Algorithmische Methode: „Teile & Herrsche“
Beispiel: Mergesort

Manfred Hauswirth | Open Distributed Systems | Einführung in die Programmierung, WS 25/26

Rückblick

Technische Universität Berlin

VL 0 „Organisation und Inhalt“: Ablauf der Vorlesung, Termine
VL 1 „Algorithmen, Pseudocode, Sortieren I“: Insertion Sort
VL 2 „Algorithmen, Pseudocode, Sortieren II“: Selection Sort, Bubble Sort, Count Sort
VL 3 „Laufzeit und Speicherplatz“: Laufzeitanalyse der vorgestellten Sortierverfahren
VL 4 „Einfache Datenstrukturen“: Arrays, verkettete Listen, Structs in C, Stack, Queue
VL 5 „Bäume“: Binärbäume, Baumtraversierung, Laufzeitanalyse Baumoperationen
VL 6 „Teile und Herrsche I“: Einführung der algorithmischen Methode, Merge Sort
VL 7 „Korrektheitsbeweise“: Rechnermodel, Beispielbeweise
VL 8 „Dateien in C“: Dateien, Dateisysteme, Verzeichnisse, Dateiverwaltung mit C
VL 9 „Prioritätenschlangen/Halden/Heaps “: Heap Sort, Binärer Heap, Heap Operationen
VL 10 „Fortgeschrittene Sortierverfahren“: Quick Sort, Radix Sort
VL 11 „AVL Bäume“: Definition, Baumoperationen, Traversierung
VL 12 „Teile und Herrsche II“: Generalisierung des algorithmischen Prinzips, Mastertheorem
VL 13 „Q & A“: Offene Vorlesung/Wiederholung

Open Distributed Sytems
ODS

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)

15	7	6	13	25	4	9	12

15	7	6	13

25	4	9	12

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)

15	7	6	13	25	4	9	12

6	7	13	15

4	9	12	25

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

Schritt 3:
Zusammenfügen

Beispiel (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)

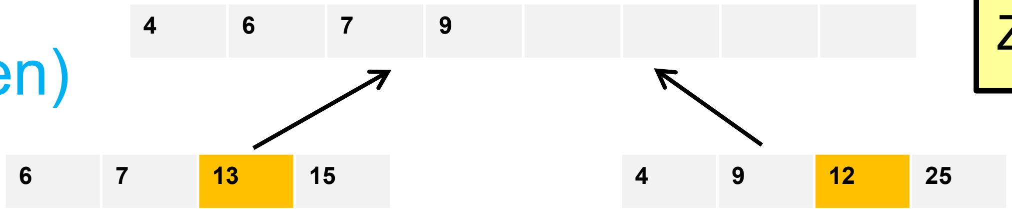
Schritt 3:
Zusammenfügen

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)



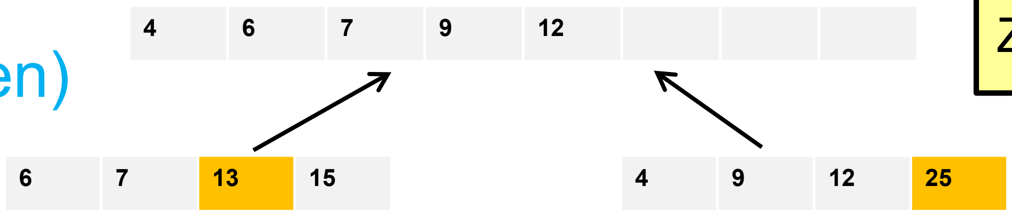
Schritt 3: Zusammenfügen

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

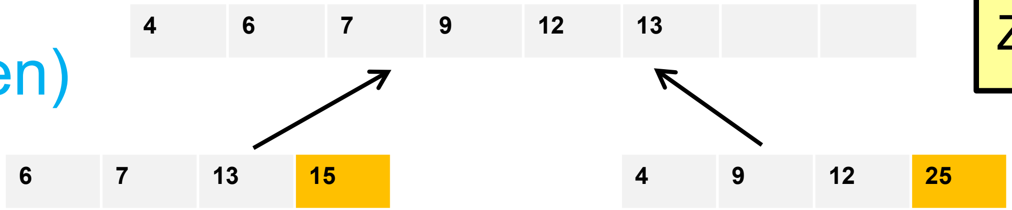
Teile & Herrsche: Beispiel Sortieren

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

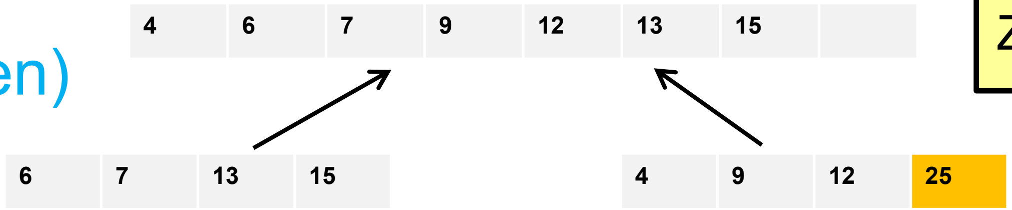


Teile & Herrsche: Beispiel Sortieren

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)

4	6	7	9	12	13	15	25

6	7	13	15

4	9	12	25

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
- Für sortieren: Folgen der Länge 1 sind sortiert

MergeSort

Teile & Herrsche: MergeSort

MergeSort(Array A, p, r)

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

Teile & Herrsche: MergeSort

MergeSort(Array A, p, r)

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

➢ Sortiere A[p,…,r]

Teile & Herrsche: MergeSort

MergeSort(Array A, p, r)

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

- Sortiere A[p,…,r]
- $p \geq r$, dann nichts zu tun

Teile & Herrsche: MergeSort

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

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

- 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)
if p < r then
    q ← ⌊(p+r)/2⌋
    MergeSort(A,p,q)
    MergeSort(A,q+1,r)
    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]
if p < r **then**	➢ p≥r, dann nichts zu tun
$q \leftarrow \lfloor (p+r)/2 \rfloor$	➢ Berechne Mitte
MergeSort(A,p,q)	➢ Sortiere linke Hälfte
MergeSort(A,q+1,r)	➢ Sortiere rechte Hälfte
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. $i \leftarrow p$, $j \leftarrow q+1$
3. $b \leftarrow 1$

- 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)            ➢ Zusammenfügen von A[p,…,q], A[q+1,…,r]
    Array B                        ➢ Hilfsarray zum Mergen Länge r-p+1
    i ← p, j ← q+1                 ➢ Hilfsvariablen für linke / rechte Hälfte von A
    b ←1                           ➢ Hilfsvariablen für Array B
    while i <= q and j <= r do     ➢ Solange Einträge auf beiden Seiten
        if A[i] <= A[j] then       ➢ links kleiner
            B[b] ← A[i]            ➢ Zuweisung nach B
                b ← b + 1          ➢ Hilfsvariablen erhöhen
                i ← i + 1          ➢ Hilfsvariablen erhöhen
```

Merge

```
Merge(Array A, p, q, r)              ➢ Zusammenfügen von A[p,…,q], A[q+1,…,r]
    Array B                          ➢ Hilfsarray zum Mergen Länge r-p+1
    i ← p, j ← q+1                   ➢ Hilfsvariablen für linke / rechte Hälfte von A
    b ←1                             ➢ Hilfsvariablen für Array B
    while i <= q and j <= r do       ➢ Solange Einträge auf beiden Seiten
        if A[i] <= A[j] then         ➢ links kleiner
            B[b] ← A[i]              ➢ Zuweisung nach B
                b ← b + 1            ➢ Hilfsvariablen erhöhen
                i ← i + 1            ➢ Hilfsvariablen erhöhen
        else                         ➢ rechts kleiner
            B[b] ← A[j]              ➢ Zuweisung nach B
                b ← b + 1            ➢ Hilfsvariablen erhöhen
                j ← j + 1            ➢ Hilfsvariablen erhöhen
```

Merge

```
Merge(Array A, p, q, r)          ➢ Zusammenfügen von A[p,…,q], A[q+1,…,r]
    Array B                      ➢ Hilfsarray zum Mergen Länge r-p+1
    i ← p, j ← q+1               ➢ Hilfsvariablen für linke / rechte Hälfte von A
    b ←1                         ➢ Hilfsvariablen für Array B
    while i <= q and j <= r do   ➢ Solange Einträge auf beiden Seiten
        if A[i] <= A[j] then         ➢ links kleiner
            B[b++] ← A[i++]              ➢ Zuweisung nach B
        else                         ➢ rechts kleiner
            B[b++] ← A[j++]              ➢ Zuweisung nach B
```

Merge

```
Merge(Array A, p, q, r)            ➢ Zusammenfügen von A[p,…,q], A[q+1,…,r]
    Array B                          ➢ Hilfsarray zum Mergen Länge r-p+1
    i ← p, j ← q+1                   ➢ Hilfsvariablen für linke / rechte Hälfte von A
    b ←1                             ➢ Hilfsvariablen für Array B
    while i <= q and j <= r do       ➢ Solange Einträge auf beiden Seiten
        if A[i] <= A[j] then           ➢ links kleiner
            B[b++] ← A[i++]              ➢ Zuweisung nach B
        else                           ➢ rechts kleiner
            B[b++] ← A[j++]              ➢ Zuweisung nach B
    while i <= q do                    ➢ Noch Einträge auf der linken Seite
        B[b++] ← A[i++]                  ➢ Zuweisung nach B
```

Merge

```
Merge(Array A, p, q, r)          ➢ Zusammenfügen von A[p,…,q], A[q+1,…,r]
    Array B                      ➢ Hilfsarray zum Mergen Länge r-p+1
    i ← p, j ← q+1               ➢ Hilfsvariablen für linke / rechte Hälfte von A
    b ←1                         ➢ Hilfsvariablen für Array B
    while i <= q and j <= r do   ➢ Solange Einträge auf beiden Seiten
        if A[i] <= A[j] then     ➢ links kleiner
            B[b++] ← A[i++]      ➢ Zuweisung nach B
        else                     ➢ rechts kleiner
            B[b++] ← A[j++]      ➢ Zuweisung nach B
    while i <= q do              ➢ Noch Einträge auf der linken Seite
        B[b++] ← A[i++]          ➢ Zuweisung nach B
    while j <= r do              ➢ Noch Einträge auf der rechten Seite
        B[b++] ← A[j++]          ➢ Zuweisung nach B
```

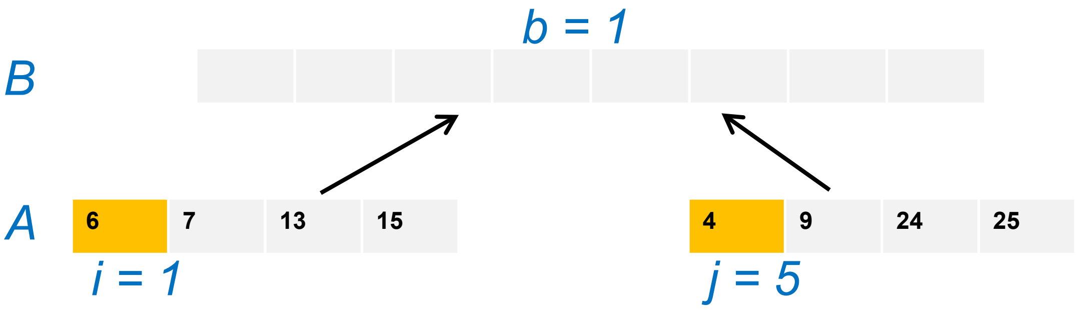
Merge

Merge(Array A, p, q, r)	➢ Zusammenfügen von A[p,…,q], A[q+1,…,r]
Array B	➢ Hilfsarray zum Mergen Länge r-p+1
i ← p, j ← q+1	➢ Hilfsvariablen für linke / rechte Hälfte von A
b ←1	➢ Hilfsvariablen für Array B
while i <= q **and** j <= r **do**	➢ Solange Einträge auf beiden Seiten
if A[i] <= A[j] **then**	➢ links kleiner
B[b++] ← A[i++]	➢ Zuweisung nach B
else	➢ rechts kleiner
B[b++] ← A[j++]	➢ Zuweisung nach B
while i <= q **do**	➢ Noch Einträge auf der linken Seite
B[b++] ← A[i++]	➢ Zuweisung nach B
while j <= r **do**	➢ Noch Einträge auf der rechten Seite
B[b++] ← A[j++]	➢ Zuweisung nach B
A[p, …, r] ← B	➢ Kopiere Hilfsarray B nach A

MergeSort: Merge Schritt

Einträge auf beiden Seiten

merge(A, 1, 4, 8)

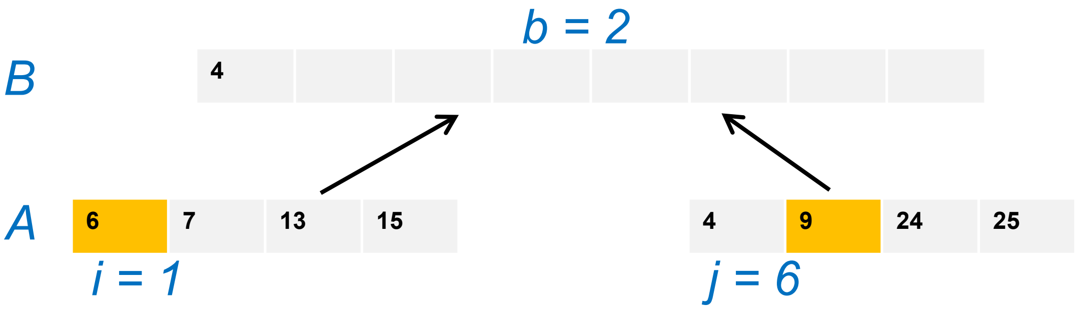


Teile & Herrsche: Beispiel Sortieren

Technische Universität Berlin

Einträge auf beiden Seiten

merge(A, 1, 4, 8)

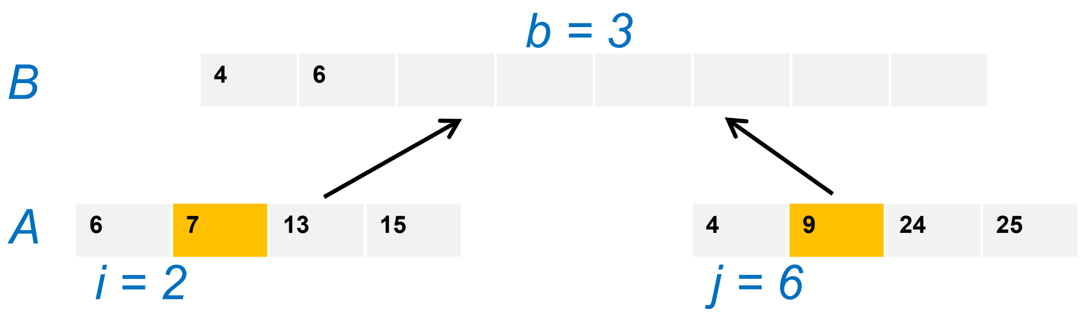


Open Distributed Sytems
ODS

Teile & Herrsche: Beispiel Sortieren

Einträge auf beiden Seiten

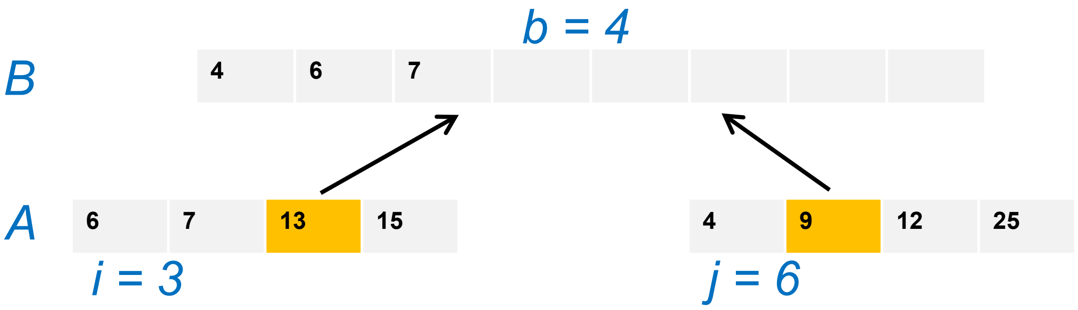
merge(A, 1, 4, 8)



Teile & Herrsche: Beispiel Sortieren

Einträge auf beiden Seiten

merge(A, 1, 4, 8)



Teile & Herrsche: Beispiel Sortieren

Einträge auf beiden Seiten

merge(A, 1, 4, 8)

$b = 5$

B								
	4	6	7	9				

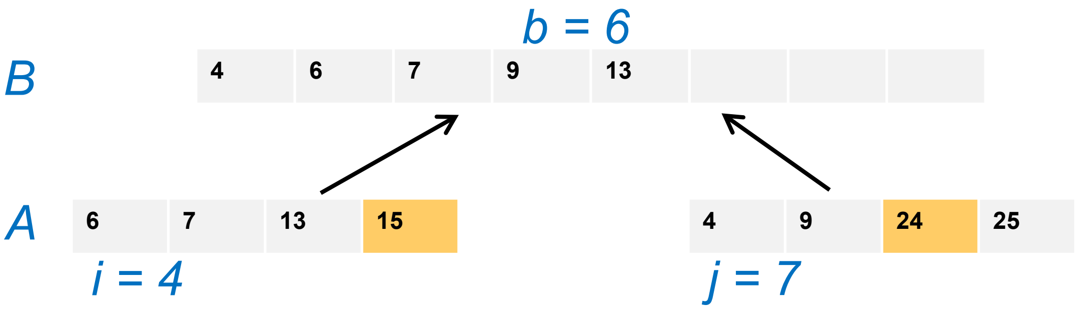
A								
	6	7	13	15	4	9	24	25

$i = 3$ $j = 7$

Teile & Herrsche: Beispiel Sortieren

Einträge auf beiden Seiten

merge(A, 1, 4, 8)



Teile & Herrsche: Beispiel Sortieren

Einträge auf beiden Seiten

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

$i = 5$ $\qquad$ $j = 9$

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**
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
- Sortiere linke Hälfte
- Sortiere rechte Hälfte
- Zusammenfügen

Aufruf des Algorithmus

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

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
- Sortiere linke Hälfte
- Sortiere rechte Hälfte
- 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. $\boxed{\textbf{if } p < r \textbf{ then}}$ | 1
2. $\quad q \leftarrow \lfloor (p+r)/2 \rfloor$
3. $\quad$ MergeSort(A,p,q)
4. $\quad$ MergeSort(A,q+1,r)
5. $\quad$ 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. $\boxed{q \leftarrow \lfloor (p+r)/2 \rfloor}$	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)
1. **if** p < r **then**
2. q ← ⌊(p+r)/2⌋
3. $\boxed{\text{MergeSort(A,p,q)}}$
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...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)	1+T(n/2)
4. $\boxed{\text{MergeSort(A,q+1,r)}}$	1+T(n/2)
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)	1+T(n/2)
4. MergeSort(A,q+1,r)	1+T(n/2)
5. Merge(A,p,q,r)	$\leq c'n$

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)	Laufzeit:
1. **if** $p < r$ **then**	1
2. $q \leftarrow \lfloor (p+r)/2 \rfloor$	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)	$\leq c'n$
	$\leq 2T(n/2) + cn$

$\boxed{c \geq c'+4}$

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

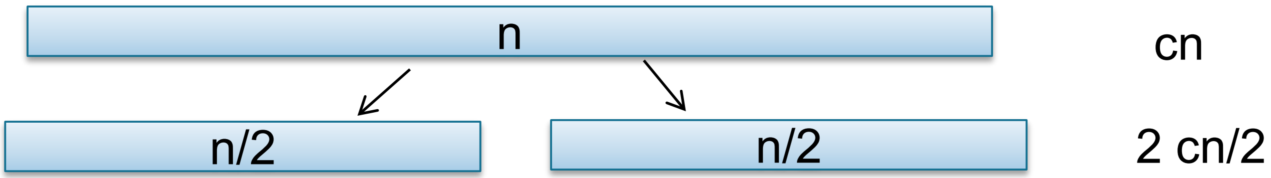
- Auflösen von $T(n) \leq 2\,T(n/2) + cn$ (Intuition)

n

cn

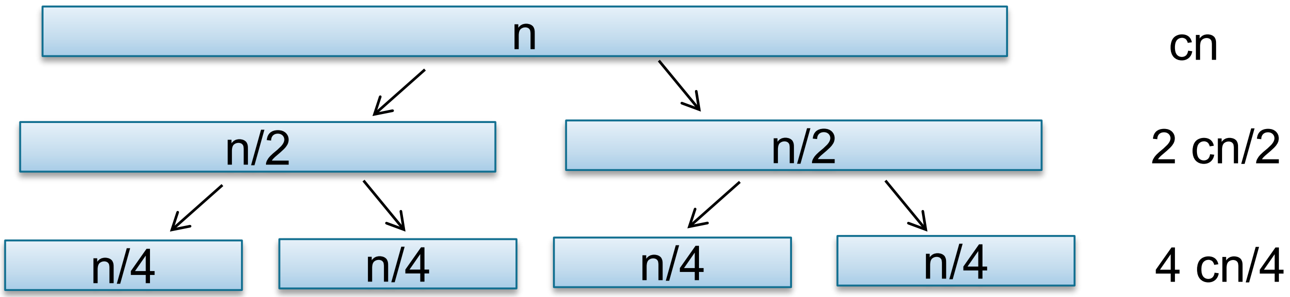
Teile & Herrsche: MergeSort

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



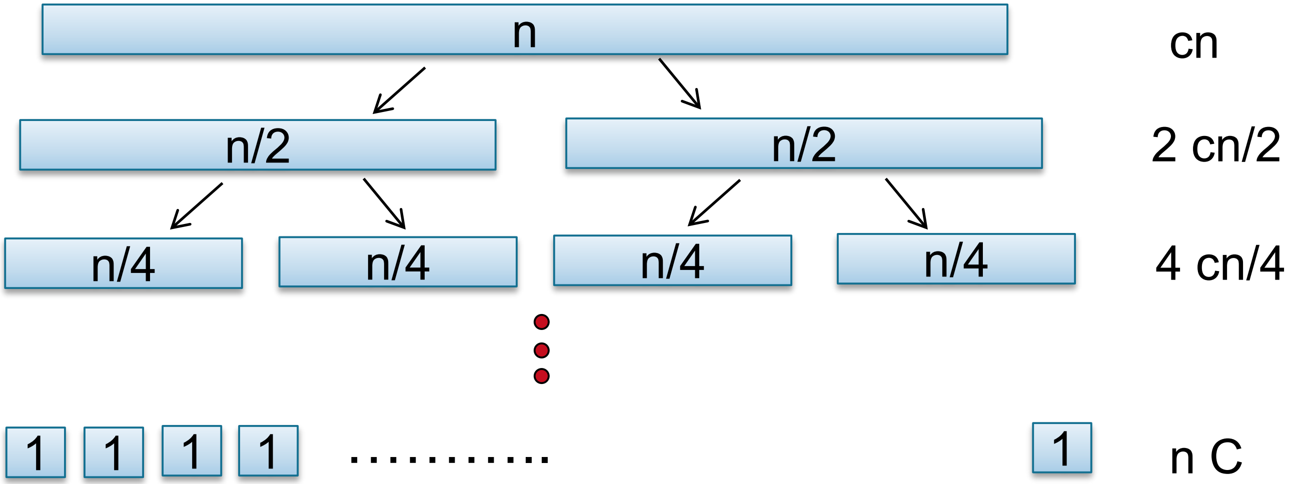
Teile & Herrsche: MergeSort

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



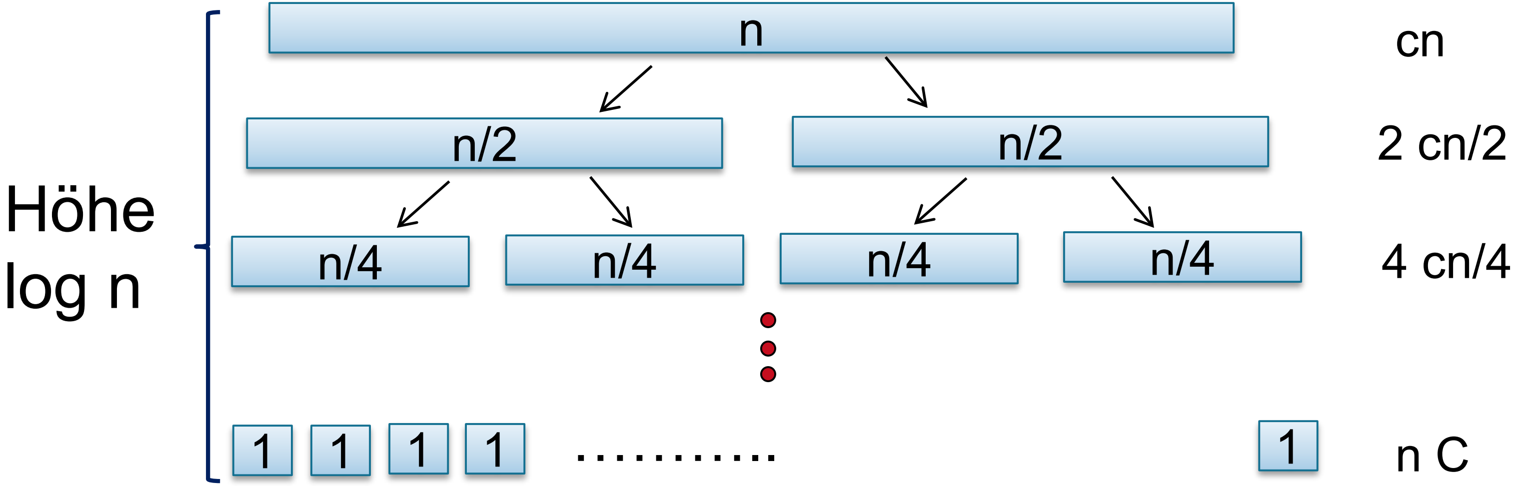
Teile & Herrsche: MergeSort

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



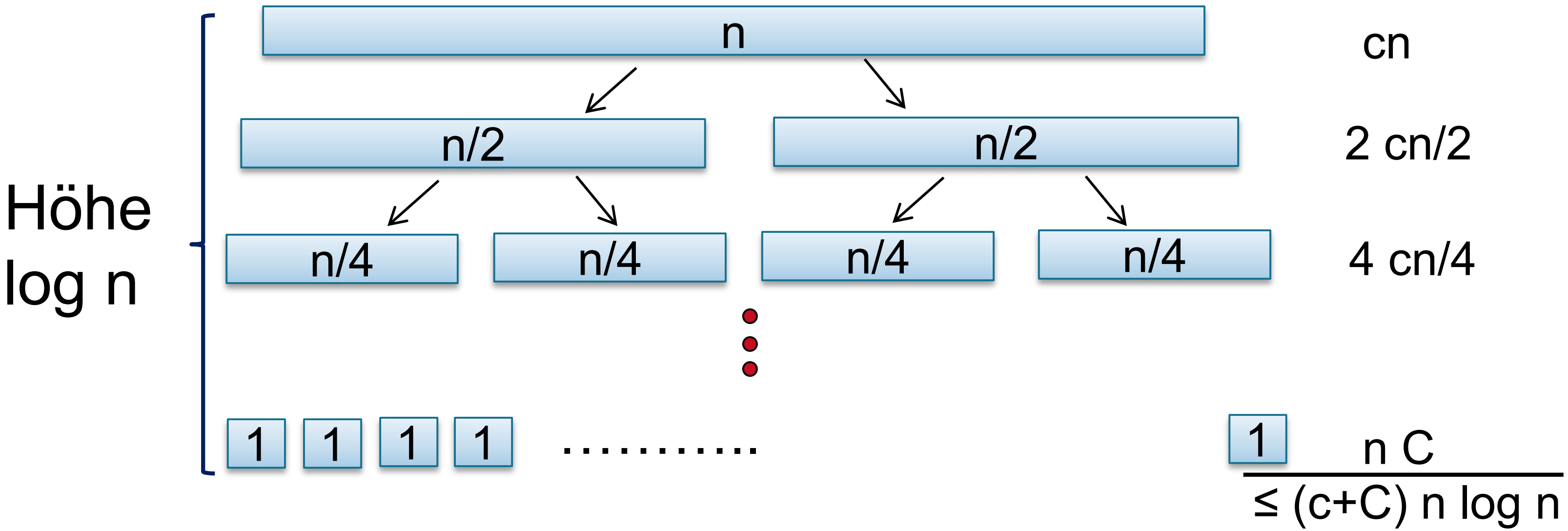
Teile & Herrsche: MergeSort

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



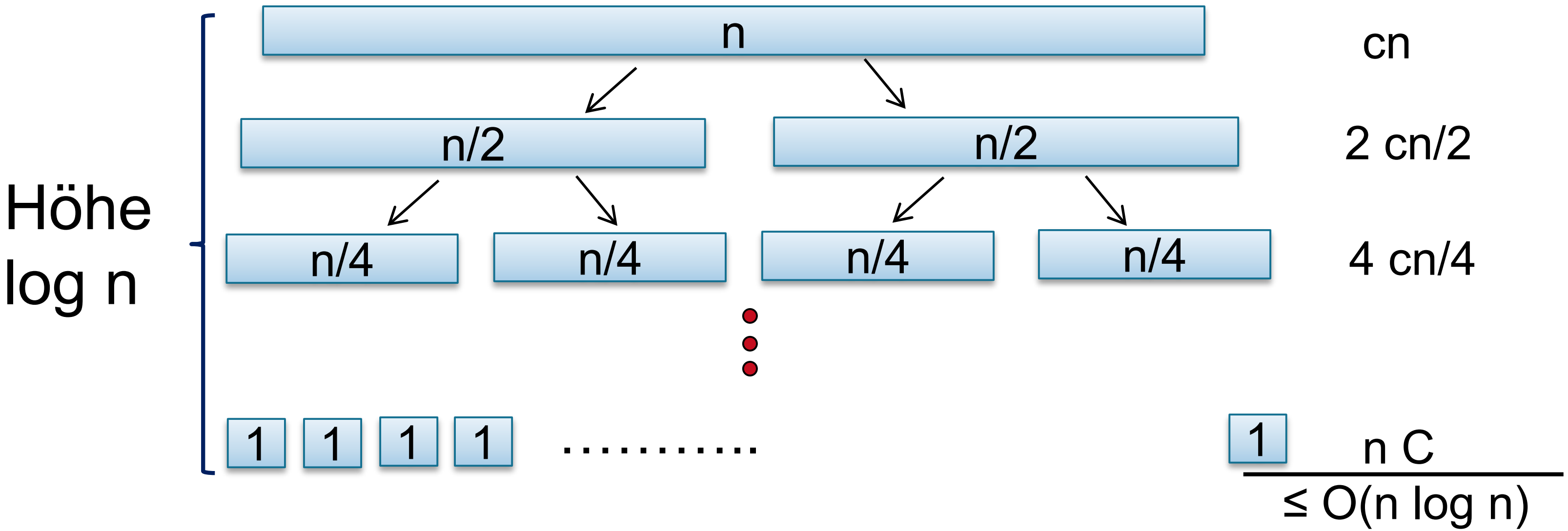
Teile & Herrsche: MergeSort

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

Algorithmische Methode Teile & Herrsche

Teile & Herrsche

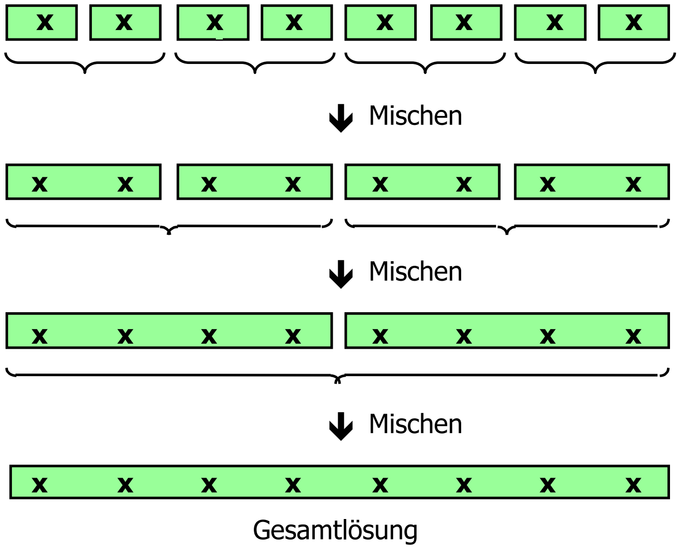
Wodurch unterscheiden sich Teile & Herrsche Algorithmen?

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

Iterativ 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

	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

	left= 1				left= 5				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
step = 2	24	30	44	58	51	92	44	85	16	30
step = 2	24	30	44	58	44	51	85	92	16	30

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

Merge-Sort: Iterative Variante

	left= 1								*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
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

*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]
    step ← 1                          ➢ Hilfsvariablen für Schrittweite
    while step < n do                 ➢ Solange noch nicht das ganze Array sortiert ist

        step ← step * 2                 ➢ Schrittweite erhöhen
```

Iterativer MergeSort

IterativMergeSort(Array A)	➢ Sortiere A[1,…,n]
step ← 1	➢ Hilfsvariablen für Schrittweite
while step < n **do**	➢ Solange noch nicht das ganze Array sortiert ist
left ← 1	➢ Initialisierung der linken Grenze
while left <= n-step **do**	➢ Schleife zum Sortieren von Teilarrays der Länge step von left an
left ← left + 2*step	➢ Verschieben der linken Grenze
step ← step * 2	➢ Schrittweite erhöhen

Iterativer MergeSort

Technische Universität Berlin

```
IterativMergeSort(Array A)
    step ← 1
    while step < n do
        left ← 1
        while left <= n-step do
            merge(A, left, middle, right)
            left ← left + 2*step
        step ← step * 2
```

- Sortiere A[1,…,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

Open Distributed Sytems
ODS

Iterativer MergeSort

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

Ausblick