
- p≥r, dann nichts zu tun
- Berechne Mitte

Teile & Herrsche: MergeSort

```
MergeSort(Array A, p, r)
1.    if p< r then
2.       q ← ⌊(p+r)/2⌋
3.       MergeSort(A,p,q)
4.       MergeSort(A,q+1,r)
5.       Merge(A,p,q,r)
```

- Sortiere A[p,…,r]
- p≥r, dann nichts zu tun
- Berechne Mitte

Teile & Herrsche: MergeSort

MergeSort(Array A, p, r)

1. **if** p< r **then**
2. q ← ⌊(p+r)/2⌋
3. MergeSort(A,p,q)
4. MergeSort(A,q+1,r)
5. Merge(A,p,q,r)

- Sortiere A[p,…,r]
- p≥r, dann nichts zu tun
- Berechne Mitte
- Sortiere linke Hälfte

Teile & Herrsche: MergeSort

```
MergeSort(Array A, p, r)
1.    if p< r then
2.       q ← ⌊(p+r)/2⌋
3.       MergeSort(A,p,q)
4.       MergeSort(A,q+1,r)
5.       Merge(A,p,q,r)
```

- Sortiere A[p,…,r]
- p≥r, dann nichts zu tun
- Berechne Mitte
- Sortiere linke Hälfte

Teile & Herrsche: MergeSort

MergeSort(Array A, p, r)

1. **if** p< r **then**
2. q ← ⌊(p+r)/2⌋
3. MergeSort(A,p,q)
4. MergeSort(A,q+1,r)
5. Merge(A,p,q,r)

- Sortiere A[p,…,r]
- p≥r, dann nichts zu tun
- Berechne Mitte
- Sortiere linke Hälfte
- Sortiere rechte Hälfte

Teile & Herrsche: MergeSort

```
MergeSort(Array A, p, r)
1.    if p< r then
2.       q ← ⌊(p+r)/2⌋
3.       MergeSort(A,p,q)
4.       MergeSort(A,q+1,r)
5.       Merge(A,p,q,r)
```

- Sortiere A[p,…,r]
- p≥r, dann nichts zu tun
- Berechne Mitte
- Sortiere linke Hälfte
- Sortiere rechte Hälfte

Teile & Herrsche: MergeSort

MergeSort(Array A, p, r)		➢ Sortiere A[p,…,r]
1.	**if** p< r **then**	➢ p≥r, dann nichts zu tun
2.	q ← ⌊(p+r)/2⌋	➢ Berechne Mitte
3.	MergeSort(A,p,q)	➢ Sortiere linke Hälfte
4.	MergeSort(A,q+1,r)	➢ Sortiere rechte Hälfte
5.	Merge(A,p,q,r)	➢ Zusammenfügen

Teile & Herrsche: MergeSort

MergeSort(Array A, p, r)

1. **if** p< r **then**
2. q ← ⌊(p+r)/2⌋
3. MergeSort(A,p,q)
4. MergeSort(A,q+1,r)
5. Merge(A,p,q,r)

- Sortiere A[p,…,r]
- p≥r, dann nichts zu tun
- Berechne Mitte
- Sortiere linke Hälfte
- Sortiere rechte Hälfte
- Zusammenfügen

Teile & Herrsche: MergeSort

MergeSort(Array A, p, r)		➢ Sortiere A[p,…,r]
1.	**if** p< r **then**	➢ p≥r, dann nichts zu tun
2.	q ← ⌊(p+r)/2⌋	➢ Berechne Mitte
3.	MergeSort(A,p,q)	➢ Sortiere linke Hälfte
4.	MergeSort(A,q+1,r)	➢ Sortiere rechte Hälfte
5.	Merge(A,p,q,r)	➢ Zusammenfügen

Aufruf des Algorithmus

- MergeSort(A,1,r) für r=length(A)

Merge

Merge(Array A, p, q, r)

1. **Array B**
2. i ← p, j ← q+1
3. b ←1

- Zusammenfügen von A[p,…,q], A[q+1,…,r]
  - Hilfsarray zum Mergen Länge r-p+1
  - Hilfsvariablen für linke / rechte Hälfte von A
  - Hilfsvariablen für Array B

Merge

```
Merge(Array A, p, q, r)
1.    Array B
2.    i ← p, j ← q+1
3.    b ←1
4.    while i <= q and j <= r do
5.        if A[i] <= A[j] then
6.            B[b] ← A[i]
7.            b ← b + 1
8.            i ← i + 1
```

- Zusammenfügen von A[p,…,q], A[q+1,…,r]
- Hilfsarray zum Mergen Länge r-p+1
- Hilfsvariablen für linke / rechte Hälfte von A
- Hilfsvariablen für Array B
- Solange Einträge auf beiden Seiten
- links kleiner
- Zuweisung nach B
- Hilfsvariablen erhöhen
- Hilfsvariablen erhöhen

Merge

```
Merge(Array A, p, q, r)
1.    Array B
2.    i ← p, j ← q+1
3.    b ←1
4.    while i <= q and j <= r do
5.        if A[i] <= A[j] then
6.            B[b] ← A[i]
7.            b ← b + 1
8.            i ← i + 1
9.        else
10.           B[b] ← A[j]
11.           b ← b + 1
12.           j ← j + 1
```

- Zusammenfügen von A[p,…,q], A[q+1,…,r]
- Hilfsarray zum Mergen Länge r-p+1
- Hilfsvariablen für linke / rechte Hälfte von A
- Hilfsvariablen für Array B
- Solange Einträge auf beiden Seiten
- links kleiner
- Zuweisung nach B
- Hilfsvariablen erhöhen
- Hilfsvariablen erhöhen
- rechts kleiner
- Zuweisung nach B
- Hilfsvariablen erhöhen
- Hilfsvariablen erhöhen

Merge

```
Merge(Array A, p, q, r)
1.    Array B
2.    i ← p, j ← q+1
3.    b ←1
4.    while i <= q and j <= r do
5.        if A[i] <= A[j] then
6.            B[b++] ← A[i++]
7.        else
8.            B[b++] ← A[j++]
```

- Zusammenfügen von A[p,…,q], A[q+1,…,r]
- Hilfsarray zum Mergen Länge r-p+1
- Hilfsvariablen für linke / rechte Hälfte von A
- Hilfsvariablen für Array B
- Solange Einträge auf beiden Seiten
- links kleiner
- Zuweisung nach B
- rechts kleiner
- Zuweisung nach B

Merge

```
Merge(Array A, p, q, r)
1.    Array B
2.    i ← p, j ← q+1
3.    b ←1
4.    while i <= q and j <= r do
5.        if A[i] <= A[j] then
6.            B[b++] ← A[i++]
7.        else
8.            B[b++] ← A[j++]
9.    while i <= q do
10.       B[b++] ← A[i++]
```

- Zusammenfügen von A[p,…,q], A[q+1,…,r]
- Hilfsarray zum Mergen Länge r-p+1
- Hilfsvariablen für linke / rechte Hälfte von A
- Hilfsvariablen für Array B
- Solange Einträge auf beiden Seiten
  - links kleiner
    - Zuweisung nach B
  - rechts kleiner
    - Zuweisung nach B
- Noch Einträge auf der linken Seite
  - Zuweisung nach B

Merge

```
Merge(Array A, p, q, r)
1.    Array B
2.    i ← p, j ← q+1
3.    b ←1
4.    while i <= q and j <= r do
5.        if A[i] <= A[j] then
6.            B[b++] ← A[i++]
7.        else
8.            B[b++] ← A[j++]
9.    while i <= q do
10.       B[b++] ← A[i++]
11.   while j <= r do
12.       B[b++] ← A[j++]
```

- Zusammenfügen von A[p,…,q], A[q+1,…,r]
- Hilfsarray zum Mergen Länge r-p+1
- Hilfsvariablen für linke / rechte Hälfte von A
- Hilfsvariablen für Array B
- Solange Einträge auf beiden Seiten
  - links kleiner
    - Zuweisung nach B
  - rechts kleiner
    - Zuweisung nach B
- Noch Einträge auf der linken Seite
  - Zuweisung nach B
- Noch Einträge auf der rechten Seite
  - Zuweisung nach B

Merge

```
Merge(Array A, p, q, r)
1.    Array B
2.    i ← p, j ← q+1
3.    b ←1
4.    while i <= q and j <= r do
5.        if A[i] <= A[j] then
6.            B[b++] ← A[i++]
7.        else
8.            B[b++] ← A[j++]
9.    while i <= q do
10.       B[b++] ← A[i++]
11.   while j <= r do
12.       B[b++] ← A[j++]
13.   A[p, …, r] ← B
```

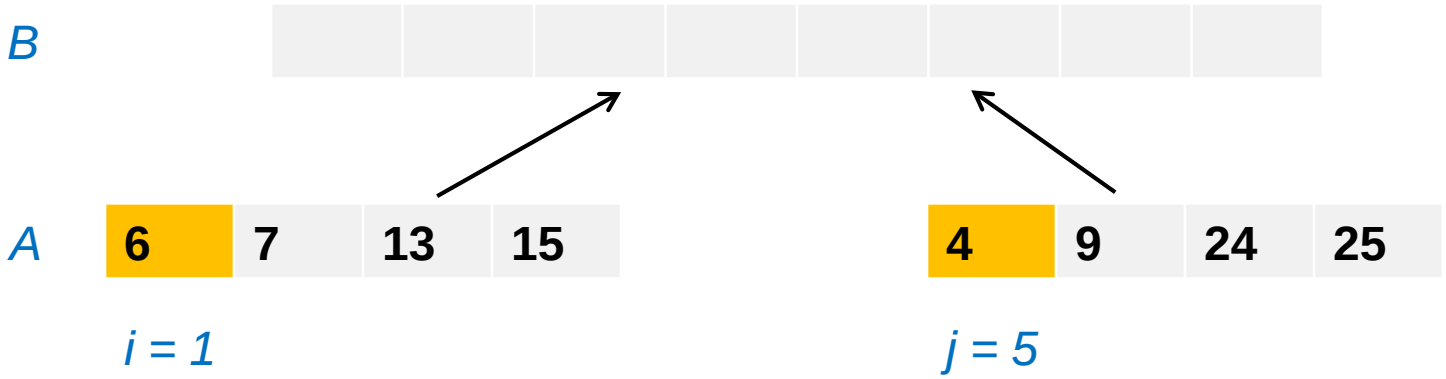
- Zusammenfügen von A[p,…,q], A[q+1,…,r]
- Hilfsarray zum Mergen Länge r-p+1
- Hilfsvariablen für linke / rechte Hälfte von A
- Hilfsvariablen für Array B
- Solange Einträge auf beiden Seiten
  - links kleiner
    - Zuweisung nach B
  - rechts kleiner
    - Zuweisung nach B
- Noch Einträge auf der linken Seite
  - Zuweisung nach B
- Noch Einträge auf der rechten Seite
  - Zuweisung nach B
- Kopiere Hilfsarray B nach A

MergeSort: Merge Schritt

Einträge auf beiden Seiten

merge(A, 1, 4, 8)

b = 1

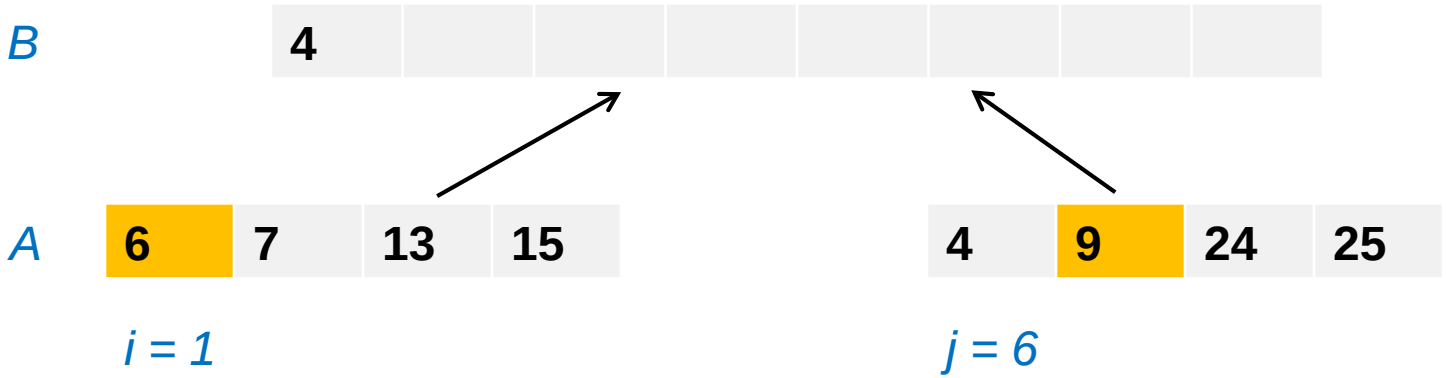


Teile & Herrsche: Beispiel Sortieren

Einträge auf beiden Seiten

merge(A, 1, 4, 8)

b = 2

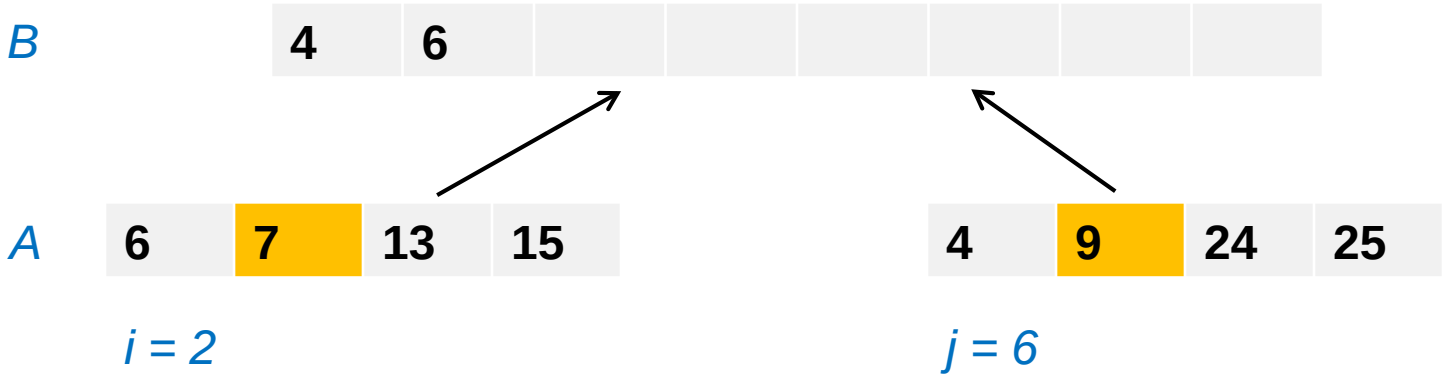


Teile & Herrsche: Beispiel Sortieren

Einträge auf beiden Seiten

merge(A, 1, 4, 8)

b = 3

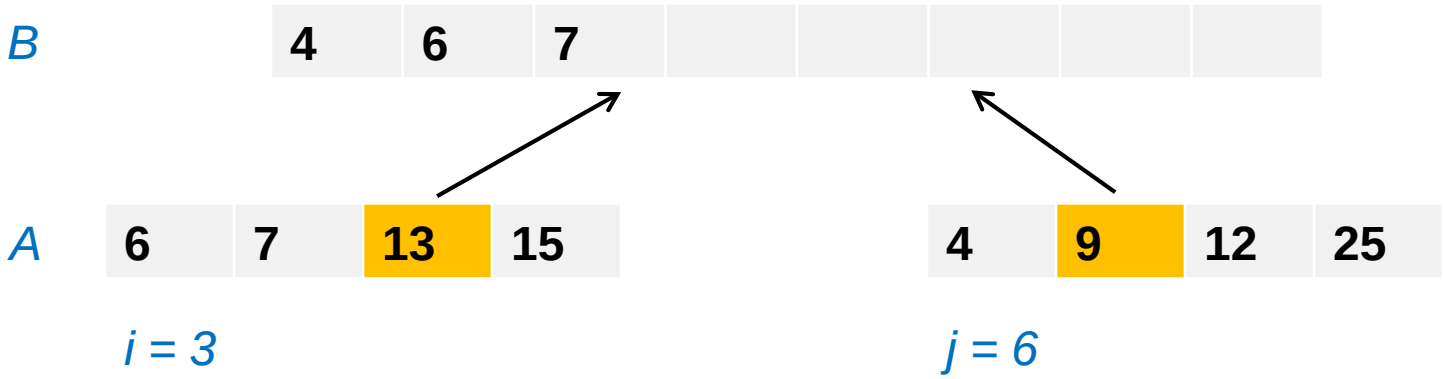


Teile & Herrsche: Beispiel Sortieren

Einträge auf beiden Seiten

merge(A, 1, 4, 8)

b = 4

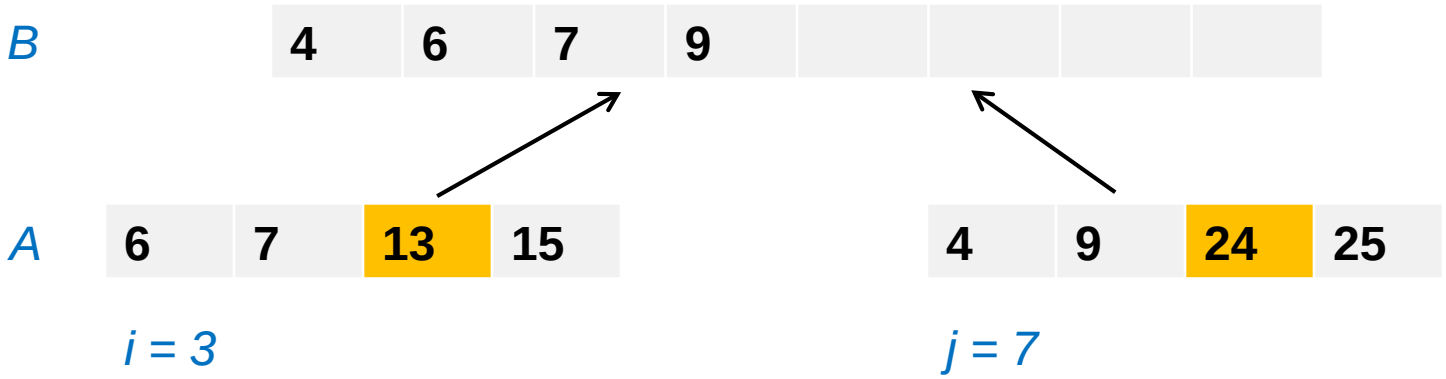


Teile & Herrsche: Beispiel Sortieren

Einträge auf beiden Seiten

merge(A, 1, 4, 8)

b = 5

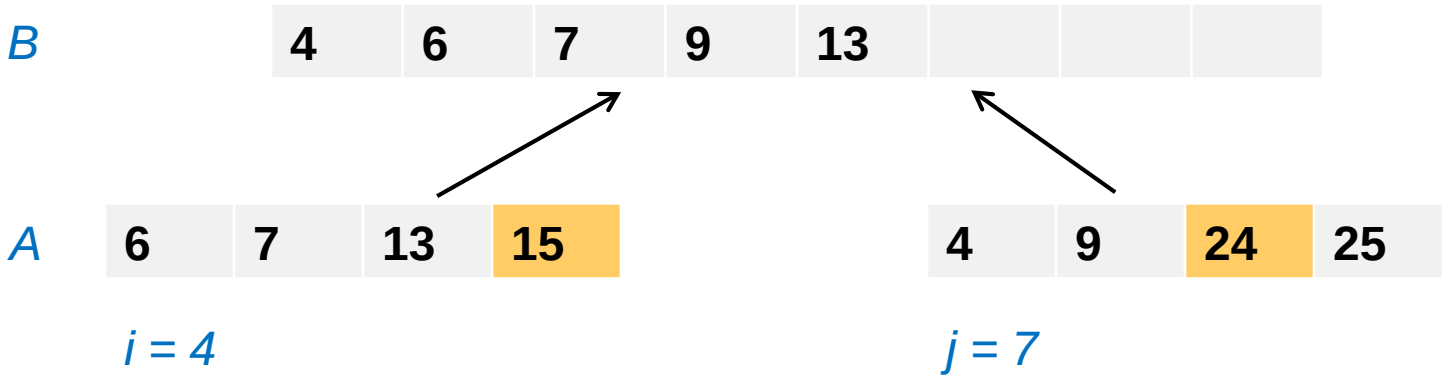


Teile & Herrsche: Beispiel Sortieren

Einträge auf beiden Seiten

merge(A, 1, 4, 8)

b = 6

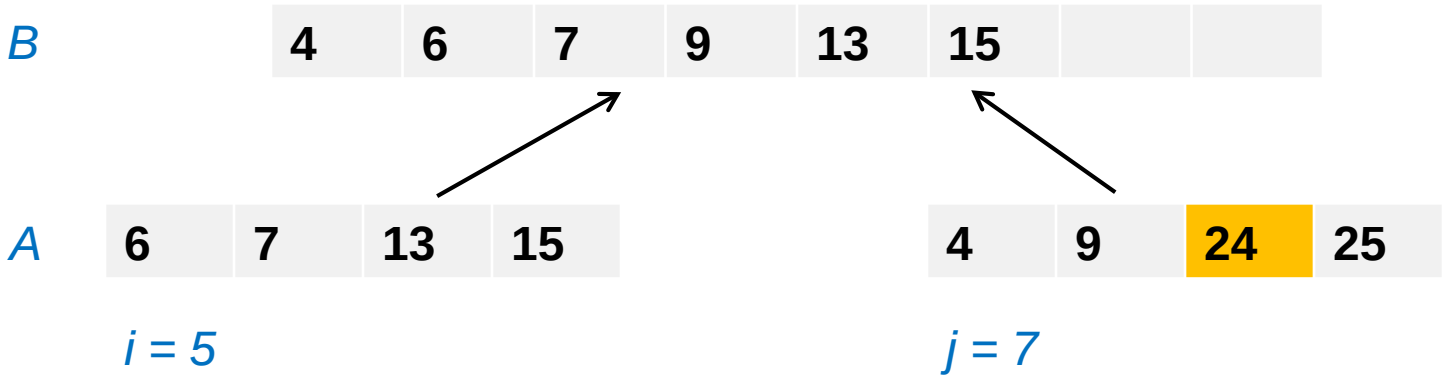


Teile & Herrsche: Beispiel Sortieren

Einträge auf beiden Seiten

merge(A, 1, 4, 8)

b = 7

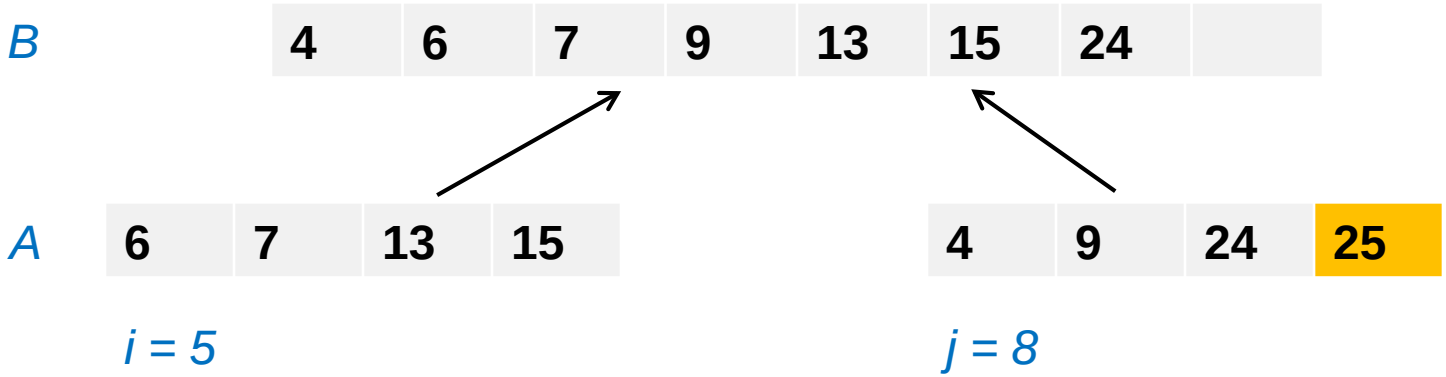


Teile & Herrsche: Beispiel Sortieren

Einträge nur auf rechter Seite

merge(A, 1, 4, 8)

b = 8

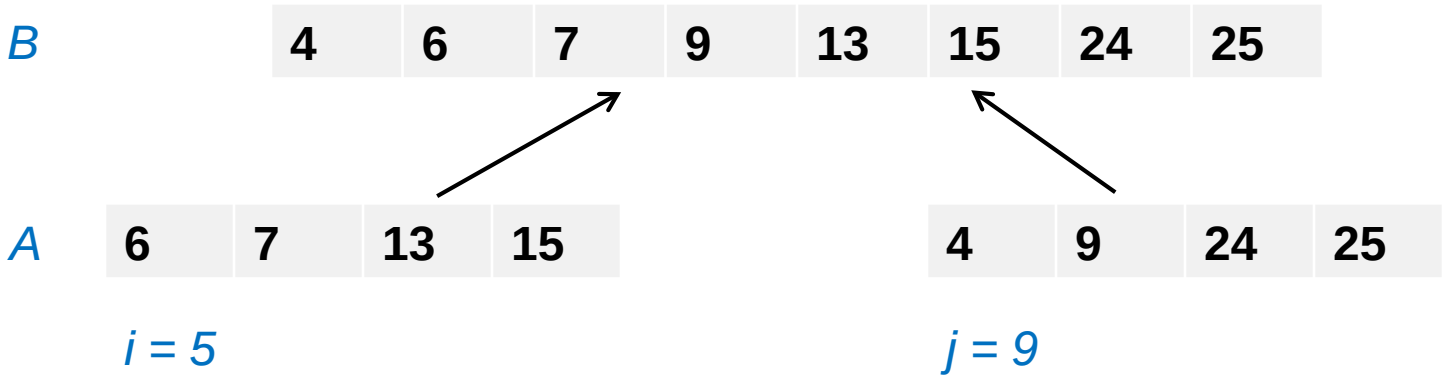


Teile & Herrsche: Beispiel Sortieren

Einträge nur auf rechter Seite

merge(A, 1, 4, 8)

b = 9



Teile & Herrsche: Beispiel Sortieren

Rückkopieren von
Hilfsarray B nach Array A

merge(A, 1, 4, 8)

A	**4**	**6**	**7**	**9**	**13**	**15**	**24**	**25**

MERGESORT: LAUFZEIT

Teile & Herrsche: MergeSort

MergeSort(Array A, p, r) ➢ Sortiere A[p,…,r]

1. **if** p< r **then** ➢ p≥r, dann nichts zu tun
2. q ← ⌊(p+r)/2⌋ ➢ Berechne Mitte
3. MergeSort(A,p,q) ➢ Sortiere linke Hälfte
4. MergeSort(A,q+1,r) ➢ Sortiere rechte Hälfte
5. Merge(A,p,q,r) ➢ Zusammenfügen

Aufruf des Algorithmus

- MergeSort(A,1,n) für Feld A[1...n]
- Laufzeit?

Teile & Herrsche: MergeSort

MergeSort(Array A, p, r)		➢ Sortiere A[p,…,r]
1.	**if** p< r **then**	➢ p≥r, dann nichts zu tun
2.	q ← ⌊(p+r)/2⌋	➢ Berechne Mitte
3.	MergeSort(A,p,q)	➢ Sortiere linke Hälfte
4.	MergeSort(A,q+1,r)	➢ Sortiere rechte Hälfte
5.	Merge(A,p,q,r)	➢ Zusammenfügen

Aufruf des Algorithmus

- MergeSort(A,1,n) für Feld A[1...n]
- T(m) = maximale Laufzeit bei Eingabe A, p, r mit r-p+1=m

Teile & Herrsche: MergeSort

MergeSort(Array A, p, r)	Laufzeit:
1. **if** p< r **then**	1
2. q ← ⌊(p+r)/2⌋	
3. MergeSort(A,p,q)	
4. MergeSort(A,q+1,r)	
5. Merge(A,p,q,r)	

Aufruf des Algorithmus

- MergeSort(A,1,n) für Feld A[1...n]
- T(m) = maximale Laufzeit bei Eingabe A, p, r mit r-p+1=m

Teile & Herrsche: MergeSort

MergeSort(Array A, p, r)	Laufzeit:
1. **if** p< r **then**	1
2. $q \leftarrow \lfloor (p+r)/2 \rfloor$	1
3. MergeSort(A,p,q)	
4. MergeSort(A,q+1,r)	
5. Merge(A,p,q,r)	

Aufruf des Algorithmus

- MergeSort(A,1,n) für Feld A[1...n]
- T(m) = maximale Laufzeit bei Eingabe A, p, r mit r-p+1=m

Teile & Herrsche: MergeSort

MergeSort(Array A, p, r)	Laufzeit:
1. **if** p< r **then**	1
2. q ← ⌊(p+r)/2⌋	1
3. MergeSort(A,p,q)	1+T(n/2)
4. MergeSort(A,q+1,r)	
5. Merge(A,p,q,r)	

Wir nehmen an, dass n eine Zweierpotenz ist, d.h. wir müssen uns nicht um das Runden kümmern.

Aufruf des Algorithmus

- MergeSort(A,1,n) für Feld A[1...n]
- T(m) = maximale Laufzeit bei Eingabe A, p, r mit r-p+1=m

Teile & Herrsche: MergeSort

MergeSort(Array A, p, r)	Laufzeit:
1. **if** p< r **then**	1
2. q ← ⌊(p+r)/2⌋	1
3. MergeSort(A,p,q)	1+T(n/2)
4. MergeSort(A,q+1,r)	1+T(n/2)
5. Merge(A,p,q,r)	

Wir nehmen an, dass n eine Zweierpotenz ist, d.h. wir müssen uns nicht um das Runden kümmern.

Aufruf des Algorithmus

- MergeSort(A,1,n) für Feld A[1...n]
- T(m) = maximale Laufzeit bei Eingabe A, p, r mit r-p+1=m

Teile & Herrsche: MergeSort

MergeSort(Array A, p, r)	Laufzeit:
1. **if** p< r **then**	1
2. q ← ⌊(p+r)/2⌋	1
3. MergeSort(A,p,q)	1+T(n/2)
4. MergeSort(A,q+1,r)	1+T(n/2)
5. Merge(A,p,q,r)	≤ c‘n

c‘ ist genügend große Konstante

Aufruf des Algorithmus

- MergeSort(A,1,n) für Feld A[1...n]
- T(m) = maximale Laufzeit bei Eingabe A, p, r mit r-p+1=m

Teile & Herrsche: MergeSort

MergeSort(Array A, p, r)	Laufzeit:
1. **if** p< r **then**	1
2. q ← ⌊(p+r)/2⌋	1
3. MergeSort(A,p,q)	1+T(n/2)
4. MergeSort(A,q+1,r)	1+T(n/2)
5. Merge(A,p,q,r)	≤ c‘n
	≤ 2T(n/2) + cn

c ≥ c‘+4

Aufruf des Algorithmus

- MergeSort(A,1,n) für Feld A[1...n]
- T(m) = maximale Laufzeit bei Eingabe A, p, r mit r-p+1=m

Teile & Herrsche: MergeSort

Laufzeit als Rekursion

- $T(n) \leq \begin{cases} C & \text{, falls } n=1 \\ 2\,T(n/2) + cn & \text{, falls } n>1 \end{cases}$

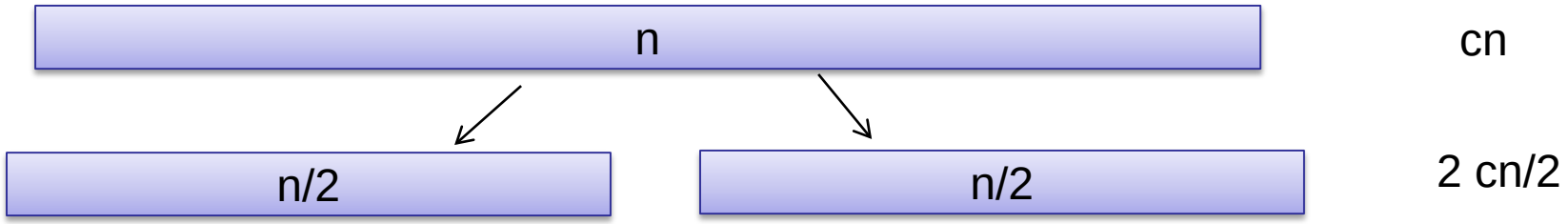
- Wobei c,C geeignete Konstanten sind.

Teile & Herrsche: MergeSort

Auflösen von $T(n) \leq 2\,T(n/2) + cn$ (Intuition)

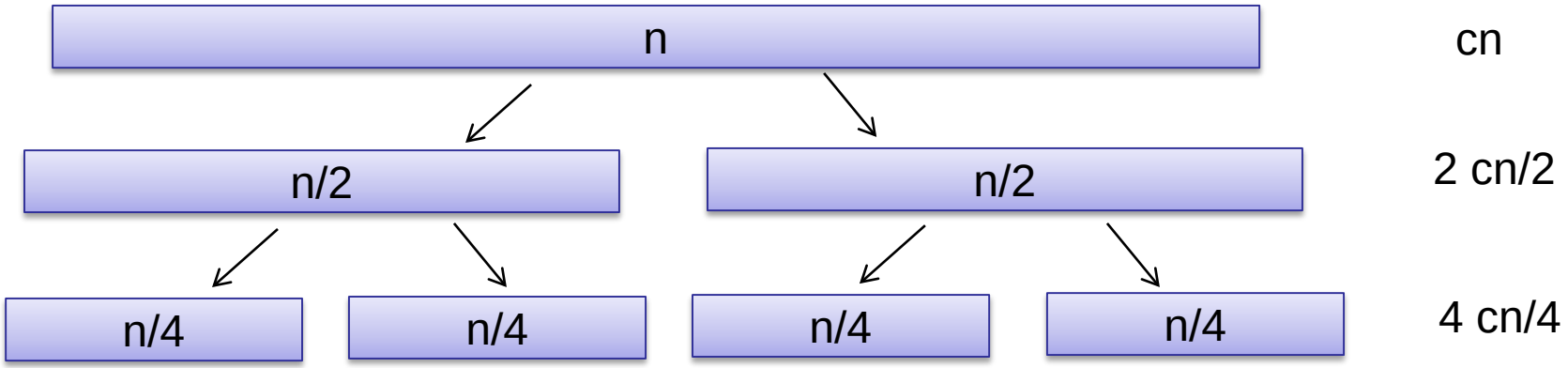
Teile & Herrsche: MergeSort

Auflösen von $T(n) \leq 2\,T(n/2) + cn$ (Intuition)



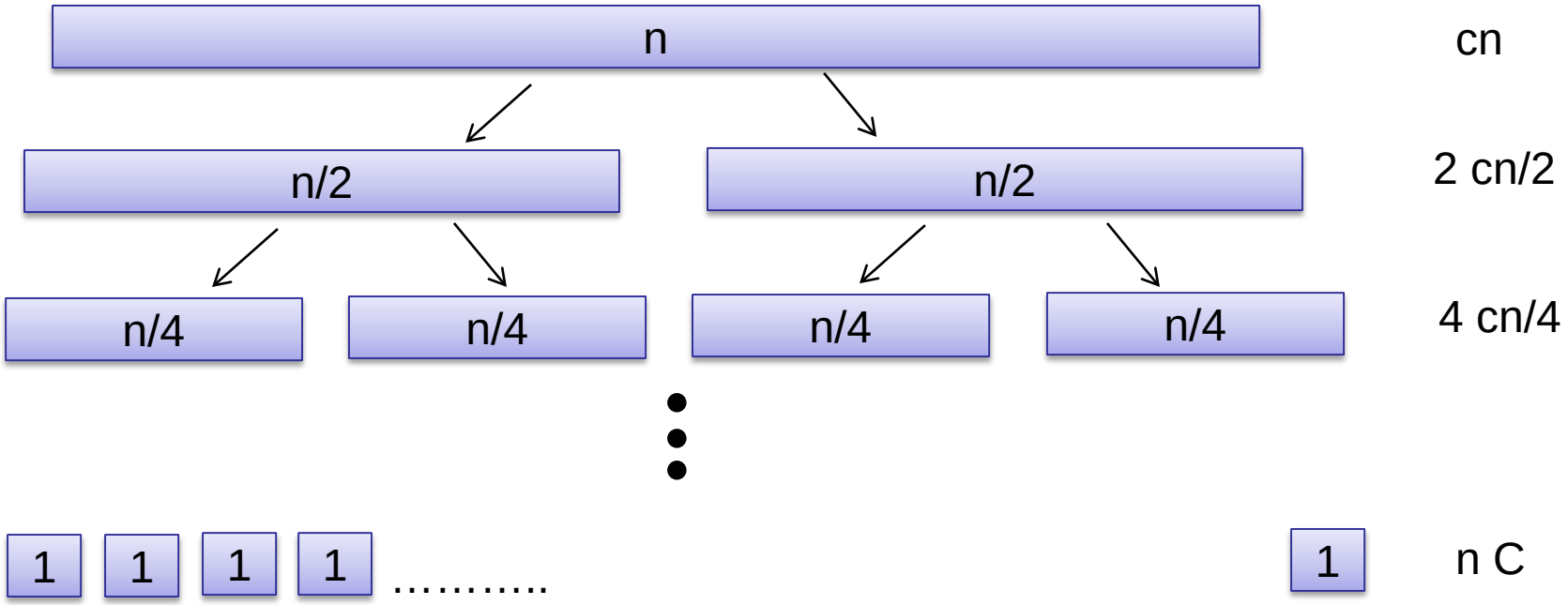
Teile & Herrsche: MergeSort

Auflösen von $T(n) \leq 2\,T(n/2) + cn$ (Intuition)



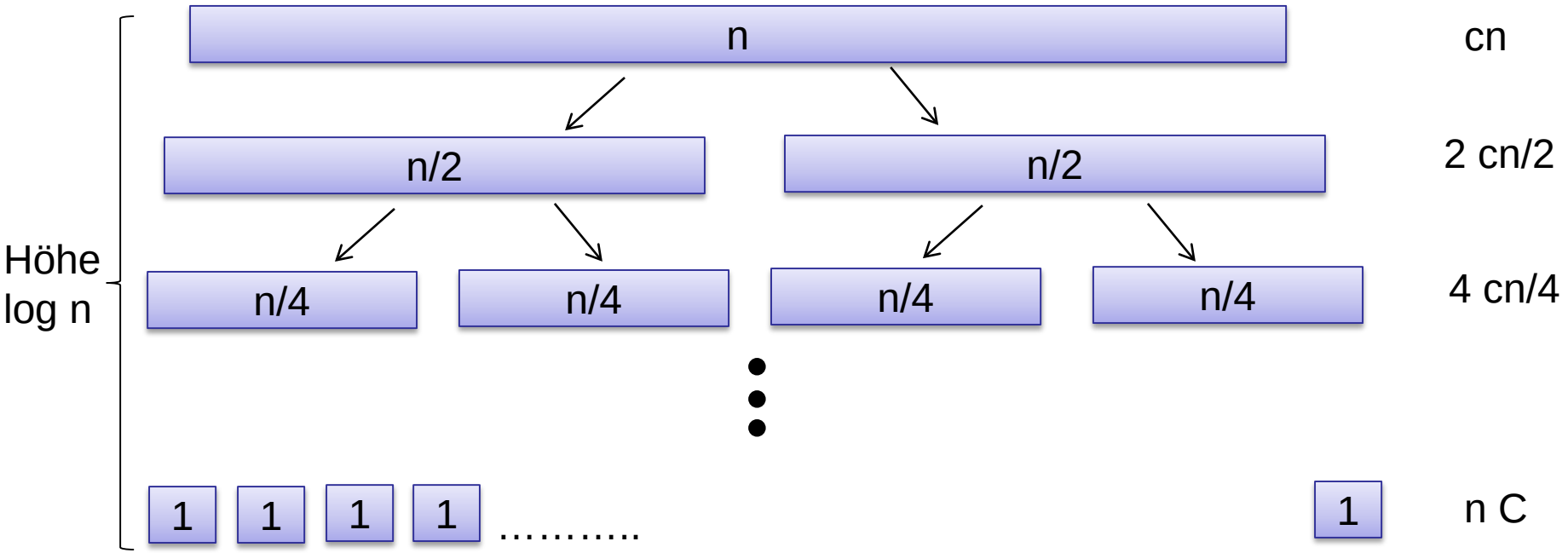
Teile & Herrsche: MergeSort

Auflösen von $T(n) \leq 2\,T(n/2) + cn$ (Intuition)



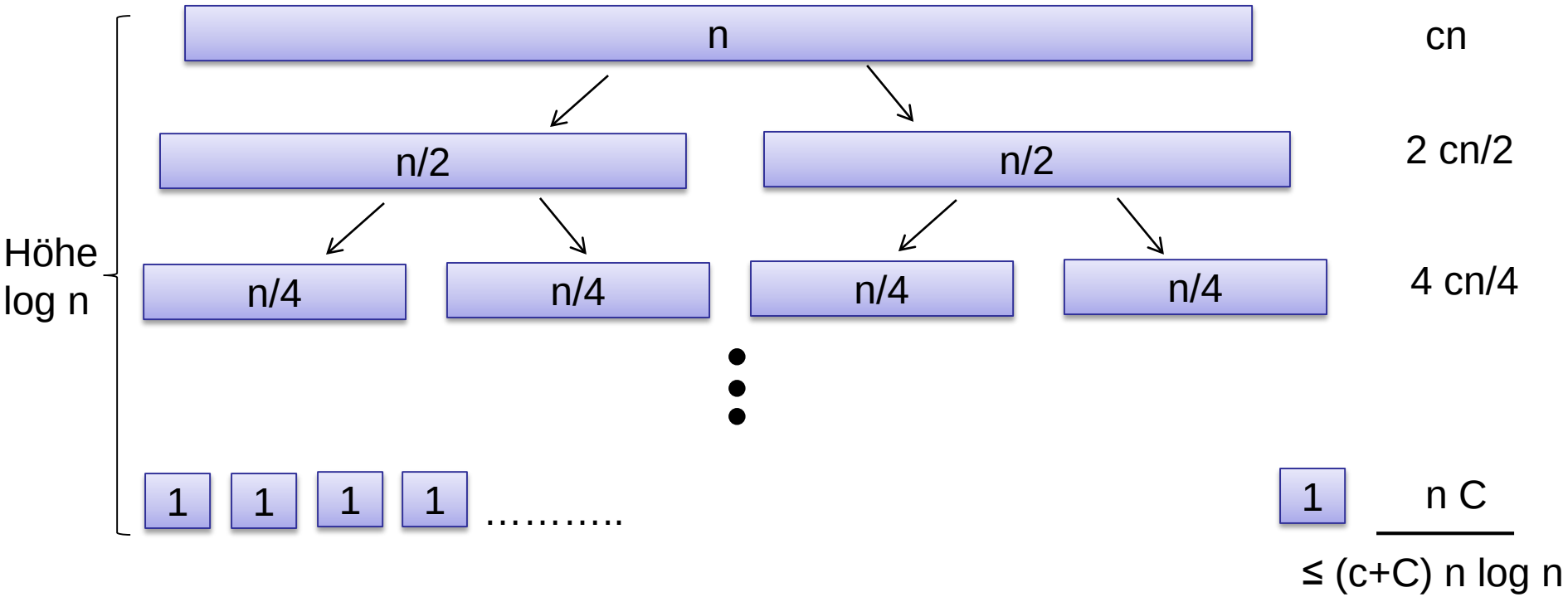
Teile & Herrsche: MergeSort

Auflösen von $T(n) \leq 2\,T(n/2) + cn$ (Intuition)



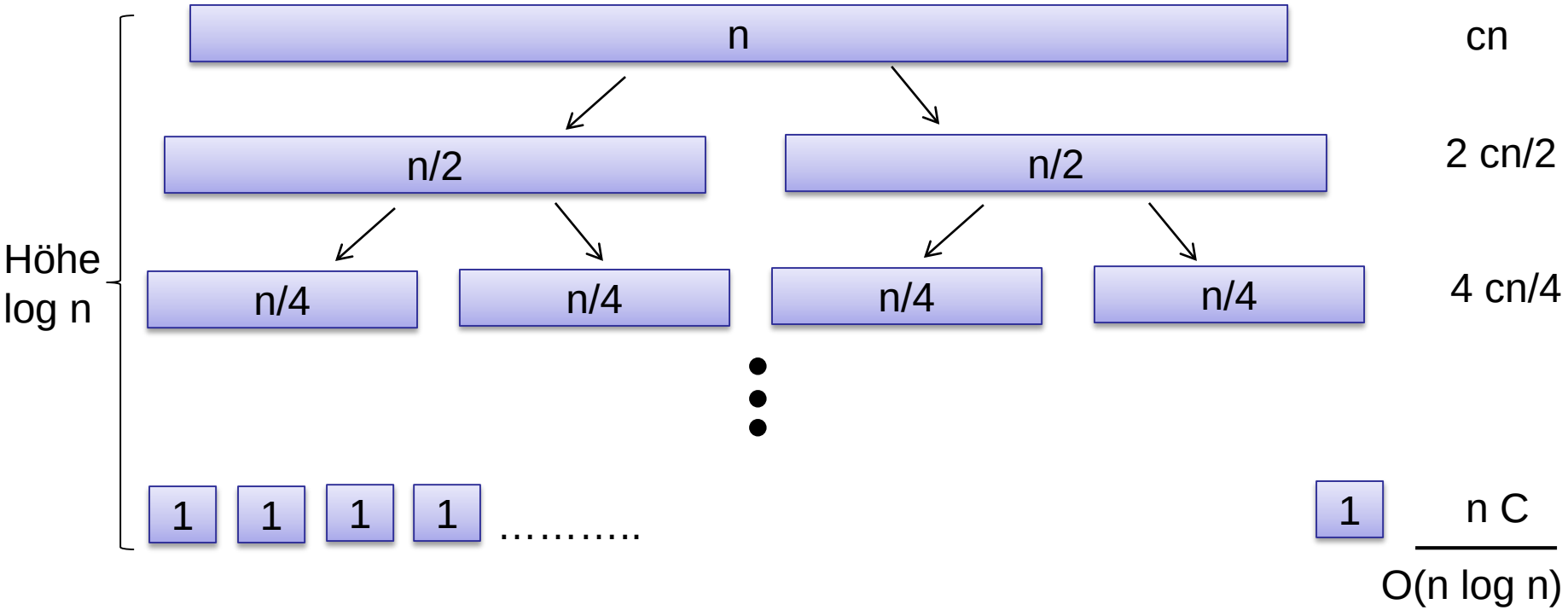
Teile & Herrsche: MergeSort

Auflösen von $T(n) \leq 2\,T(n/2) + cn$ (Intuition)



Teile & Herrsche: MergeSort

Auflösen von $T(n) \leq 2\,T(n/2) + cn$ (Intuition)



MergeSort: Laufzeit

Satz

- Algorithmus MergeSort hat eine Laufzeit von O(n log n).

Beweis

MergeSort: Laufzeit

Satz

- Algorithmus MergeSort hat eine Laufzeit von O(n log n).

Beweis

- Die Laufzeit für T(1) und T(2) ist konstant.

MergeSort: Laufzeit

Satz

- Algorithmus MergeSort hat eine Laufzeit von O(n log n).

Beweis

- Die Laufzeit für T(1) und T(2) ist konstant.
- Sei also T(2) ≤ C‘ und C* ≥ max{c,C‘}. Wir zeigen per Induktion, T(n) ≤ C*n log n für alle n≥2

MergeSort: Laufzeit

Satz

- Algorithmus MergeSort hat eine Laufzeit von O(n log n).

Beweis

- Die Laufzeit für T(1) und T(2) ist konstant.
- Sei also T(2) ≤ C‘ und C* ≥ max{c,C‘}. Wir zeigen per Induktion, T(n) ≤ C*n log n für alle n≥2
- (I.A.) für n=2 gilt T(2) ≤ C‘ ≤ C* 2 log 2.

MergeSort: Laufzeit

Satz

- Algorithmus MergeSort hat eine Laufzeit von O(n log n).

Beweis

- Die Laufzeit für T(1) und T(2) ist konstant.
- Sei also T(2) ≤ C‘ und C* ≥ max{c,C‘}. Wir zeigen per Induktion, T(n) ≤ C*n log n für alle n≥2
- (I.A.) für n=2 gilt T(2) ≤ C‘ ≤ C* 2 log 2.
- (I.V.) Für Eingabelänge m<n ist die Laufzeit T(m) ≤ C* m log m.

MergeSort: Laufzeit

Satz

- Algorithmus MergeSort hat eine Laufzeit von O(n log n).

Beweis

- Die Laufzeit für T(1) und T(2) ist konstant.
- Sei also T(2) ≤ C‘ und C* ≥ max{c,C‘}. Wir zeigen per Induktion, T(n) ≤ C*n log n für alle n≥2
- (I.A.) für n=2 gilt T(2) ≤ C‘ ≤ C* 2 log 2.
- (I.V.) Für Eingabelänge m<n ist die Laufzeit T(m) ≤ C* m log m.
- (I.S.) Es gilt T(n) ≤ 2 T(n/2) + cn.

MergeSort: Laufzeit

Satz

- Algorithmus MergeSort hat eine Laufzeit von O(n log n).

Beweis

- Die Laufzeit für T(1) und T(2) ist konstant.
- Sei also T(2) ≤ C‘ und C* ≥ max{c,C‘}. Wir zeigen per Induktion, T(n) ≤ C*n log n für alle n≥2
- (I.A.) für n=2 gilt T(2) ≤ C‘ ≤ C* 2 log 2.
- (I.V.) Für Eingabelänge m<n ist die Laufzeit T(m) ≤ C* m log m.
- (I.S.) Es gilt T(n) ≤ 2 T(n/2) + cn. Nach (I.V.) gilt
  T(n) ≤ 2 C* n/2 log(n/2) + cn

MergeSort: Laufzeit

Satz

- Algorithmus MergeSort hat eine Laufzeit von O(n log n).

Beweis

- Die Laufzeit für T(1) und T(2) ist konstant.
- Sei also T(2) ≤ C‘ und C* ≥ max{c,C‘}. Wir zeigen per Induktion, T(n) ≤ C*n log n für alle n≥2
- (I.A.) für n=2 gilt T(2) ≤ C‘ ≤ C* 2 log 2.
- (I.V.) Für Eingabelänge m<n ist die Laufzeit T(m) ≤ C* m log m.
- (I.S.) Es gilt T(n) ≤ 2 T(n/2) + cn. Nach (I.V.) gilt
  T(n) ≤ 2 C* n/2 log(n/2) + cn
  ≤ C* n (log(n)-1) + cn

MergeSort: Laufzeit

Satz

- Algorithmus MergeSort hat eine Laufzeit von O(n log n).

Beweis

- Die Laufzeit für T(1) und T(2) ist konstant.
- Sei also T(2) ≤ C‘ und C* ≥ max{c,C‘}. Wir zeigen per Induktion, T(n) ≤ C*n log n für alle n≥2
- (I.A.) für n=2 gilt T(2) ≤ C‘ ≤ C* 2 log 2.
- (I.V.) Für Eingabelänge m<n ist die Laufzeit T(m) ≤ C* m log m.
- (I.S.) Es gilt T(n) ≤ 2 T(n/2) + cn. Nach (I.V.) gilt

  $T(n) \leq 2\, C^* \, n/2 \log(n/2) + cn$

  $\leq C^* \, n \, (\log(n)-1) + cn$

  $\leq C^* \, n \, (\log(n)-1) + C^* n \leq C^* \, n \log(n)$

MergeSort: Laufzeit

Satz

- Algorithmus MergeSort hat eine Laufzeit von O(n log n).

Beweis

- Die Laufzeit für T(1) und T(2) ist konstant.
- Sei also T(2) ≤ C‘ und C* ≥ max{c,C‘}. Wir zeigen per Induktion, T(n) ≤ C*n log n für alle n≥2
- (I.A.) für n=2 gilt T(2) ≤ C‘ ≤ C* 2 log 2.
- (I.V.) Für Eingabelänge m<n ist die Laufzeit T(m) ≤ C* m log m.
- (I.S.) Es gilt T(n) ≤ 2 T(n/2) + cn. Nach (I.V.) gilt
  $T(n) \leq 2\, C^* \, n/2 \log(n/2) + cn$
  $\leq C^* \, n \, (\log(n)-1) + cn$
  $\leq C^* \, n \, (\log(n)-1) + C^*n = C^* \, n \log(n)$
- Also gilt T(n) = O(n log n), [da für $n \geq n_0=2$, $T(n) \leq C^*n \log n$ ist]

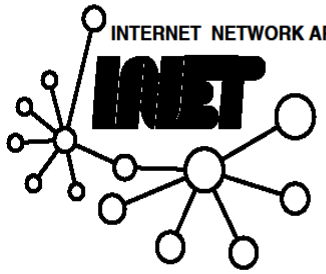
MergeSort: Laufzeit

Satz

- Algorithmus MergeSort hat eine Laufzeit von O(n log n).

Beweis

- Die Laufzeit für T(1) und T(2) ist konstant.
- Sei also T(2) ≤ C‘ und C* ≥ max{c,C‘}. Wir zeigen per Induktion, T(n) ≤ C*n log n für alle n≥2
- (I.A.) für n=2 gilt T(2) ≤ C‘ ≤ C* 2 log 2.
- (I.V.) Für Eingabelänge m<n ist die Laufzeit T(m) ≤ C* m log m.
- (I.S.) Es gilt T(n) ≤ 2 T(n/2) + cn. Nach (I.V.) gilt
  $T(n) \leq 2\, C^* \, n/2 \log(n/2) + cn$
  $\leq C^* \, n \, (\log(n)-1) + cn$
  $\leq C^* \, n \, (\log(n)-1) + C^* n \leq C^* \, n \log(n)$
- Also gilt T(n) = O(n log n), [da für $n \geq n_0 = 2$, T(n) ≤ C*n log n ist]

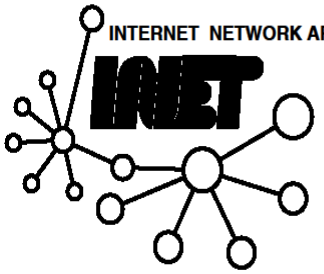


ALGORITHMISCHE METHODE TEILE & HERRSCHE

Teile & Herrsche

Wodurch unterscheiden sich Teile & Herrsche Algorithmen?

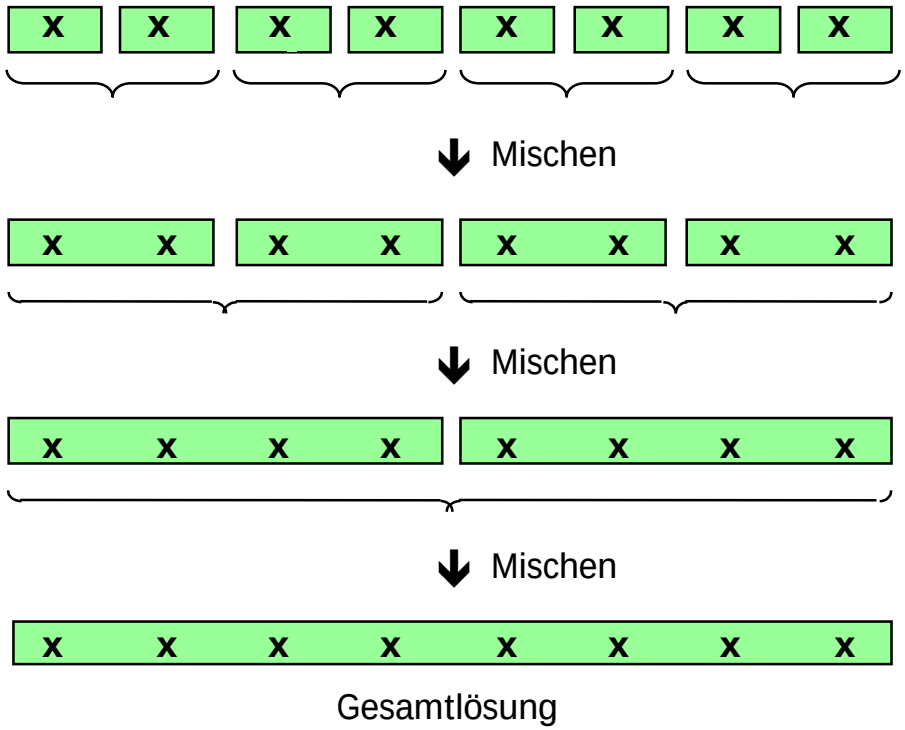
- Die Anzahl der Teilprobleme
- Die Größe der Teilprobleme
- Den Algorithmus für das Zusammensetzen der Teilprobleme
- Den Rekursionsabbruch



ITERATIVE VS. REKURSIV BEISPIEL: MERGESORT

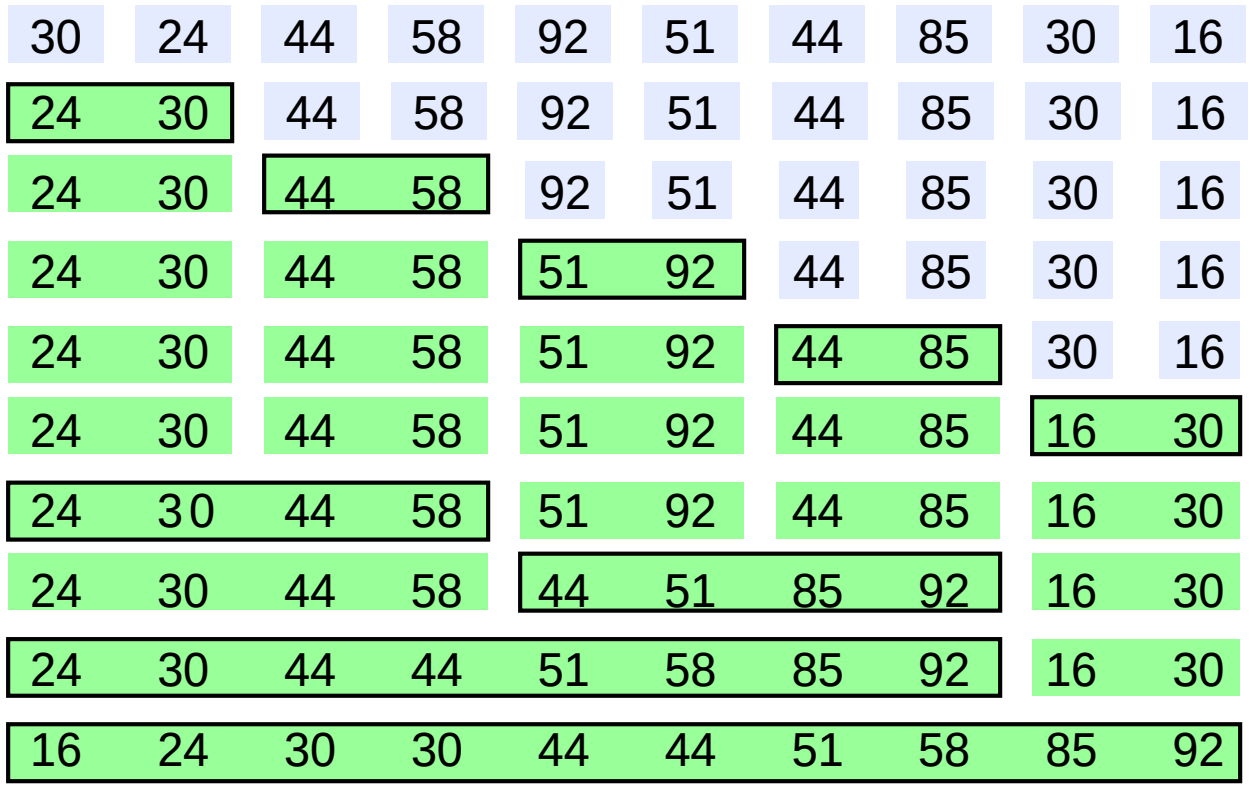
Merge-Sort: Iterative Variante

- Arbeitsweise:



Merge-Sort: Iterative Variante

- Beispiel:



Sortieren durch Mischen (merge sort)

- Laufzeiten (gemessen)

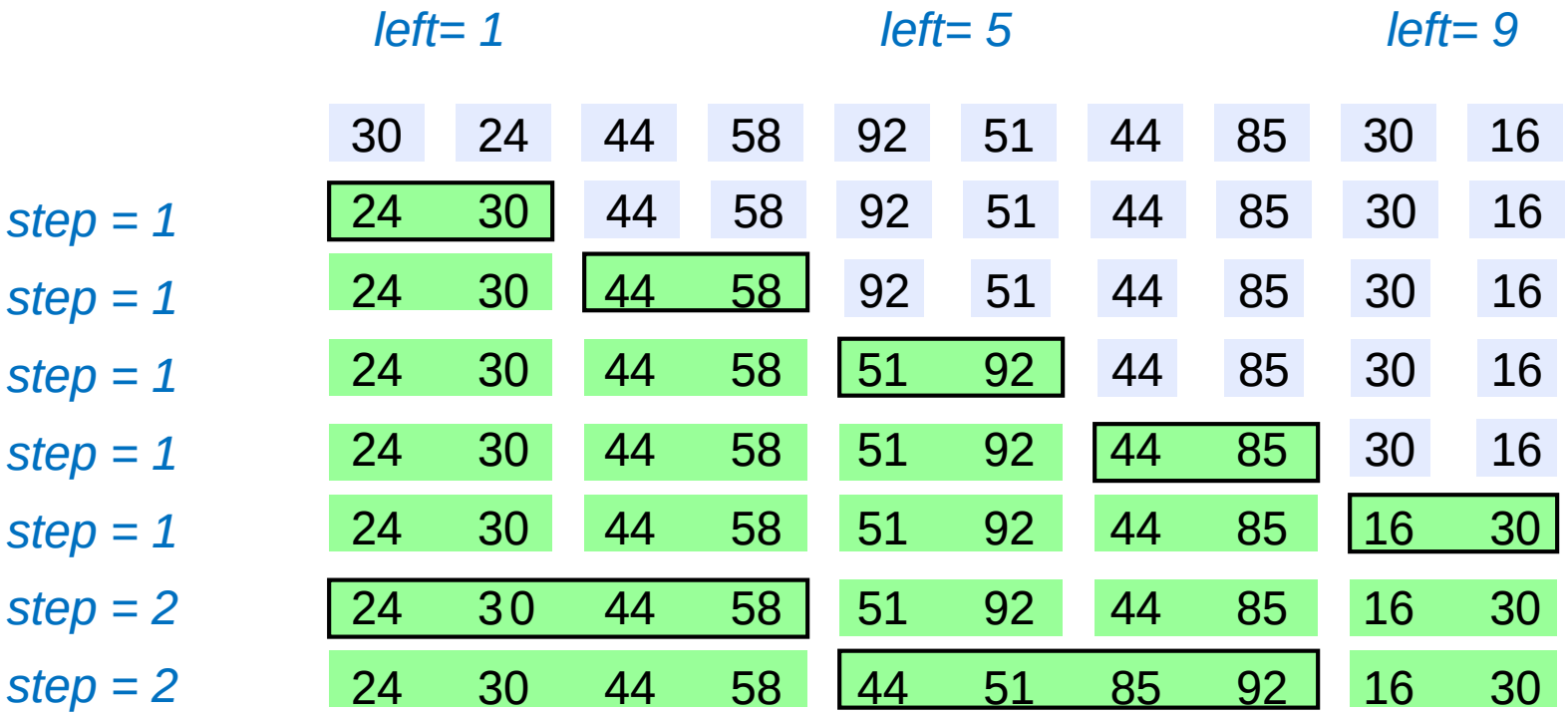
Anzahl	Sortierzeiten [msec]	
	Rekursiv	Iterativ
100	4.50	3.72
200	9.98	8.36
400	21.88	18.68
800	47.92	41.24
1600	103.83	91.16
3200	224.86	199.01

Merge-Sort: Iterative Variante

	left= 1		*left= 3*		*left= 5*		*left= 7*		*left= 9*	
	30	24	44	58	92	51	44	85	30	16
step = 1	**24**	**30**	44	58	92	51	44	85	30	16
step = 1	24	30	**44**	**58**	92	51	44	85	30	16
step = 1	24	30	44	58	**51**	**92**	44	85	30	16
step = 1	24	30	44	58	51	92	**44**	**85**	30	16
step = 1	24	30	44	58	51	92	44	85	**16**	**30**

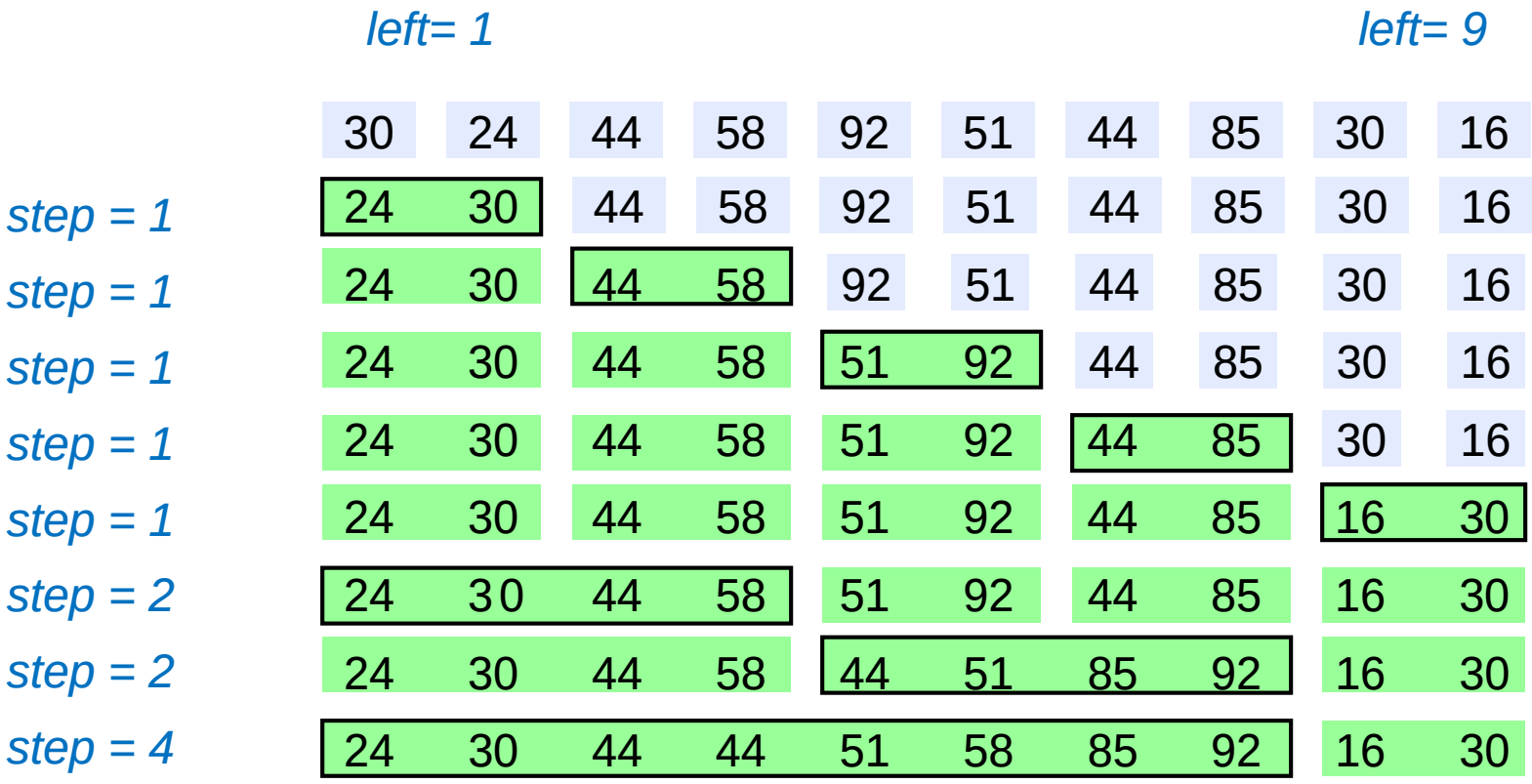
*Merge(A, left, left+step - 1, left + 2*step – 1)*

Merge-Sort: Iterative Variante



*Merge(A, left, left+step - 1, left + 2*step – 1)*

Merge-Sort: Iterative Variante



*Merge(A, left, left+step - 1, left + 2*step – 1)*

Merge-Sort: Iterative Variante

	30	24	44	58	92	51	44	85	30	16
step = 1	24	30	44	58	92	51	44	85	30	16
step = 1	24	30	44	58	92	51	44	85	30	16
step = 1	24	30	44	58	51	92	44	85	30	16
step = 1	24	30	44	58	51	92	44	85	30	16
step = 1	24	30	44	58	51	92	44	85	16	30
step = 2	24	30	44	58	51	92	44	85	16	30
step = 2	24	30	44	58	44	51	85	92	16	30
step = 4	24	30	44	44	51	58	85	92	16	30
step = 8	16	24	30	30	44	44	51	58	85	92

Iterativer MergeSort

IterativMergeSort(Array A)
- ➢ Sortiere A[1,…,n]

1. step ← 1
   - ➢ Hilfsvariablen für Schrittweite
2. **while** step <= n **do**
   - ➢ Solange noch nicht das ganze Array sortiert ist
3.
4.
5.
6.
7.
8.
9.
10. step ← step * 2
    - ➢ Schrittweite erhöhen

Iterativer MergeSort

IterativMergeSort(Array A) ➢ Sortiere A[1,…,n]

1. step ← 1 ➢ Hilfsvariablen für Schrittweite
2. **while** step <= n **do** ➢ Solange noch nicht das ganze Array sortiert ist
3. left ← 1 ➢ Initialisierung der linken Grenze
4. **while** left <= n-step **do** ➢ Schleife zum Sortieren von Teilarrays der Länge Step von left an
5.
6.
7.
8.
9. left ← left + 2*step ➢ Verschieben der linken Grenze
10. step ← step * 2 ➢ Schrittweite erhöhen

Iterativer MergeSort

```
IterativMergeSort(Array A)        ➢ Sortiere A[1,…,n]
1.    step ← 1                    ➢ Hilfsvariablen für Schrittweite
2.    while step <= n do          ➢ Solange noch nicht das ganze Array sortiert ist
3.        left ← 1                ➢ Initialisierung der linken Grenze
4.        while left <= n-step do ➢ Schleife zum Sortieren von Teilarrays
                                      der Länge Step von left an
5.
6.
7.
8.            merge(A, left, middle, right)  ➢ Merge
9.            left ← left + 2*step           ➢ Verschieben der linken Grenze
10.       step ← step * 2                    ➢ Schrittweite erhöhen
```

Iterativer MergeSort

```
IterativMergeSort(Array A)      ➢ Sortiere A[1,…,n]
1.    step ← 1                  ➢ Hilfsvariablen für Schrittweite
2.    while step <= n do        ➢ Solange noch nicht das ganze Array sortiert ist
3.        left ← 1              ➢ Initialisierung der linken Grenze
4.        while left <= n-step do ➢ Schleife zum Sortieren von Teilarrays
                                    der Länge Step von left an
5.            middle ← left + step - 1   ➢ Hilfsvariablen für Mitte
6.            middle ← min( middle, n)   ➢ Arraygrenzen nicht überschreiten
7.            right ← left + 2 * step - 1 ➢ Hilfsvariablen für rechte Grenze
              right ← min( right, n)      ➢ Arraygrenzen nicht überschreiten

8.            merge(A, left, middle, right) ➢ Merge

9.            left ← left + 2*step        ➢ Verschieben der linken Grenze
10.       step ← step * 2             ➢ Schrittweite erhöhen
```