

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)



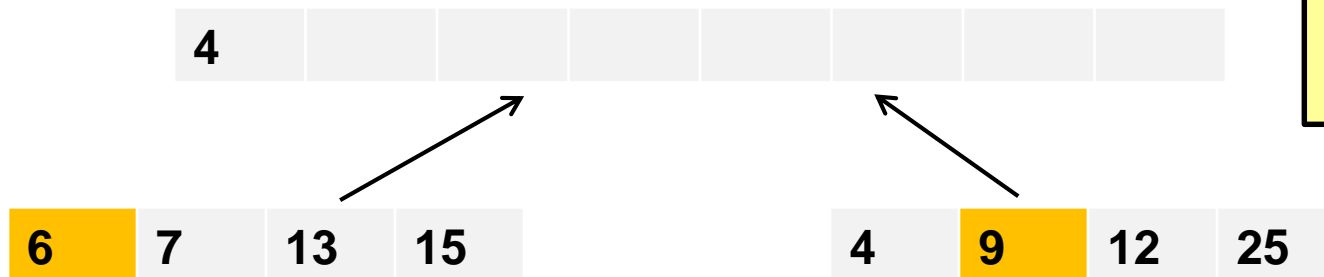
Schritt 3:
Zusammenfügen

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)



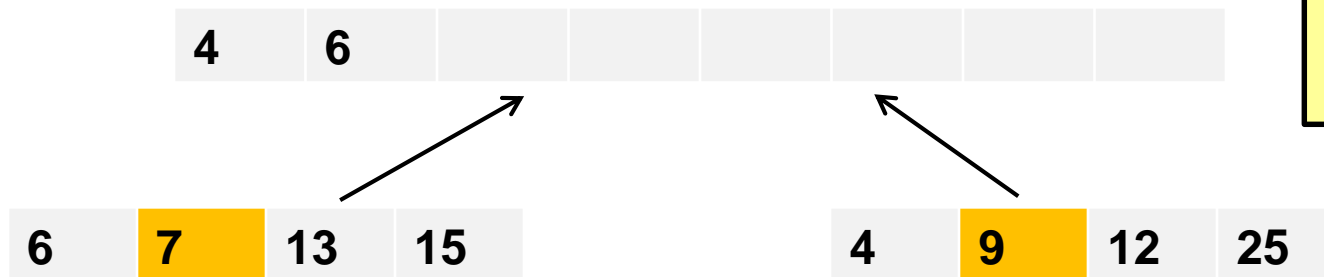
Schritt 3:
Zusammenfügen

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)



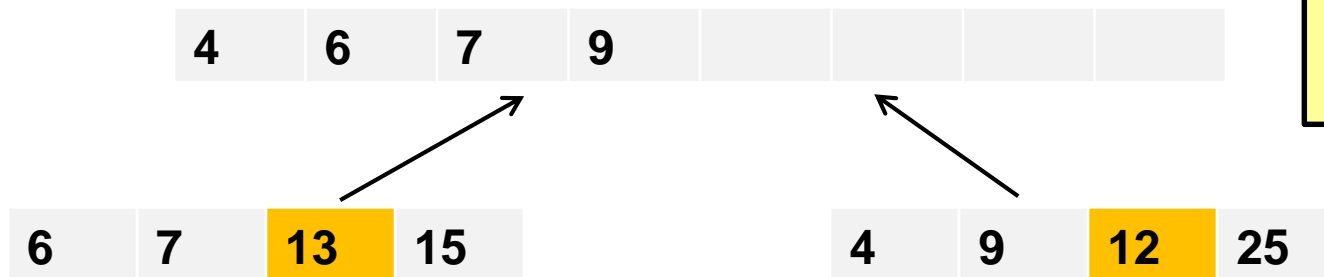
Schritt 3:
Zusammenfügen

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

➤ Sortiere $A[p, \dots, r]$

Teile & Herrsche: MergeSort

MergeSort(Array A, p, r)

➤ Sortiere $A[p, \dots, r]$

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)

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 $A[p, \dots, r]$
- $p \geq r$, 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 $A[p, \dots, r]$
- $p \geq r$, 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 $A[p, \dots, r]$
- $p \geq r$, dann nichts zu tun
- Berechne Mitte

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 $A[p, \dots, r]$
- $p \geq r$, dann nichts zu tun
- Berechne Mitte

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 $A[p, \dots, r]$
- $p \geq r$, dann nichts zu tun
- Berechne Mitte
- Sortiere linke Hälfte

Teile & Herrsche: MergeSort

MergeSort(Array A, p, r)

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

- Sortiere $A[p, \dots, r]$
- $p \geq r$, dann nichts zu tun
- Berechne Mitte
- Sortiere linke Hälfte

Teile & Herrsche: MergeSort

MergeSort(Array A, p, r)

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

- Sortiere $A[p, \dots, r]$
- $p \geq r$, 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. $q \leftarrow \lfloor (p+r)/2 \rfloor$
3. MergeSort(A, p, q)
4. MergeSort(A, q+1, r)
5. Merge(A, p, q, r)

- Sortiere $A[p, \dots, r]$
- $p \geq r$, 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. $q \leftarrow \lfloor (p+r)/2 \rfloor$
3. MergeSort(A, p, q)
4. MergeSort(A, q+1, r)
5. Merge(A, p, q, r)

- Sortiere $A[p, \dots, r]$
- $p \geq r$, dann nichts zu tun
- Berechne Mitte
- Sortiere linke Hälfte
- Sortiere rechte Hälfte
- Zusammenfügen

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 $A[p, \dots, r]$
- $p \geq r$, dann nichts zu tun
- Berechne Mitte
- Sortiere linke Hälfte
- Sortiere rechte Hälfte
- Zusammenfügen

Teile & Herrsche: MergeSort

MergeSort(Array A, p, r)

- | | | |
|----|--|-----------------------------------|
| 1. | if $p < r$ then | ➤ Sortiere $A[p, \dots, r]$ |
| 2. | $q \leftarrow \lfloor (p+r)/2 \rfloor$ | ➤ $p \geq r$, dann nichts zu tun |
| 3. | MergeSort(A, p, q) | ➤ Berechne Mitte |
| 4. | MergeSort(A, q+1, r) | ➤ Sortiere linke Hälfte |
| 5. | Merge(A, p, q, r) | ➤ Sortiere rechte Hälfte |
| | | ➤ Zusammenfügen |

Aufruf des Algorithmus

- MergeSort(A, 1, r) für $r = \text{length}(A)$

Merge(Array A, p, q, r)

1. **Array B**

2. $i \leftarrow p, j \leftarrow q+1$

3. $b \leftarrow 1$

➤ Zusammenfügen von $A[p, \dots, q]$, $A[q+1, \dots, r]$

➤ Hilfsarray zum Mergen Länge $r-p+1$

➤ Hilfsvariablen für *linke* / *rechte* Hälfte von A

➤ Hilfsvariablen für Array B

Merge(Array A, p, q, r)

1. **Array B**

2. $i \leftarrow p, j \leftarrow q+1$

3. $b \leftarrow 1$

4. **while** $i \leq q$ **and** $j \leq r$ **do**

5. **if** $A[i] \leq A[j]$ **then**

6. $B[b] \leftarrow A[i]$

7. $b \leftarrow b + 1$

8. $i \leftarrow i + 1$

➤ Zusammenfügen von $A[p, \dots, q]$, $A[q+1, \dots, r]$

➤ 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(Array A, p, q, r)

1. **Array B**

2. $i \leftarrow p, j \leftarrow q+1$

3. $b \leftarrow 1$

4. **while** $i \leq q$ **and** $j \leq r$ **do**

5. **if** $A[i] \leq A[j]$ **then**

6. $B[b] \leftarrow A[i]$

7. $b \leftarrow b + 1$

8. $i \leftarrow i + 1$

9. **else**

10. $B[b] \leftarrow A[j]$

11. $b \leftarrow b + 1$

12. $j \leftarrow j + 1$

➤ Zusammenfügen von $A[p, \dots, q]$, $A[q+1, \dots, r]$

➤ 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(Array A, p, q, r)

1. **Array B**

2. $i \leftarrow p, j \leftarrow q+1$

3. $b \leftarrow 1$

4. **while** $i \leq q$ **and** $j \leq r$ **do**

5. **if** $A[i] \leq A[j]$ **then**

6. $B[b++] \leftarrow A[i++]$

7. **else**

8. $B[b++] \leftarrow A[j++]$

➤ Zusammenfügen von $A[p, \dots, q]$, $A[q+1, \dots, r]$

➤ 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(Array A, p, q, r)

1. **Array B**

2. $i \leftarrow p, j \leftarrow q+1$

3. $b \leftarrow 1$

4. **while** $i \leq q$ **and** $j \leq r$ **do**

5. **if** $A[i] \leq A[j]$ **then**

6. $B[b++] \leftarrow A[i++]$

7. **else**

8. $B[b++] \leftarrow A[j++]$

9. **while** $i \leq q$ **do**

10. $B[b++] \leftarrow A[i++]$

➤ Zusammenfügen von $A[p, \dots, q]$, $A[q+1, \dots, r]$

➤ 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(Array A, p, q, r)

1. **Array B**

2. $i \leftarrow p, j \leftarrow q+1$

3. $b \leftarrow 1$

4. **while** $i \leq q$ **and** $j \leq r$ **do**

5. **if** $A[i] \leq A[j]$ **then**

6. $B[b++] \leftarrow A[i++]$

7. **else**

8. $B[b++] \leftarrow A[j++]$

9. **while** $i \leq q$ **do**

10. $B[b++] \leftarrow A[i++]$

11. **while** $j \leq r$ **do**

12. $B[b++] \leftarrow A[j++]$

➤ Zusammenfügen von $A[p, \dots, q]$, $A[q+1, \dots, r]$

➤ 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(Array A, p, q, r)

1. **Array B**

2. $i \leftarrow p, j \leftarrow q+1$

3. $b \leftarrow 1$

4. **while** $i \leq q$ **and** $j \leq r$ **do**

5. **if** $A[i] \leq A[j]$ **then**

6. $B[b++] \leftarrow A[i++]$

7. **else**

8. $B[b++] \leftarrow A[j++]$

9. **while** $i \leq q$ **do**

10. $B[b++] \leftarrow A[i++]$

11. **while** $j \leq r$ **do**

12. $B[b++] \leftarrow A[j++]$

13. $A[p, \dots, r] \leftarrow B$

➤ Zusammenfügen von $A[p, \dots, q]$, $A[q+1, \dots, r]$

➤ 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

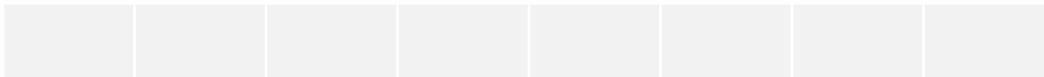
MergeSort: Merge Schritt

Einträge auf beiden Seiten

$\text{merge}(A, 1, 4, 8)$

$b = 1$

B



A



$i = 1$



$j = 5$

Teile & Herrsche: Beispiel Sortieren

Einträge auf beiden Seiten

$merge(A, 1, 4, 8)$

$b = 2$

B



A



$i = 1$

$j = 6$

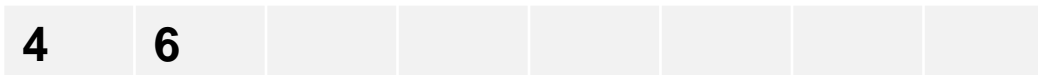
Teile & Herrsche: Beispiel Sortieren

Einträge auf beiden Seiten

$merge(A, 1, 4, 8)$

$b = 3$

B



A



$i = 2$

$j = 6$

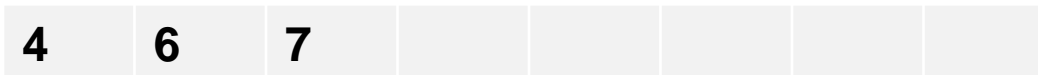
Teile & Herrsche: Beispiel Sortieren

Einträge auf beiden Seiten

$merge(A, 1, 4, 8)$

$b = 4$

B



A



$i = 3$

$j = 6$

Teile & Herrsche: Beispiel Sortieren

Einträge auf beiden Seiten

$merge(A, 1, 4, 8)$

$b = 5$

B



A



$i = 3$

$j = 7$

Teile & Herrsche: Beispiel Sortieren

Einträge auf beiden Seiten

$merge(A, 1, 4, 8)$

$b = 6$

B



A



$i = 4$

$j = 7$

Teile & Herrsche: Beispiel Sortieren

Einträge auf beiden Seiten

$\text{merge}(A, 1, 4, 8)$

$b = 7$

B

4	6	7	9	13	15		
---	---	---	---	----	----	--	--

A

6	7	13	15
---	---	----	----

4	9	24	25
---	---	----	----

$i = 5$

$j = 7$

Teile & Herrsche: Beispiel Sortieren

Einträge nur auf
rechter Seite

$merge(A, 1, 4, 8)$

$b = 8$

B

4	6	7	9	13	15	24	
---	---	---	---	----	----	----	--

A

6	7	13	15
---	---	----	----

4	9	24	25
---	---	----	----

$i = 5$

$j = 8$

Teile & Herrsche: Beispiel Sortieren

Einträge nur auf
rechter Seite

$merge(A, 1, 4, 8)$

$b = 9$

B

4	6	7	9	13	15	24	25
---	---	---	---	----	----	----	----

A

6	7	13	15
---	---	----	----

4	9	24	25
---	---	----	----

$i = 5$

$j = 9$

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)

- | | | |
|----|--|-----------------------------------|
| 1. | if $p < r$ then | ➤ Sortiere $A[p, \dots, r]$ |
| 2. | $q \leftarrow \lfloor (p+r)/2 \rfloor$ | ➤ $p \geq r$, dann nichts zu tun |
| 3. | MergeSort(A, p, q) | ➤ Berechne Mitte |
| 4. | MergeSort(A, q+1, r) | ➤ Sortiere linke Hälfte |
| 5. | Merge(A, p, q, r) | ➤ Sortiere rechte Hälfte |
| | | ➤ Zusammenfügen |

Aufruf des Algorithmus

- MergeSort(A, 1, n) für Feld $A[1 \dots n]$
- Laufzeit?

Teile & Herrsche: MergeSort

MergeSort(Array A, p, r)

- | | | |
|----|--|-----------------------------------|
| 1. | if $p < r$ then | ➤ Sortiere $A[p, \dots, r]$ |
| 2. | $q \leftarrow \lfloor (p+r)/2 \rfloor$ | ➤ $p \geq r$, dann nichts zu tun |
| 3. | MergeSort(A, p, q) | ➤ Berechne Mitte |
| 4. | MergeSort(A, q+1, r) | ➤ Sortiere linke Hälfte |
| 5. | Merge(A, p, q, r) | ➤ Sortiere rechte Hälfte |
| | | ➤ Zusammenfügen |

Aufruf des Algorithmus

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

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)

Laufzeit:

1

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**
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)

1

1

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)

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)

Laufzeit:

1

1

$1+T(n/2)$

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 $A[1\dots n]$
- $T(m)$ = maximale Laufzeit bei Eingabe A, p, r mit $r-p+1=m$

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)

Laufzeit:

1
1
 $1+T(n/2)$
 $1+T(n/2)$

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 $A[1 \dots n]$
- $T(m)$ = maximale Laufzeit bei Eingabe A, p, r mit $r-p+1=m$

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)

Laufzeit:

1

1

$1 + T(n/2)$

$1 + T(n/2)$

$\leq c'n$

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)

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)

Laufzeit:

1

1

$1 + T(n/2)$

$1 + T(n/2)$

$\leq c'n$

$\leq 2T(n/2) + cn$

$c \geq c' + 4$

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

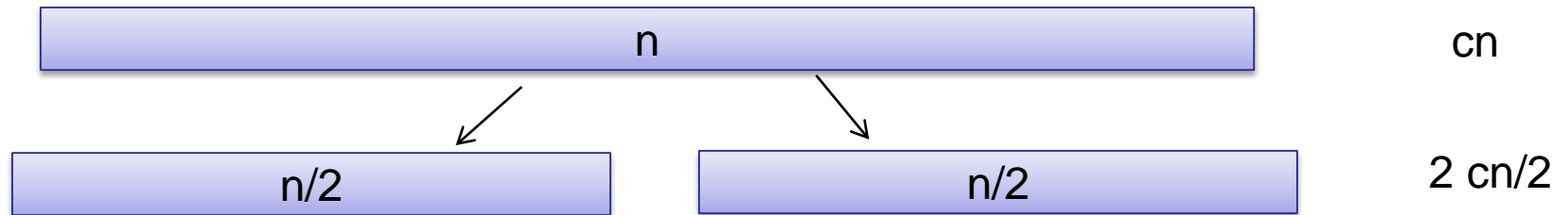
Auflösen von $T(n) \leq 2 T(n/2) + cn$ (Intuition)



cn

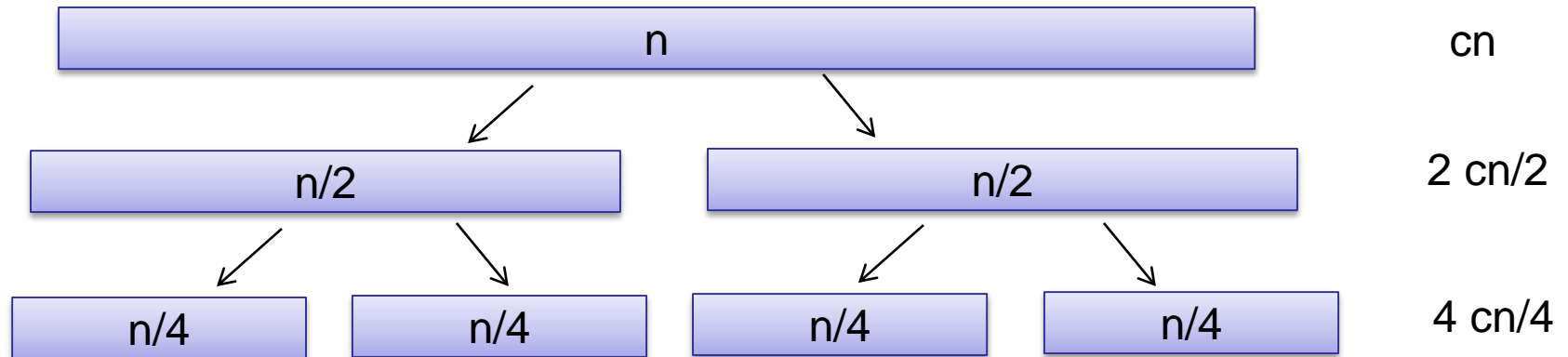
Teile & Herrsche: MergeSort

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



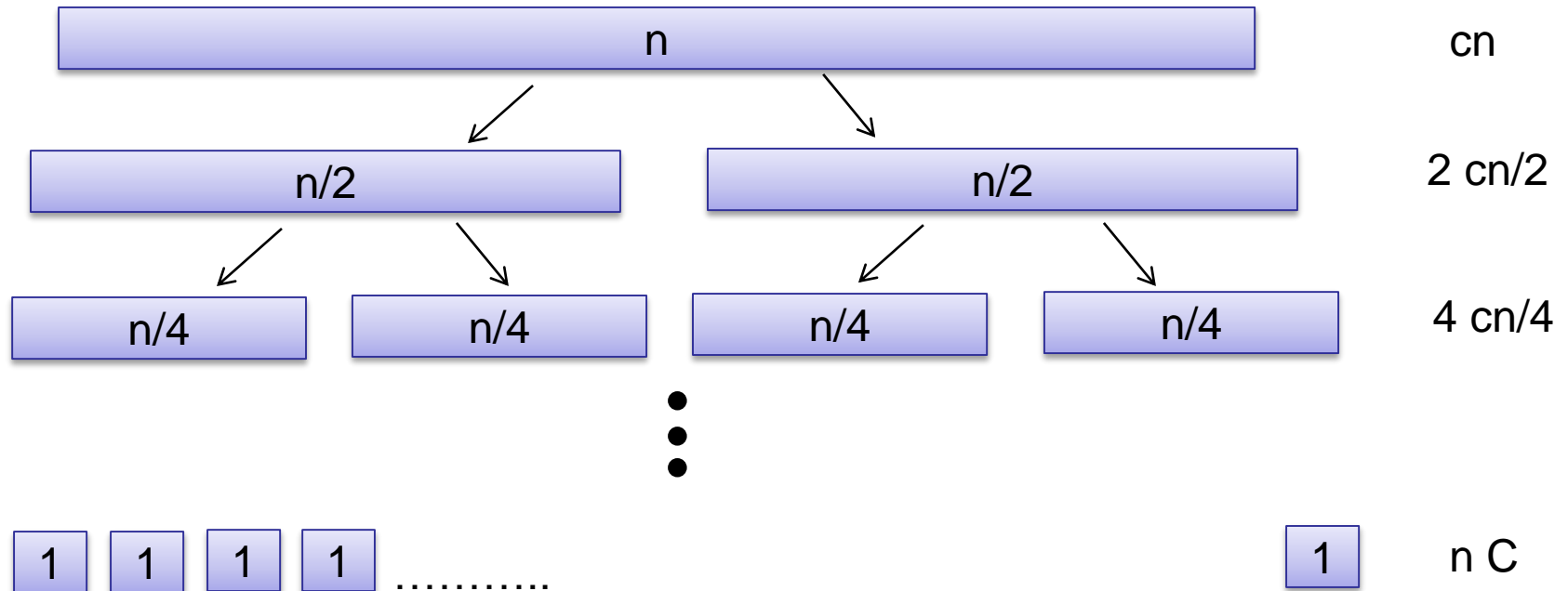
Teile & Herrsche: MergeSort

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



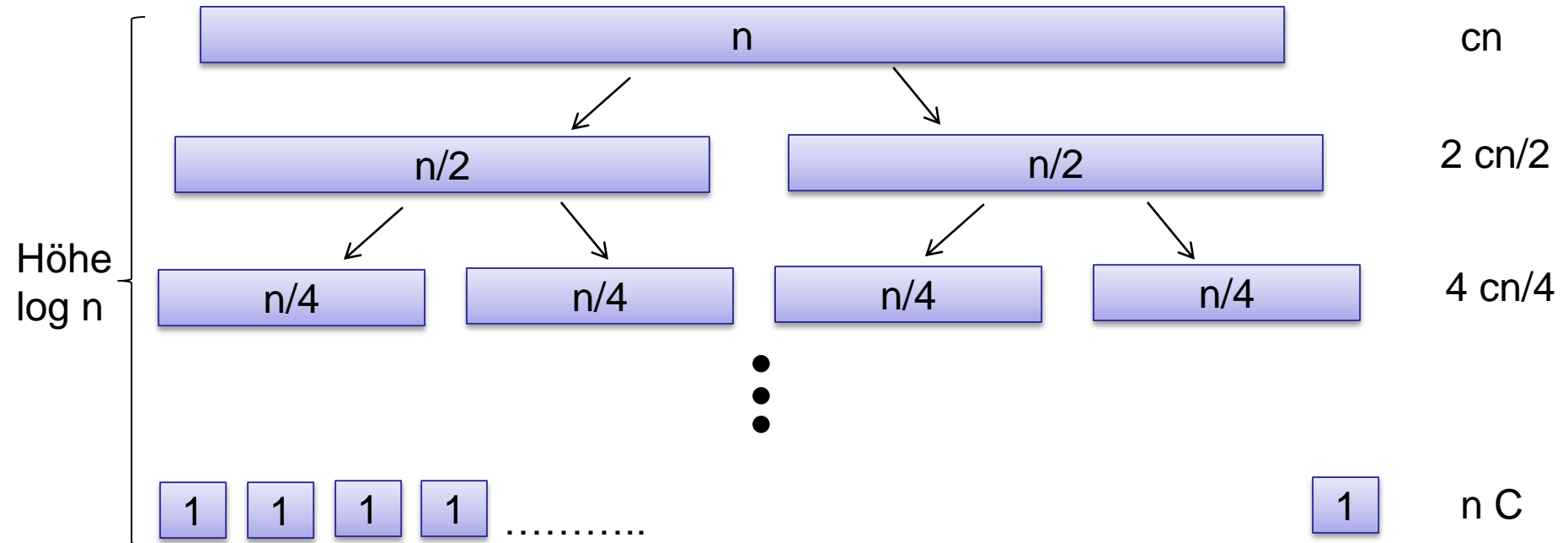
Teile & Herrsche: MergeSort

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



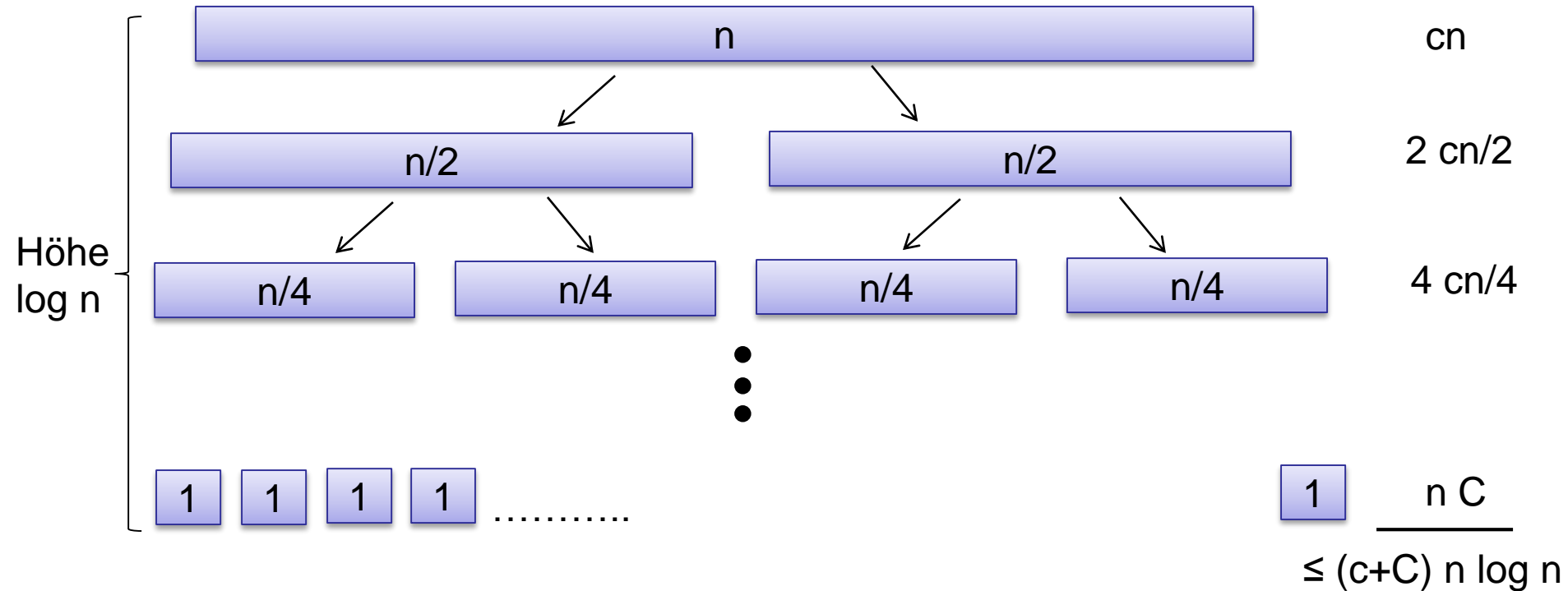
Teile & Herrsche: MergeSort

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



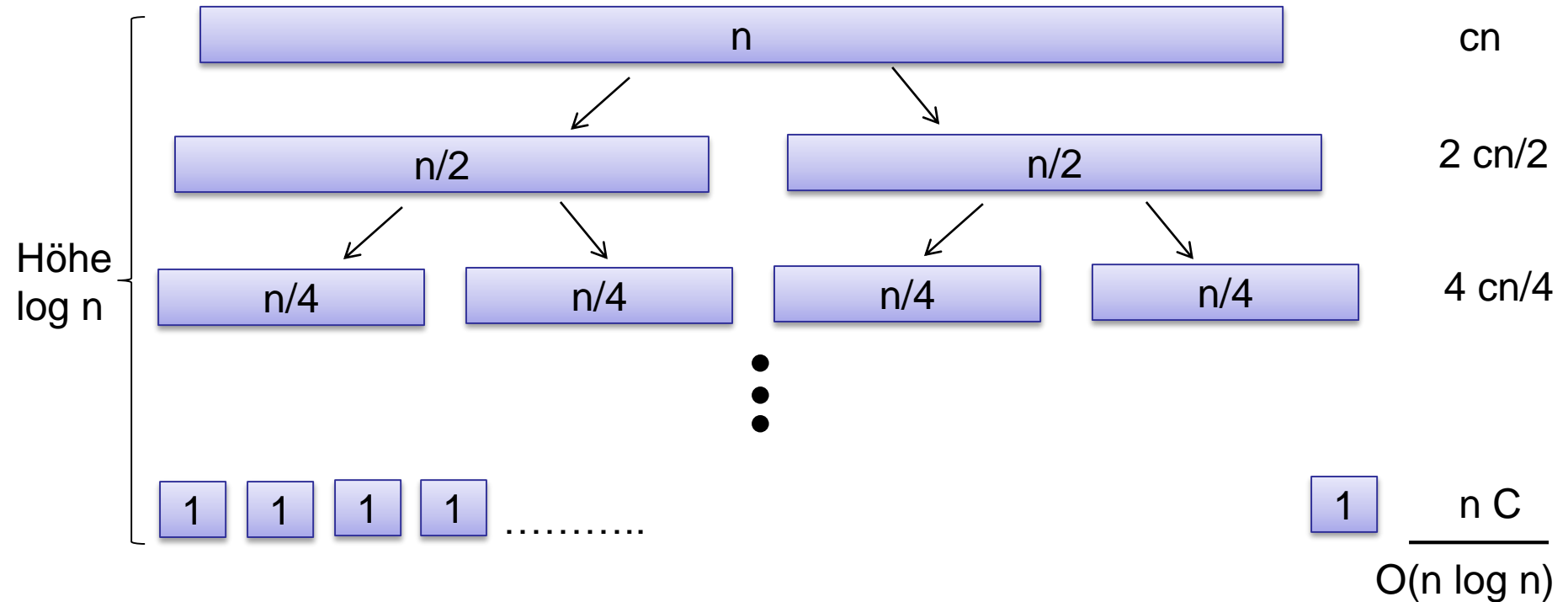
Teile & Herrsche: MergeSort

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 $T(2) \leq C'$ und $C^* \geq \max\{c, C'\}$. Wir zeigen per Induktion, $T(n) \leq C^* n \log n$ für alle $n \geq 2$

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 $T(2) \leq C'$ und $C^* \geq \max\{c, C'\}$. Wir zeigen per Induktion, $T(n) \leq C^* n \log n$ für alle $n \geq 2$
- (I.A.) für $n=2$ gilt $T(2) \leq C' \leq C^* 2 \log 2$.

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 $T(2) \leq C'$ und $C^* \geq \max\{c, C'\}$. Wir zeigen per Induktion, $T(n) \leq C^* n \log n$ für alle $n \geq 2$
- (I.A.) für $n=2$ gilt $T(2) \leq C' \leq C^* 2 \log 2$.
- (I.V.) Für Eingabelänge $m < n$ ist die Laufzeit $T(m) \leq C^* m \log m$.

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 $T(2) \leq C'$ und $C^* \geq \max\{c, C'\}$. Wir zeigen per Induktion, $T(n) \leq C^* n \log n$ für alle $n \geq 2$
- (I.A.) für $n=2$ gilt $T(2) \leq C' \leq C^* 2 \log 2$.
- (I.V.) Für Eingabelänge $m < n$ ist die Laufzeit $T(m) \leq C^* m \log m$.
- (I.S.) Es gilt $T(n) \leq 2 T(n/2) + cn$.

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 $T(2) \leq C'$ und $C^* \geq \max\{c, C'\}$. Wir zeigen per Induktion, $T(n) \leq C^* n \log n$ für alle $n \geq 2$
- (I.A.) für $n=2$ gilt $T(2) \leq C' \leq C^* 2 \log 2$.
- (I.V.) Für Eingabelänge $m < n$ ist die Laufzeit $T(m) \leq C^* m \log m$.
- (I.S.) Es gilt $T(n) \leq 2 T(n/2) + cn$. **Nach (I.V.) gilt**
$$T(n) \leq 2 C^* n/2 \log(n/2) + cn$$

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 $T(2) \leq C'$ und $C^* \geq \max\{c, C'\}$. Wir zeigen per Induktion, $T(n) \leq C^* n \log n$ für alle $n \geq 2$
- (I.A.) für $n=2$ gilt $T(2) \leq C' \leq C^* 2 \log 2$.
- (I.V.) Für Eingabelänge $m < n$ ist die Laufzeit $T(m) \leq C^* m \log m$.
- (I.S.) Es gilt $T(n) \leq 2 T(n/2) + cn$. Nach (I.V.) gilt
$$T(n) \leq 2 C^* n/2 \log(n/2) + cn$$
$$\leq C^* n (\log(n)-1) + cn$$

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 $T(2) \leq C'$ und $C^* \geq \max\{c, C'\}$. Wir zeigen per Induktion, $T(n) \leq C^* n \log n$ für alle $n \geq 2$
- (I.A.) für $n=2$ gilt $T(2) \leq C' \leq C^* 2 \log 2$.
- (I.V.) Für Eingabelänge $m < n$ ist die Laufzeit $T(m) \leq C^* m \log m$.
- (I.S.) Es gilt $T(n) \leq 2 T(n/2) + cn$. Nach (I.V.) gilt
$$\begin{aligned} T(n) &\leq 2 C^* n/2 \log(n/2) + cn \\ &\leq C^* n (\log(n)-1) + cn \\ &\leq C^* n (\log(n)-1) + C^* n \leq C^* n \log(n) \end{aligned}$$

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 $T(2) \leq C'$ und $C^* \geq \max\{c, C'\}$. Wir zeigen per Induktion, $T(n) \leq C^* n \log n$ für alle $n \geq 2$
- (I.A.) für $n=2$ gilt $T(2) \leq C' \leq C^* 2 \log 2$.
- (I.V.) Für Eingabelänge $m < n$ ist die Laufzeit $T(m) \leq C^* m \log m$.
- (I.S.) Es gilt $T(n) \leq 2 T(n/2) + cn$. Nach (I.V.) gilt
$$\begin{aligned} T(n) &\leq 2 C^* n/2 \log(n/2) + cn \\ &\leq C^* n (\log(n)-1) + cn \\ &\leq C^* n (\log(n)-1) + C^* n = C^* n \log(n) \end{aligned}$$
- 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 $T(2) \leq C'$ und $C^* \geq \max\{c, C'\}$. Wir zeigen per Induktion, $T(n) \leq C^* n \log n$ für alle $n \geq 2$
- (I.A.) für $n=2$ gilt $T(2) \leq C' \leq C^* 2 \log 2$.
- (I.V.) Für Eingabelänge $m < n$ ist die Laufzeit $T(m) \leq C^* m \log m$.
- (I.S.) Es gilt $T(n) \leq 2 T(n/2) + cn$. Nach (I.V.) gilt
$$\begin{aligned} T(n) &\leq 2 C^* n/2 \log(n/2) + cn \\ &\leq C^* n (\log(n)-1) + cn \\ &\leq C^* n (\log(n)-1) + C^* n \leq C^* n \log(n) \end{aligned}$$
- Also gilt $T(n) = O(n \log n)$, [da für $n \geq n_0=2$, $T(n) \leq C^* n \log n$ ist]

ALGORITHMISCHE METHODE TEILE & HERRSCHE

Teile & Herrsche

Wodurch unterscheiden sich Teile & Herrsche Algorithmen?

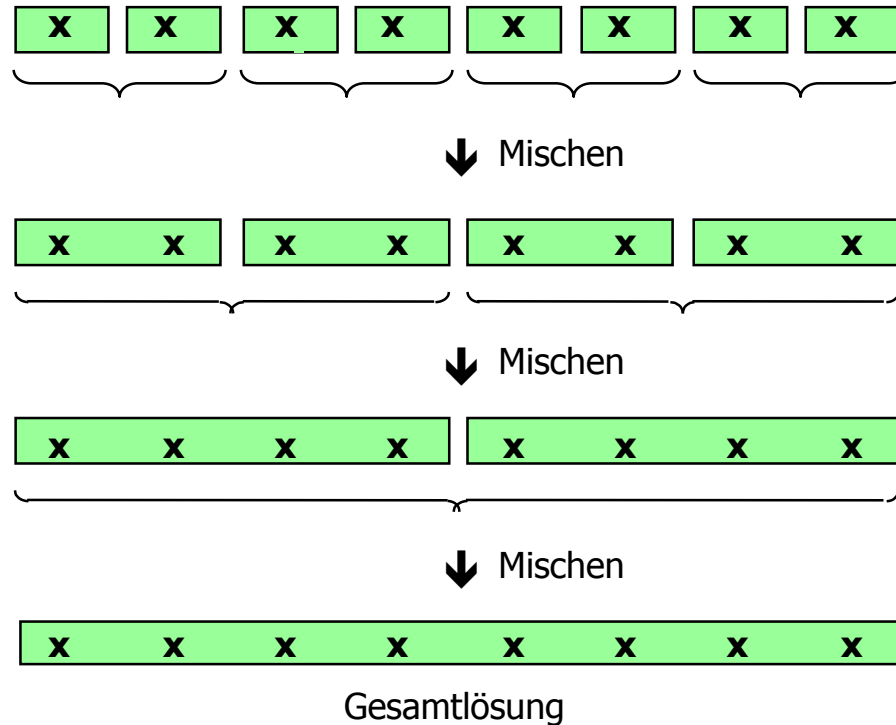
- Die Anzahl der Teilprobleme
- Die Größe der Teilprobleme
- Den Algorithmus für das Zusammensetzen der Teilprobleme
- Den Rekursionsabbruch

ITERATIVE VS. REKURSIV

BEISPIEL: MERGESORT

Merge-Sort: Iterative Variante

- Arbeitsweise:



Merge-Sort: Iterative Variante

■ Beispiel:

30	24	44	58	92	51	44	85	30	16
24	30	44	58	92	51	44	85	30	16
24	30	44	58	92	51	44	85	30	16
24	30	44	58	51	92	44	85	30	16
24	30	44	58	51	92	44	85	30	16
24	30	44	58	51	92	44	85	16	30
24	30	44	58	51	92	44	85	16	30
24	30	44	58	44	51	85	92	16	30
24	30	44	44	51	58	85	92	16	30
16	24	30	30	44	44	51	58	85	92

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

	<i>left= 1</i>	<i>left= 3</i>	<i>left= 5</i>	<i>left= 7</i>	<i>left= 9</i>
	30 24	44 58	92 51	44 85	30 16
<i>step = 1</i>	24 30	44 58	92 51	44 85	30 16
<i>step = 1</i>	24 30	44 58	92 51	44 85	30 16
<i>step = 1</i>	24 30	44 58	51 92	44 85	30 16
<i>step = 1</i>	24 30	44 58	51 92	44 85	30 16
<i>step = 1</i>	24 30	44 58	51 92	44 85	16 30

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

Merge-Sort: Iterative Variante

	<i>left= 1</i>				<i>left= 5</i>				<i>left= 9</i>	
	30	24	44	58	92	51	44	85	30	16
<i>step = 1</i>	24	30	44	58	92	51	44	85	30	16
<i>step = 1</i>	24	30	44	58	92	51	44	85	30	16
<i>step = 1</i>	24	30	44	58	51	92	44	85	30	16
<i>step = 1</i>	24	30	44	58	51	92	44	85	30	16
<i>step = 1</i>	24	30	44	58	51	92	44	85	16	30
<i>step = 2</i>	24	30	44	58	51	92	44	85	16	30
<i>step = 2</i>	24	30	44	58	44	51	85	92	16	30

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

Merge-Sort: Iterative Variante

	<i>left= 1</i>								<i>left= 9</i>	
	30	24	44	58	92	51	44	85	30	16
<i>step = 1</i>	24	30	44	58	92	51	44	85	30	16
<i>step = 1</i>	24	30	44	58	92	51	44	85	30	16
<i>step = 1</i>	24	30	44	58	51	92	44	85	30	16
<i>step = 1</i>	24	30	44	58	51	92	44	85	30	16
<i>step = 1</i>	24	30	44	58	51	92	44	85	16	30
<i>step = 2</i>	24	30	44	58	51	92	44	85	16	30
<i>step = 2</i>	24	30	44	58	44	51	85	92	16	30
<i>step = 4</i>	24	30	44	44	51	58	85	92	16	30

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

Merge-Sort: Iterative Variante

	30	24	44	58	92	51	44	85	30	16
<i>step = 1</i>	24	30	44	58	92	51	44	85	30	16
<i>step = 1</i>	24	30	44	58	92	51	44	85	30	16
<i>step = 1</i>	24	30	44	58	51	92	44	85	30	16
<i>step = 1</i>	24	30	44	58	51	92	44	85	30	16
<i>step = 1</i>	24	30	44	58	51	92	44	85	16	30
<i>step = 2</i>	24	30	44	58	51	92	44	85	16	30
<i>step = 2</i>	24	30	44	58	44	51	85	92	16	30
<i>step = 4</i>	24	30	44	44	51	58	85	92	16	30
<i>step = 8</i>	16	24	30	30	44	44	51	58	85	92

Iterativer MergeSort

IterativMergeSort(Array A)

1. step \leftarrow 1

2. **while** step \leq n **do**

3.

4.

5.

6.

7.

8.

9.

10. step \leftarrow step * 2

➤ Sortiere $A[1, \dots, n]$

➤ Hilfsvariablen für Schrittweite

➤ Solange noch nicht das ganze Array sortiert ist

➤ Schrittweite erhöhen

Iterativer MergeSort

IterativMergeSort(Array A)

1. $\text{step} \leftarrow 1$

2. **while** $\text{step} \leq n$ **do**

3. $\text{left} \leftarrow 1$

4. **while** $\text{left} \leq n - \text{step}$ **do**

➤ Sortiere $A[1, \dots, n]$

➤ Hilfsvariablen für **Schrittweite**

➤ Solange noch nicht das ganze Array sortiert ist

➤ Initialisierung der linken Grenze

➤ Schleife zum Sortieren von Teilarrays
der Länge Step von left an

5.

6.

7.

8.

9. $\text{left} \leftarrow \text{left} + 2 * \text{step}$

➤ Verschieben der linken Grenze

10. $\text{step} \leftarrow \text{step} * 2$

➤ **Schrittweite erhöhen**

Iterativer MergeSort

IterativMergeSort(Array A)

1. $\text{step} \leftarrow 1$
 2. **while** $\text{step} \leq n$ **do**
 3. $\text{left} \leftarrow 1$
 4. **while** $\text{left} \leq n - \text{step}$ **do**
 5. $\text{merge}(\text{A}, \text{left}, \text{middle}, \text{right})$
 6. $\text{left} \leftarrow \text{left} + 2 * \text{step}$
 7. $\text{step} \leftarrow \text{step} * 2$
 8. $\text{merge}(\text{A}, \text{left}, \text{middle}, \text{right})$
 9. $\text{left} \leftarrow \text{left} + 2 * \text{step}$
 10. $\text{step} \leftarrow \text{step} * 2$
- Sortiere $\text{A}[1, \dots, n]$
 - Hilfsvariablen für **Schrittweite**
 - Solange noch nicht das ganze Array sortiert ist
 - Initialisierung der linken Grenze
 - Schleife zum Sortieren von Teilarrays der Länge Step von left an
 - **Merge**
 - Verschieben der linken Grenze
 - **Schrittweite erhöhen**

Iterativer MergeSort

IterativMergeSort(Array A)

1. $\text{step} \leftarrow 1$ ➤ Sortiere $A[1, \dots, n]$
2. **while** $\text{step} \leq n$ **do** ➤ Hilfsvariablen für **Schrittweite**
3. $\text{left} \leftarrow 1$ ➤ Solange noch nicht das ganze Array sortiert ist
4. **while** $\text{left} \leq n - \text{step}$ **do** ➤ Initialisierung der linken Grenze
5. $\text{middle} \leftarrow \text{left} + \text{step} - 1$ ➤ Schleife zum Sortieren von Teilarrays der Länge Step von left an
6. $\text{middle} \leftarrow \min(\text{middle}, n)$ ➤ Hilfsvariablen für Mitte
7. $\text{right} \leftarrow \text{left} + 2 * \text{step} - 1$ ➤ Arraygrenzen nicht überschreiten
8. $\text{right} \leftarrow \min(\text{right}, n)$ ➤ Hilfsvariablen für rechte Grenze
9. $\text{merge}(A, \text{left}, \text{middle}, \text{right})$ ➤ **Merge**
10. $\text{left} \leftarrow \text{left} + 2 * \text{step}$ ➤ Verschieben der linken Grenze
11. $\text{step} \leftarrow \text{step} * 2$ ➤ **Schrittweite erhöhen**