technische universität
dortmund

LS 2 / Informatik

Datenstrukturen, Algorithmen und Programmierung 2 (DAP2)

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

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

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

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

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

6	**7**	**13**	**15**

4	**9**	**12**	**25**

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

Beispiel (Sortieren)

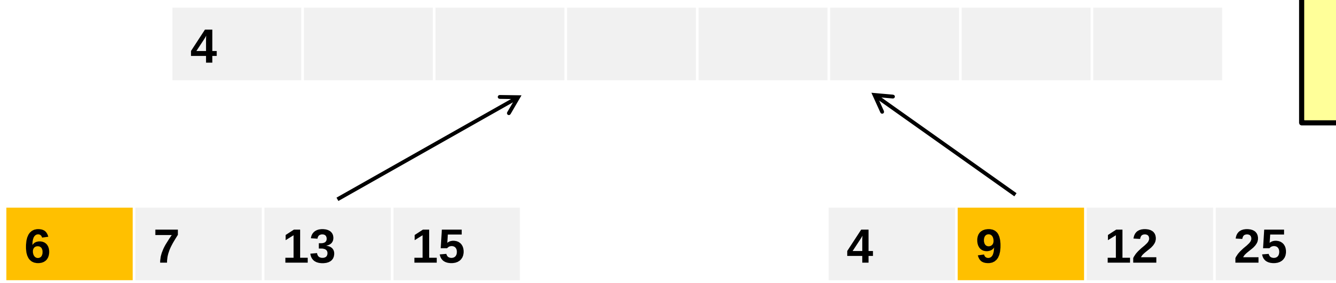
Schritt 3:
Zusammenfügen

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

Beispiel (Sortieren)



Schritt 3: Zusammenfügen

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

Beispiel (Sortieren)

Schritt 3:
Zusammenfügen

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

Beispiel (Sortieren)

Schritt 3: Zusammenfügen

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

Beispiel (Sortieren)

Schritt 3:
Zusammenfügen

4	**6**	**7**	**9**				

6	**7**	**13**	**15**

4	**9**	**12**	**25**

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

Beispiel (Sortieren)

Schritt 3:
Zusammenfügen

4	**6**	**7**	**9**	**12**			

6	**7**	**13**	**15**

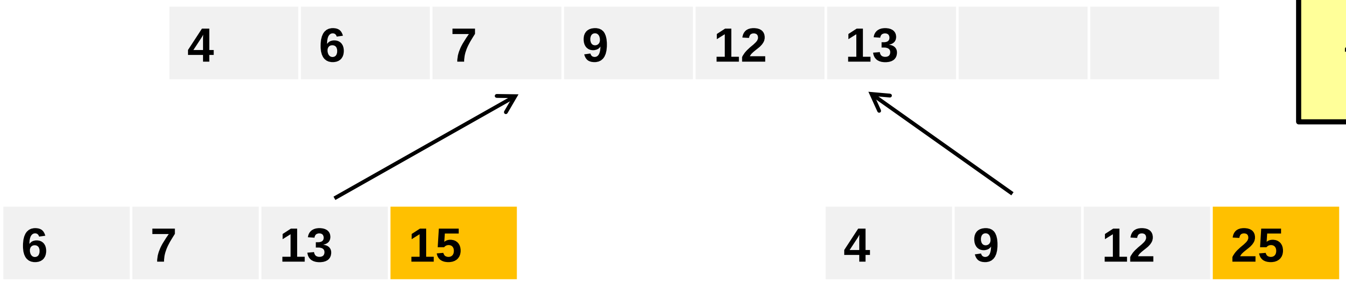
4	**9**	**12**	**25**

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

Beispiel (Sortieren)



Schritt 3: Zusammenfügen

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

Beispiel (Sortieren)

Schritt 3: Zusammenfügen

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

6	**7**	**13**	**15**

4	**9**	**12**	**25**

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

Beispiel (Sortieren)

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

6	**7**	**13**	**15**

4	**9**	**12**	**25**

Schritt 3:
Zusammenfügen

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

Wichtig

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

Teile & Herrsche

$\text{MergeSort}(\text{Array } A, p, r)$

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

Teile & Herrsche

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 $A[p..r]$

Teile & Herrsche

MergeSort(Array A, p, r)

1. $\boxed{\textbf{if } p < r \textbf{ then}}$
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 $A[p..r]$

Teile & Herrsche

MergeSort(Array A, p, r)

1. $\boxed{\textbf{if } p < r \textbf{ then}}$
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

Teile & Herrsche

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

Teile & Herrsche

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(Array A, p, r)

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

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

Teile & Herrsche

MergeSort(Array A, p, r)

1. **if** $p < r$ **then**
2. $q \leftarrow \lfloor (p+r)/2 \rfloor$
3. $\boxed{\text{MergeSort}(A, p, q)}$
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(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 $A[p..r]$
- $p \geq r$, dann nichts zu tun
- Berechne Mitte
- Sortiere linke Hälfte

Teile & Herrsche

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

Teile & Herrsche

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

Teile & Herrsche

$\text{MergeSort}(\text{Array } A, p, r)$

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

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

Teile & Herrsche

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
- Sortiere rechte Hälfte
- Zusammenfügen

Aufruf des Algorithmus

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

Teile & Herrsche

Erweiterte Induktion

- (I.A.) Aussage $A(1)$ ist richtig
- (I.V.) Aussage $A(m)$ gilt für alle $1 \leq m \leq n$
- (I.S.) Aus (I.V.) folgt Aussage $A(n+1)$

- Bisher hatten wir nur $A(n)$ benutzt, um $A(n+1)$ zu folgern. Nun nutzen wir alle $A(m)$ mit $1 \leq m \leq n$ (oder eine Teilmenge).

Teile & Herrsche

Satz 4

Algorithmus $\mathrm{MergeSort}(A, p, r)$ sortiert das Feld $A[p..r]$ korrekt.

Teile & Herrsche

Satz 4

Algorithmus $\text{MergeSort}(A, p, r)$ sortiert das Feld $A[p..r]$ korrekt.

Beweis

- Wir zeigen die Korrektheit per Induktion über $n = r - p$.

Teile & Herrsche

Satz 4

Algorithmus $\mathrm{MergeSort}(A, p, r)$ sortiert das Feld $A[p..r]$ korrekt.

Beweis

- Wir zeigen die Korrektheit per Induktion über $n = r - p$.
- (I.A.) Für $n = 0$, d.h. $p = r$, macht der Algorithmus nichts. Das Feld $A[p..r]$ enthält nur ein Element und ist somit sortiert.

Teile & Herrsche

Satz 4

Algorithmus $\text{MergeSort}(A, p, r)$ sortiert das Feld $A[p..r]$ korrekt.

Beweis

- Wir zeigen die Korrektheit per Induktion über $n = r - p$.
- (I.A.) Für $n = 0$, d.h. $p = r$, macht der Algorithmus nichts. Das Feld $A[p..r]$ enthält nur ein Element und ist somit sortiert.
- (I.V.) Für alle r, p mit $m = r - p$ und $0 \leq m \leq n$ sortiert $\text{MergeSort}(A, p, r)$ das Feld $A[p..r]$ korrekt.

Teile & Herrsche

Satz 4

Algorithmus $\mathrm{MergeSort}(A, p, r)$ sortiert das Feld $A[p..r]$ korrekt.

Beweis

- (I.V.) Für alle r, p mit $m = r - p$ und $0 \leq m \leq n$ sortiert $\mathrm{MergeSort}(A, p, r)$ das Feld $A[p..r]$ korrekt.

Teile & Herrsche

Satz 4

Algorithmus $\mathrm{MergeSort}(A, p, r)$ sortiert das Feld $A[p..r]$ korrekt.

Beweis

- (I.V.) Für alle r, p mit $m = r - p$ und $0 \leq m \leq n$ sortiert $\mathrm{MergeSort}(A, p, r)$ das Feld $A[p..r]$ korrekt.
- (I.S.) Wir betrachten den Aufruf von $\mathrm{MergeSort}$ für beliebige p, r mit $n + 1 = r - p$. Da $n + 1 > 0$ folgt $p < r$ und der Algorithmus führt den **then**-Fall aus. Hier wird q auf $\lfloor (p + r)/2 \rfloor$ gesetzt.

Teile & Herrsche

Satz 4

Algorithmus $\mathrm{MergeSort}(A, p, r)$ sortiert das Feld $A[p..r]$ korrekt.

Beweis

- (I.V.) Für alle r, p mit $m = r - p$ und $0 \leq m \leq n$ sortiert $\mathrm{MergeSort}(A, p, r)$ das Feld $A[p..r]$ korrekt.
- (I.S.) Wir betrachten den Aufruf von $\mathrm{MergeSort}$ für beliebige p, r mit $n + 1 = r - p$. Da $n + 1 > 0$ folgt $p < r$ und der Algorithmus führt den **then**-Fall aus. Hier wird q auf $\lfloor (p + r)/2 \rfloor$ gesetzt. Es gilt $q \geq p$ und $q < r$.

Teile & Herrsche

Satz 4

Algorithmus $\mathrm{MergeSort}(A, p, r)$ sortiert das Feld $A[p..r]$ korrekt.

Beweis

- (I.V.) Für alle r, p mit $m = r - p$ und $0 \leq m \leq n$ sortiert $\mathrm{MergeSort}(A, p, r)$ das Feld $A[p..r]$ korrekt.
- (I.S.) Wir betrachten den Aufruf von $\mathrm{MergeSort}$ für beliebige p, r mit $n + 1 = r - p$. Da $n + 1 > 0$ folgt $p < r$ und der Algorithmus führt den **then**-Fall aus. Hier wird q auf $\lfloor (p + r)/2 \rfloor$ gesetzt. Es gilt $q \geq p$ und $q < r$. Dann wird $\mathrm{MergeSort}$ rekursiv in den Grenzen p, q bzw. $q + 1, r$ aufgerufen.

Teile & Herrsche

Satz 4

Algorithmus $\text{MergeSort}(A, p, r)$ sortiert das Feld $A[p..r]$ korrekt.

Beweis

- (I.V.) Für alle r, p mit $m = r - p$ und $0 \leq m \leq n$ sortiert $\text{MergeSort}(A, p, r)$ das Feld $A[p..r]$ korrekt.
- (I.S.) Wir betrachten den Aufruf von MergeSort für beliebige p, r mit $n + 1 = r - p$. Da $n + 1 > 0$ folgt $p < r$ und der Algorithmus führt den **then**-Fall aus. Hier wird q auf $\lfloor (p + r)/2 \rfloor$ gesetzt. Es gilt $q \geq p$ und $q < r$. Dann wird MergeSort rekursiv in den Grenzen p, q bzw. $q + 1, r$ aufgerufen. Nach (I.V.) sortiert MergeSort in diesem Fall korrekt.

Teile & Herrsche

Satz 4

Algorithmus $\mathrm{MergeSort}(A,p,r)$ sortiert das Feld $A[p..r]$ korrekt.

Beweis

- (I.V.) Für alle r,p mit $m=r-p$ und $0 \le m \le n$ sortiert $\mathrm{MergeSort}(A,p,r)$ das Feld $A[p..r]$ korrekt.
- (I.S.) Wir betrachten den Aufruf von $\mathrm{MergeSort}$ für beliebige p,r mit $n+1=r-p$. Da $n+1>0$ folgt $p<r$ und der Algorithmus führt den **then**-Fall aus. Hier wird q auf $\lfloor (p+r)/2 \rfloor$ gesetzt. Es gilt $q \ge p$ und $q < r$. Dann wird $\mathrm{MergeSort}$ rekursiv in den Grenzen p,q bzw. $q+1,r$ aufgerufen. Nach (I.V.) sortiert $\mathrm{MergeSort}$ in diesem Fall korrekt. Nun folgt die Korrektheit aus der Tatsache, dass Merge die beiden Bereiche korrekt zu einem sortierten Feld zusammenfügt.

Teile & Herrsche

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
- Sortiere rechte Hälfte
- Zusammenfügen

Aufruf des Algorithmus

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

Teile & Herrsche

$\mathrm{MergeSort}(\mathrm{Array}\ A, p, r)$

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

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

Aufruf des Algorithmus

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

Teile & Herrsche

MergeSort(Array A, p, r)

1. $\boxed{\textbf{if } p < r \textbf{ then}}$
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(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)

Laufzeit:

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

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

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

Aufruf des Algorithmus

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

Teile & Herrsche

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

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

Aufruf des Algorithmus

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

Teile & Herrsche

MergeSort(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. $\boxed{\text{Merge}(A, p, q, r)}$	$\le c'n$

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

	Laufzeit:
$\text{MergeSort}(\text{Array } A, p, r)$	
1. $\;$ **if** $p < r$ **then**	1
2. $\quad q \leftarrow \lfloor (p+r)/2 \rfloor$	1
3. $\quad \text{MergeSort}(A, p, q)$	$1 + T(n/2)$
4. $\quad \text{MergeSort}(A, q+1, r)$	$1 + T(n/2)$
5. $\quad \boxed{\text{Merge}(A, p, q, r)}$	$\le c'n$
	$\le 2T(n/2) + cn$ $\quad \boxed{c \ge c' + 4}$

Aufruf des Algorithmus

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

Teile & Herrsche

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

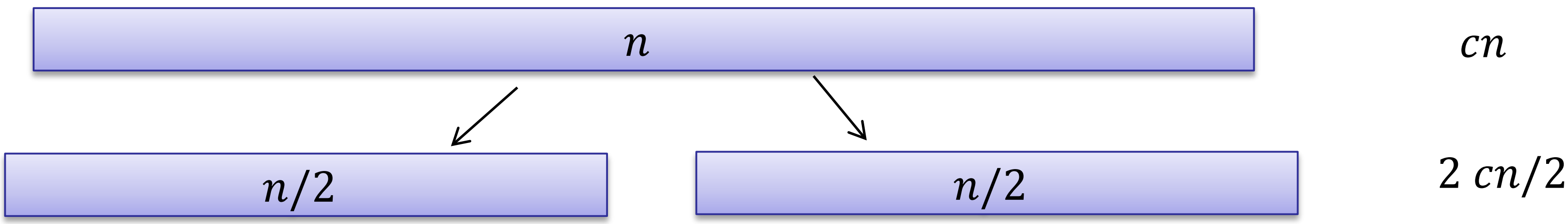
Teile & Herrsche

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

n $\qquad$ cn

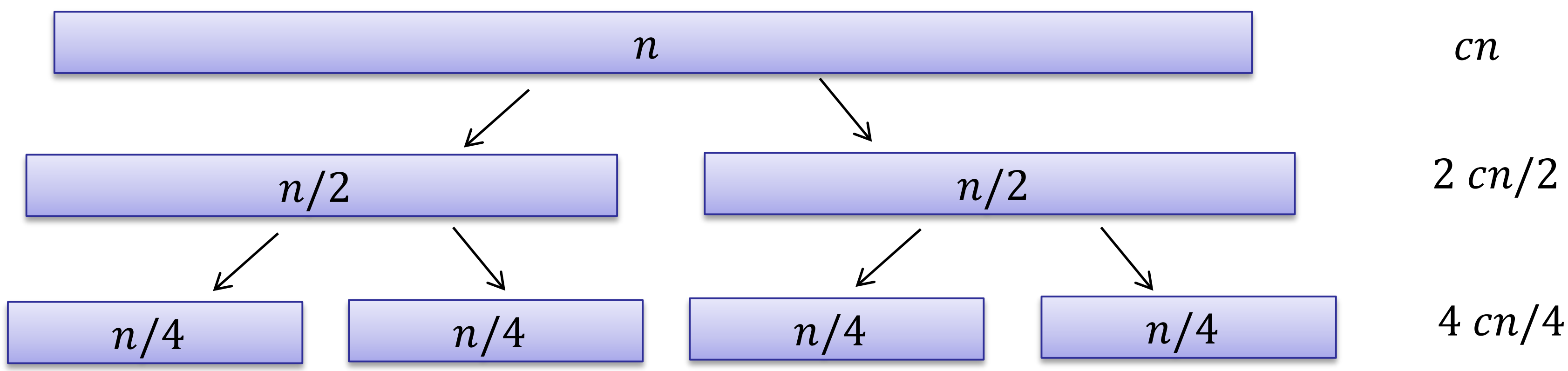
Teile & Herrsche

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



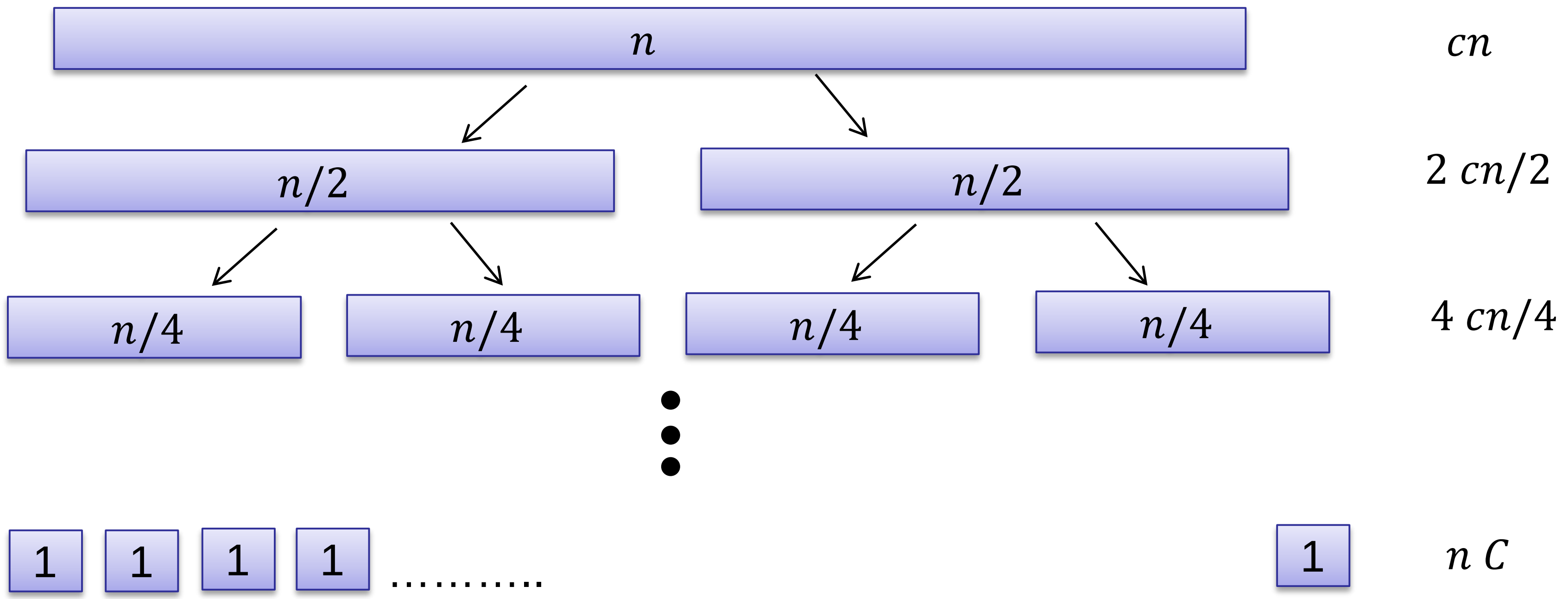
Teile & Herrsche

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



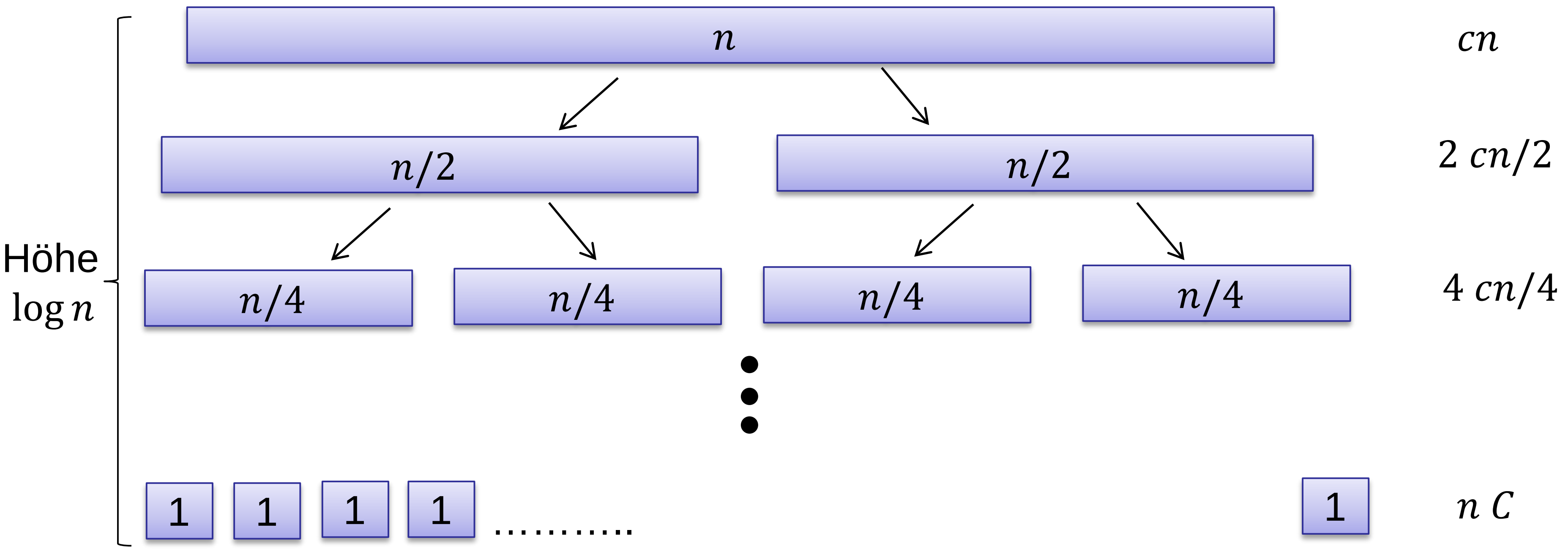
Teile & Herrsche

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



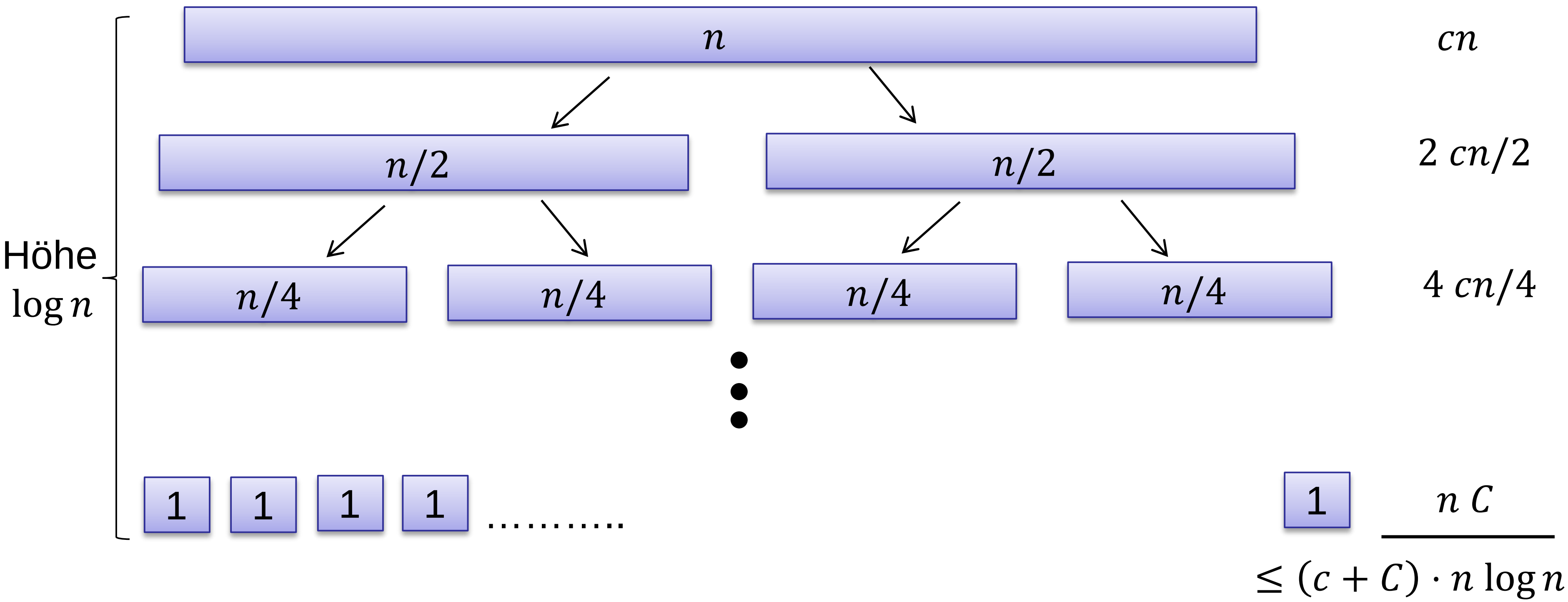
Teile & Herrsche

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



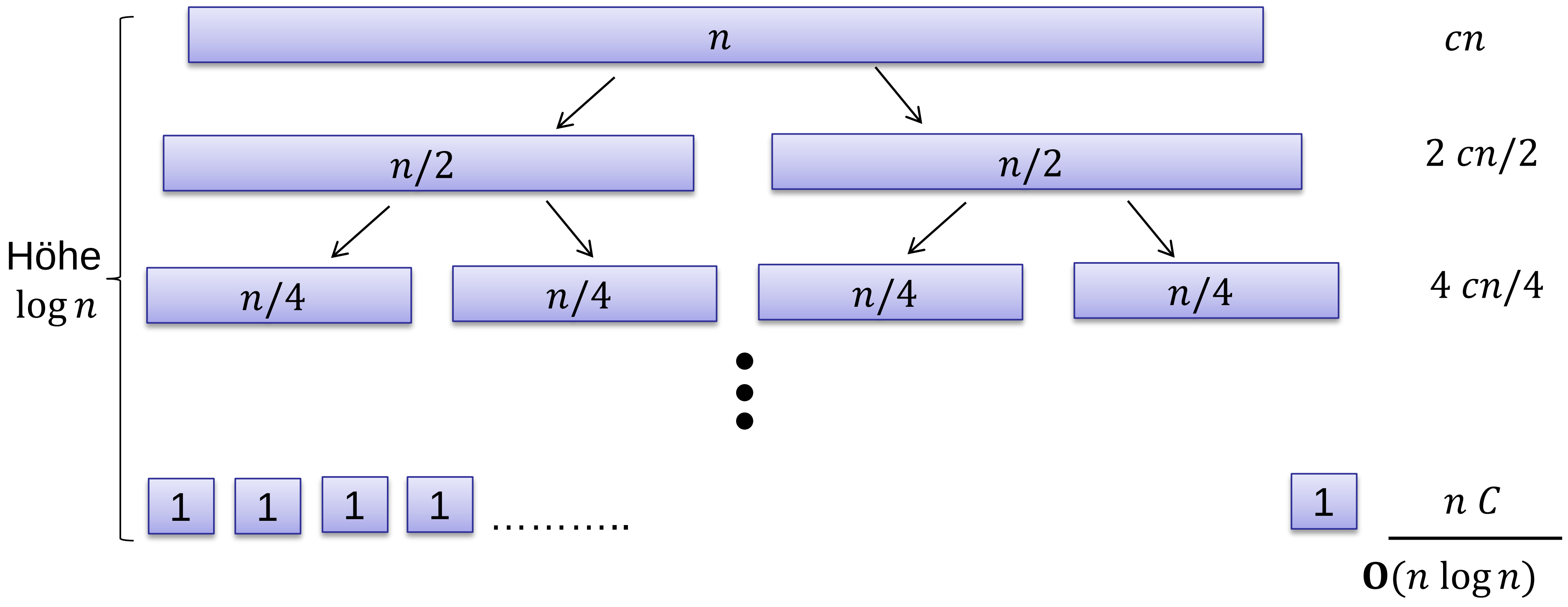
Teile & Herrsche

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



Teile & Herrsche

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



Teile & Herrsche

Satz 5

Algorithmus $\mathrm{MergeSort}$ hat eine Laufzeit von $\mathbf{O}(n \log n)$.

Beweis

- Wir zeigen den Satz nur für den Fall, dass n eine Zweierpotenz ist.

Teile & Herrsche

Satz 5

Algorithmus MergeSort hat eine Laufzeit von $\mathbf{O}(n \log n)$.

Beweis

- Wir zeigen den Satz nur für den Fall, dass n eine Zweierpotenz ist.
- 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 $n \geq 2$

Teile & Herrsche

Satz 5

Algorithmus $\mathrm{MergeSort}$ hat eine Laufzeit von $\mathbf{O}(n \log n)$.

Beweis

- Wir zeigen den Satz nur für den Fall, dass n eine Zweierpotenz ist.
- 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 $T(2) \le C' \le C^* 2 \log 2$.

Teile & Herrsche

Satz 5

Algorithmus MergeSort hat eine Laufzeit von $\mathbf{O}(n \log n)$.

Beweis

- Wir zeigen den Satz nur für den Fall, dass n eine Zweierpotenz ist.
- 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 $m < n, m$ Zweierpotenz, ist die Laufzeit $T(m) \leq C^* m \log m$

Teile & Herrsche

Satz 5

Algorithmus $\mathrm{MergeSort}$ hat eine Laufzeit von $\mathbf{O}(n \log n)$.

Beweis

- Wir zeigen den Satz nur für den Fall, dass n eine Zweierpotenz ist.
- 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 $m < n, m$ Zweierpotenz, ist die Laufzeit $T(m) \le C^* m \log m$

Teile & Herrsche

Satz 5

Algorithmus MergeSort hat eine Laufzeit von $\mathbf{O}(n \log n)$.

Beweis (Fortsetzung)

- (I.S.) Sei n eine Zweierpotenz. Es gilt $T(n) \leq 2\,T(n/2) + cn$.
  Nach (I.V.) gilt $T(n) \leq 2\,C^* \frac{n}{2} \log(n/2) + cn$

Teile & Herrsche

Satz 5

Algorithmus $\mathrm{MergeSort}$ hat eine Laufzeit von $\mathbf{O}(n \log n)$.

Beweis (Fortsetzung)

- (I.S.) Sei n eine Zweierpotenz. Es gilt $T(n) \leq 2\,T(n/2) + cn$.
  Nach (I.V.) gilt $T(n) \leq 2\,C^* \frac{n}{2} \log(n/2) + cn$
  $\leq \quad C^*\, n(\log(n) - 1) + cn$

Teile & Herrsche

Satz 5

Algorithmus $\mathrm{MergeSort}$ hat eine Laufzeit von $\mathbf{O}(n \log n)$.

Beweis (Fortsetzung)

- (I.S.) Sei n eine Zweierpotenz. Es gilt $T(n) \le 2\,T(n/2) + cn$.
  Nach (I.V.) gilt $T(n) \le 2\,C^* \frac{n}{2}\log(n/2) + cn$

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

Teile & Herrsche

Satz 5

Algorithmus $\mathrm{MergeSort}$ hat eine Laufzeit von $\mathbf{O}(n \log n)$.

Beweis (Fortsetzung)

- (I.S.) Sei n eine Zweierpotenz. Es gilt $T(n) \leq 2\, T(n/2) + cn$.
  Nach (I.V.) gilt $T(n) \leq 2\, C^* \frac{n}{2} \log(n/2) + cn$

$$\begin{aligned} &\leq C^*\, n(\log(n) - 1) + cn \\ &\leq C^*\, n(\log(n) - 1) + C^* n \\ &\leq C^*\, n \log(n) \end{aligned}$$

- Also gilt $T(n) = \mathbf{O}(n \log n)$, [da für $n \geq n_0 = 2$, $T(n) \leq C^* n \log n$ ist]

Teile & Herrsche

Satz 5

Algorithmus $\mathrm{MergeSort}$ hat eine Laufzeit von $\mathbf{O}(n \log n)$.

Beweis (Fortsetzung)

- (I.S.) Sei n eine Zweierpotenz. Es gilt $T(n) \le 2\,T(n/2) + cn$.
  Nach (I.V.) gilt $T(n) \le 2\,C^* \frac{n}{2} \log(n/2) + cn$

$$\begin{aligned} &\le C^*\, n(\log(n) - 1) + cn \\ &\le C^*\, n(\log(n) - 1) + C^* n \\ &\le C^*\, n \log(n) \end{aligned}$$

- Also gilt $T(n) = \mathbf{O}(n \log n)$, [da für $n \ge n_0 = 2$, $T(n) \le C^* n \log n$ ist]

Teile & Herrsche

(Falsche) Behauptung

$\mathrm{MergeSort}$ hat Laufzeit $\mathbf{O}(n)$.

(Falscher) Beweis

- Wir zeigen die Aussage nur für den Fall, dass n eine Zweierpotenz ist.
- Die Laufzeit für $T(1)$ und $T(2)$ ist $\mathbf{O}(1)$. Wir zeigen per Induktion, $T(n) = \mathbf{O}(n)$ für alle $n \geq 2$.
- (I.A.) für $n = 2$ gilt $T(2) = \mathbf{O}(1)$.
- (I.V.) Für Eingabelänge $m < n$, m Zweierpotenz, ist die Laufzeit $T(m) = \mathbf{O}(m)$.
- (I.S.) Sei n eine Zweierpotenz. Es gilt $T(n) = 2\,T(n/2) + \mathbf{O}(n)$. Nach (I.V.) gilt
  $$T(n) = 2\,\mathbf{O}(n) + \mathbf{O}(n) = \mathbf{O}(n)$$
- Also gilt $T(n) = \mathbf{O}(n)$.

Teile & Herrsche

> Wo liegt der Fehler?
>
> A) Im Induktionsanfang
> B) In der Induktionsvoraussetzung
> C) Im Induktionsschluss
> D) Der Beweis ist korrekt. Die Behauptung stimmt nicht, wenn n keine Zweierpotenz ist.

(Falsche) Behauptung

$\mathrm{MergeSort}$ hat Laufzeit $\mathbf{O}(n)$.

(Falscher) Beweis

- Wir zeigen die Aussage nur für den Fall, dass n eine Zweierpotenz ist.
- Die Laufzeit für $T(1)$ und $T(2)$ ist $\mathbf{O}(1)$. Wir zeigen per Induktion, $T(n) = \mathbf{O}(n)$ für alle $n \geq 2$.
- (I.A.) für $n = 2$ gilt $T(2) = \mathbf{O}(1)$.
- (I.V.) Für Eingabelänge $m < n$, m Zweierpotenz, ist die Laufzeit $T(m) = \mathbf{O}(m)$.
- (I.S.) Sei n eine Zweierpotenz. Es gilt $T(n) = 2\,T(n/2) + \mathbf{O}(n)$. Nach (I.V.) gilt
  $$T(n) = 2\,\mathbf{O}(n) + \mathbf{O}(n) = \mathbf{O}(n)$$
- Also gilt $T(n) = \mathbf{O}(n)$.

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

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

Wann lohnt sich Teile & Herrsche?

Kann durch Laufzeitanalyse vorhergesagt werden

Teile & Herrsche

Laufzeiten der Form

$$T(n) = a \cdot T(n/b) + f(n)$$

(und $T(1)$ = const)

Teile & Herrsche

Laufzeiten der Form

$$T(n) = a \cdot T(n/b) + f(n)$$

Anzahl Unterprobleme

(und $T(1)$ = const)

Teile & Herrsche

Laufzeiten der Form

$$T(n) = a \cdot T(n/b) + f(n)$$

Anzahl Unterprobleme

Größe der Unterprobleme
(bestimmt Höhe des Rekursionsbaums)

(und $T(1)$ = const)

Teile & Herrsche

Laufzeiten der Form

$$T(n) = a \cdot T(n/b) + f(n)$$

Anzahl Unterprobleme

Aufwand für Aufteilen und
Zusammenfügen

Größe der Unterprobleme
(bestimmt Höhe des Rekursionsbaums)

(und $T(1)$ = const)

Teile & Herrsche

Laufzeiten der Form

$$T(n) = a \cdot T(n/b) + f(n)$$

Anzahl Unterprobleme

Aufwand für Aufteilen und Zusammenfügen

Größe der Unterprobleme
(bestimmt Höhe des Rekursionsbaums)

(und $T(1)$ = const)

Welche unterschiedlichen Fälle gibt es?

Teile & Herrsche

Beispiel $MergeSort$:

$$T(n) = 2 \cdot T(n/2) + cn$$

Anzahl Unterprobleme

Aufwand für Aufteilen und Zusammenfügen

Größe der Unterprobleme
(bestimmt Höhe des Rekursionsbaums)

(und $T(1)$ = const)

(n Zweierpotenz)

Teile & Herrsche

Weiteres Beispiel

- Problem: Finde Element in sortiertem Feld
- Eingabe: Sortiertes Feld A, gesuchtes Element $b \in A[1,\ldots,n]$
- Ausgabe: Index i mit $A[i] = b$

BinäreSuche(A, b, p, r)

1. **if** $p = r$ **then return** p
2. **else**
3. $\quad q \leftarrow \lfloor (p+r)/2 \rfloor$
4. $\quad$ **if** $b \leq A[q]$ **then return** BinäreSuche(A, b, p, q)
5. $\quad$ **else return** BinäreSuche$(A, b, q+1, r)$

Teile & Herrsche

BinäreSuche(A, b, p, r)

1. **if** $p = r$ **then return** p
2. **else**
3. $\quad q \leftarrow \lfloor (p+r)/2 \rfloor$
4. $\quad$ **if** $b \leq A[q]$ **then return** BinäreSuche(A, b, p, q)
5. $\quad$ **else return** BinäreSuche$(A, b, q+1, r)$

Aufruf

BinäreSuche$(A, b, 1, n)$

Teile & Herrsche

BinäreSuche(A, b, p, r)

1. **if** $p = r$ **then return** p
2. **else**
3. $\quad q \leftarrow \lfloor (p+r)/2 \rfloor$
4. $\quad$ **if** $b \le A[q]$ **then return** BinäreSuche(A, b, p, q)
5. $\quad$ **else return** BinäreSuche$(A, b, q+1, r)$

2	**7**	**10**	**11**	**23**	**34**	**47**

Suche $b = 23$

Teile & Herrsche

BinäreSuche(A, b, p, r)

1. **if** $p = r$ **then return** p
2. **else**
3. $\quad q \leftarrow \lfloor (p+r)/2 \rfloor$
4. $\quad$ **if** $b \le A[q]$ **then return** BinäreSuche(A, b, p, q)
5. $\quad$ **else return** BinäreSuche$(A, b, q+1, r)$

2	**7**	**10**	**11**	**23**	**34**	**47**

$p = 1$ $\qquad$ $r = 7$

Suche $b = 23$

Teile & Herrsche

BinäreSuche(A, b, p, r)

1. **if** $p = r$ **then return** p
2. **else**
3. $\boxed{q \leftarrow \lfloor (p + r)/2 \rfloor}$
4. **if** $b \leq A[q]$ **then return** BinäreSuche(A, b, p, q)
5. **else return** BinäreSuche$(A, b, q + 1, r)$

2	**7**	**10**	**11**	**23**	**34**	**47**
$p = 1$			$q = 4$			$r = 7$

Suche $b = 23$

Teile & Herrsche

BinäreSuche(A, b, p, r)

1. **if** $p = r$ **then return** p
2. **else**
3. $\quad q \leftarrow \lfloor (p+r)/2 \rfloor$
4. $\quad$ **if** $b \leq A[q]$ **then return** BinäreSuche(A, b, p, q)
5. $\quad$ **else return** BinäreSuche$(A, b, q+1, r)$

2	**7**	**10**	**11**	**23**	**34**	**47**
				$p = 5$		$r = 7$

Suche $b = 23$

Teile & Herrsche

$\text{BinäreSuche}(A, b, p, r)$

1. **if** $p = r$ **then return** p
2. **else**
3. $\boxed{q \leftarrow \lfloor (p + r)/2 \rfloor}$
4. **if** $b \leq A[q]$ **then return** $\text{BinäreSuche}(A, b, p, q)$
5. **else return** $\text{BinäreSuche}(A, b, q + 1, r)$

2	**7**	**10**	**11**	**23**	**34**	**47**

$p = 5$ $q = 6$ $r = 7$

Suche $b = 23$

Teile & Herrsche

BinäreSuche(A, b, p, r)

1. **if** $p = r$ **then return** p
2. **else**
3. $\quad q \leftarrow \lfloor (p+r)/2 \rfloor$
4. $\quad$ **if** $b \leq A[q]$ **then return** BinäreSuche(A, b, p, q)
5. $\quad$ **else return** BinäreSuche$(A, b, q+1, r)$

2	**7**	**10**	**11**	**23**	**34**	**47**

$p = 5$ $r = 6$

Suche $b = 23$

Teile & Herrsche

BinäreSuche(A, b, p, r)

1. **if** $p = r$ **then return** p
2. **else**
3. $\boxed{q \leftarrow \lfloor (p + r)/2 \rfloor}$
4. **if** $b \leq A[q]$ **then return** BinäreSuche(A, b, p, q)
5. **else return** BinäreSuche$(A, b, q + 1, r)$

2	**7**	**10**	**11**	**23**	**34**	**47**

$p = 5$ $r = 6$

$q = 5$

Suche $b = 23$

Teile & Herrsche

BinäreSuche(A, b, p, r)
1. **if** $p = r$ **then return** p
2. **else**
3. $\quad q \leftarrow \lfloor (p+r)/2 \rfloor$
4. $\quad$ **if** $b \leq A[q]$ **then return** BinäreSuche(A, b, p, q)
5. $\quad$ **else return** BinäreSuche$(A, b, q+1, r)$

2	**7**	**10**	**11**	**23**	**34**	**47**

$p = 5$
$r = 5$

Suche $b = 23$

Teile & Herrsche

BinäreSuche(A, b, p, r)
1. $\boxed{\textbf{if } p = r \textbf{ then return } p}$
2. **else**
3. $\quad q \leftarrow \lfloor (p+r)/2 \rfloor$
4. $\quad$ **if** $b \le A[q]$ **then return** BinäreSuche(A, b, p, q)
5. $\quad$ **else return** BinäreSuche$(A, b, q+1, r)$

2	**7**	**10**	**11**	**23**	**34**	**47**

$p = 5$

$r = 5$

Suche $b = 23$

Teile & Herrsche

Satz 6

Algorithmus $\text{BinäreSuche}(A, b, p, r)$ findet den Index einer Zahl b in einem sortierten Feld $A[p..r]$, sofern b in $A[p..r]$ vorhanden ist.

Beweis

- Wir zeigen die Korrektheit per Induktion über $n = r - p$. Ist $n < 0$, so ist nichts zu zeigen. Wir nehmen an, dass b in $A[p..r]$ ist, da es sonst nichts zu zeigen gibt.

Teile & Herrsche

Satz 6

Algorithmus $\mathrm{BinäreSuche}(A, b, p, r)$ findet den Index einer Zahl b in einem sortierten Feld $A[p..r]$, sofern b in $A[p..r]$ vorhanden ist.

Beweis

- Wir zeigen die Korrektheit per Induktion über $n = r - p$. Ist $n < 0$, so ist nichts zu zeigen. Wir nehmen an, dass b in $A[p..r]$ ist, da es sonst nichts zu zeigen gibt.
- (I.A.) Für $n = 0$, d.h. $p = r$, gibt der Algorithmus p zurück. Dies ist der korrekte (weil einzige) Index.

Teile & Herrsche

Satz 6

Algorithmus $\mathrm{BinäreSuche}(A, b, p, r)$ findet den Index einer Zahl b in einem sortierten Feld $A[p..r]$, sofern b in $A[p..r]$ vorhanden ist.

Beweis

- Wir zeigen die Korrektheit per Induktion über $n = r - p$. Ist $n < 0$, so ist nichts zu zeigen. Wir nehmen an, dass b in $A[p..r]$ ist, da es sonst nichts zu zeigen gibt.
- (I.A.) Für $n = 0$, d.h. $p = r$, gibt der Algorithmus p zurück. Dies ist der korrekte (weil einzige) Index.
- (I.V.) Für alle r, p mit $m = r - p$ und $0 \leq m \leq n$ findet $\mathrm{BinäreSuche}(A, b, p, r)$ den Index einer Zahl b in einem sortierten Feld $A[p..r]$, sofern b im Feld vorhanden ist.

Teile & Herrsche

Satz 6

Algorithmus $\mathrm{BinäreSuche}(A, b, p, r)$ findet den Index einer Zahl b in einem sortierten Feld $A[p..r]$, sofern b in $A[p..r]$ vorhanden ist.

Beweis

- (I.V.) Für alle r, p mit $m = r - p$ und $0 \leq m \leq n$ findet $\mathrm{BinäreSuche}(A, b, p, r)$ den Index einer Zahl b in einem sortierten Feld $A[p..r]$, sofern b im Feld vorhanden ist.

Teile & Herrsche

Satz 6

Algorithmus $\text{BinäreSuche}(A, b, p, r)$ findet den Index einer Zahl b in einem sortierten Feld $A[p..r]$, sofern b in $A[p..r]$ vorhanden ist.

Beweis

- (I.V.) Für alle r, p mit $m = r - p$ und $0 \leq m \leq n$ findet $\text{BinäreSuche}(A, b, p, r)$ den Index einer Zahl b in einem sortierten Feld $A[p..r]$, sofern b im Feld vorhanden ist.
- (I.S.) Wir betrachten den Aufruf von BinäreSuche für beliebige p, r mit $n + 1 = r - p$. Da $n + 1 > 0$ folgt $p < r$ und der Algorithmus führt den **else**-Fall aus. Dort wird q auf $\lfloor (p + r)/2 \rfloor$ gesetzt.

Teile & Herrsche

Satz 6

Algorithmus $\mathrm{BinäreSuche}(A, b, p, r)$ findet den Index einer Zahl b in einem sortierten Feld $A[p..r]$, sofern b in $A[p..r]$ vorhanden ist.

Beweis

- (I.V.) Für alle r, p mit $m = r - p$ und $0 \leq m \leq n$ findet $\mathrm{BinäreSuche}(A, b, p, r)$ den Index einer Zahl b in einem sortierten Feld $A[p..r]$, sofern b im Feld vorhanden ist.
- (I.S.) Wir betrachten den Aufruf von $\mathrm{BinäreSuche}$ für beliebige p, r mit $n + 1 = r - p$. Da $n + 1 > 0$ folgt $p < r$ und der Algorithmus führt den **else**-Fall aus. Dort wird q auf $\lfloor (p + r)/2 \rfloor$ gesetzt. Es gilt $q \geq p$ und $q < r$. Ist $b \leq A[q]$, so wird $\mathrm{BinäreSuche}$ rekursiv für $A[p..q]$ aufgerufen.

Teile & Herrsche

Satz 6

Algorithmus $\mathrm{BinäreSuche}(A, b, p, r)$ findet den Index einer Zahl b in einem sortierten Feld $A[p..r]$, sofern b in $A[p..r]$ vorhanden ist.

Beweis

- (I.V.) Für alle r, p mit $m = r - p$ und $0 \leq m \leq n$ findet $\mathrm{BinäreSuche}(A, b, p, r)$ den Index einer Zahl b in einem sortierten Feld $A[p..r]$, sofern b im Feld vorhanden ist.
- (I.S.) Wir betrachten den Aufruf von $\mathrm{BinäreSuche}$ für beliebige p, r mit $n + 1 = r - p$. Da $n + 1 > 0$ folgt $p < r$ und der Algorithmus führt den **else**-Fall aus. Dort wird q auf $\lfloor (p + r)/2 \rfloor$ gesetzt. Es gilt $q \geq p$ und $q < r$. Ist $b \leq A[q]$, so wird $\mathrm{BinäreSuche}$ rekursiv für $A[p..q]$ aufgerufen. Da $A[p..r]$ sortiert ist, liegt b in $A[p..q]$. Damit folgt aus (I.V.), dass der Index von b gefunden wird.

Teile & Herrsche

Satz 6

Algorithmus $\mathrm{BinäreSuche}(A, b, p, r)$ findet den Index einer Zahl b in einem sortierten Feld $A[p..r]$, sofern b in $A[p..r]$ vorhanden ist.

Beweis

- (I.V.) Für alle r, p mit $m = r - p$ und $0 \leq m \leq n$ findet $\mathrm{BinäreSuche}(A, b, p, r)$ den Index einer Zahl b in einem sortierten Feld $A[p..r]$, sofern b im Feld vorhanden ist.
- (I.S.) Wir betrachten den Aufruf von $\mathrm{BinäreSuche}$ für beliebige p, r mit $n + 1 = r - p$. Da $n + 1 > 0$ folgt $p < r$ und der Algorithmus führt den **else**-Fall aus. Dort wird q auf $\lfloor (p + r)/2 \rfloor$ gesetzt. Es gilt $q \geq p$ und $q < r$. Ist $b \leq A[q]$, so wird $\mathrm{BinäreSuche}$ rekursiv für $A[p..q]$ aufgerufen. Da $A[p..r]$ sortiert ist, liegt b in $A[p..q]$. Damit folgt aus (I.V.), dass der Index von b gefunden wird. Ist $b > A[q]$, so wird $\mathrm{BinäreSuche}$ rekursiv für $A[q + 1..r]$ aufgerufen. Da $A[p..r]$ sortiert ist, liegt b in $A[q + 1..r]$.

Teile & Herrsche

Satz 6

Algorithmus $\mathrm{BinäreSuche}(A, b, p, r)$ findet den Index einer Zahl b in einem sortierten Feld $A[p..r]$, sofern b in $A[p..r]$ vorhanden ist.

Beweis

- (I.V.) Für alle r, p mit $m = r - p$ und $0 \leq m \leq n$ findet $\mathrm{BinäreSuche}(A, b, p, r)$ den Index einer Zahl b in einem sortierten Feld $A[p..r]$, sofern b im Feld vorhanden ist.
- (I.S.) Wir betrachten den Aufruf von $\mathrm{BinäreSuche}$ für beliebige p, r mit $n + 1 = r - p$. Da $n + 1 > 0$ folgt $p < r$ und der Algorithmus führt den **else**-Fall aus. Dort wird q auf $\lfloor (p + r)/2 \rfloor$ gesetzt. Es gilt $q \geq p$ und $q < r$. Ist $b \leq A[q]$, so wird $\mathrm{BinäreSuche}$ rekursiv für $A[p..q]$ aufgerufen. Da $A[p..r]$ sortiert ist, liegt b in $A[p..q]$. Damit folgt aus (I.V.), dass der Index von b gefunden wird. Ist $b > A[q]$, so wird $\mathrm{BinäreSuche}$ rekursiv für $A[q + 1..r]$ aufgerufen. Da $A[p..r]$ sortiert ist, liegt b in $A[q + 1..r]$. Damit folgt aus (I.V.), dass der Index von b gefunden wird.

Teile & Herrsche

Satz 6

Algorithmus $\mathrm{BinäreSuche}(A, b, p, r)$ findet den Index einer Zahl b in einem sortierten Feld $A[p..r]$, sofern b in $A[p..r]$ vorhanden ist.

Beweis

- (I.V.) Für alle r, p mit $m = r - p$ und $0 \leq m \leq n$ findet $\mathrm{BinäreSuche}(A, b, p, r)$ den Index einer Zahl b in einem sortierten Feld $A[p..r]$, sofern b im Feld vorhanden ist.
- (I.S.) Wir betrachten den Aufruf von $\mathrm{BinäreSuche}$ für beliebige p, r mit $n + 1 = r - p$. Da $n + 1 > 0$ folgt $p < r$ und der Algorithmus führt den **else**-Fall aus. Dort wird q auf $\lfloor (p + r)/2 \rfloor$ gesetzt. Es gilt $q \geq p$ und $q < r$. Ist $b \leq A[q]$, so wird $\mathrm{BinäreSuche}$ rekursiv für $A[p..q]$ aufgerufen. Da $A[p..r]$ sortiert ist, liegt b in $A[p..q]$. Damit folgt aus (I.V.), dass der Index von b gefunden wird. Ist $b > A[q]$, so wird $\mathrm{BinäreSuche}$ rekursiv für $A[q + 1..r]$ aufgerufen. Da $A[p..r]$ sortiert ist, liegt b in $A[q + 1..r]$. Damit folgt aus (I.V.), dass der Index von b gefunden wird.

Teile & Herrsche

BinäreSuche(A, b, p, r)

1. **if** $p = r$ **then return** p
2. **else**
3. $\quad q \leftarrow \lfloor (p + r)/2 \rfloor$
4. $\quad$ **if** $b \le A[q]$ **then return** BinäreSuche(A, b, p, q)
5. $\quad$ **else return** BinäreSuche$(A, b, q + 1, r)$

Laufzeit:

Laufzeit

$T(n)$, wobei $n = r - p + 1$ ist

Teile & Herrsche

BinäreSuche(A, b, p, r)
1. **if** $p = r$ **then return** p
2. **else**
3. $\quad q \leftarrow \lfloor (p+r)/2 \rfloor$
4. $\quad$ **if** $b \leq A[q]$ **then return** BinäreSuche(A, b, p, q)
5. $\quad$ **else return** BinäreSuche$(A, b, q+1, r)$

Laufzeit:

1

Laufzeit

$T(n)$, wobei $n = r - p + 1$ ist

Teile & Herrsche

$\text{BinäreSuche}(A, b, p, r)$

1. **if** $p = r$ **then return** p
2. **else**
3. $\quad q \leftarrow \lfloor (p + r)/2 \rfloor$
4. $\quad$ **if** $b \leq A[q]$ **then return** $\text{BinäreSuche}(A, b, p, q)$
5. $\quad$ **else return** $\text{BinäreSuche}(A, b, q + 1, r)$

Laufzeit:

1

1

Laufzeit

$T(n)$, wobei $n = r - p + 1$ ist

Teile & Herrsche

	Laufzeit:
BinäreSuche(A,b,p,r)	
1. **if** $p=r$ **then return** p	1
2. **else**	1
3. $\quad q \leftarrow \lfloor (p+r)/2 \rfloor$	1
4. $\quad$ **if** $b \le A[q]$ **then return** BinäreSuche(A,b,p,q)	
5. $\quad$ **else return** BinäreSuche$(A,b,q+1,r)$	

Laufzeit

$T(n)$, wobei $n = r - p + 1$ ist

Teile & Herrsche

BinäreSuche(A, b, p, r)	Laufzeit:
1. **if** $p = r$ **then return** p	1
2. **else**	1
3. $\quad q \leftarrow \lfloor (p+r)/2 \rfloor$	1
4. $\quad$ **if** $b \leq A[q]$ **then return** BinäreSuche(A, b, p, q)	$1 + T(\lceil n/2 \rceil)$
5. $\quad$ **else return** BinäreSuche$(A, b, q+1, r)$	

Laufzeit

$T(n)$, wobei $n = r - p + 1$ ist

Teile & Herrsche

	Laufzeit:
BinäreSuche(A, b, p, r)	
1. **if** $p = r$ **then return** p	1
2. **else**	1
3. $\quad q \leftarrow \lfloor (p+r)/2 \rfloor$	1
4. $\quad$ **if** $b \leq A[q]$ **then return** BinäreSuche(A, b, p, q)	$1 + T(\lceil n/2 \rceil)$
5. $\quad$ **else return** BinäreSuche$(A, b, q+1, r)$	$1 + T(\lfloor n/2 \rfloor)$

Laufzeit

$T(n)$, wobei $n = r - p + 1$ ist

Teile & Herrsche

BinäreSuche(A,b,p,r)	Laufzeit:
1. **if** $p = r$ **then return** p	1
2. **else**	1
3. $\quad q \leftarrow \lfloor (p+r)/2 \rfloor$	1
4. $\quad$ **if** $b \leq A[q]$ **then return** BinäreSuche(A,b,p,q)	$1 + T(\lceil n/2 \rceil)$
5. $\quad$ **else return** BinäreSuche$(A,b,q+1,r)$	$1 + T(\lfloor n/2 \rfloor)$

$$5 + \max\{T(\lceil n/2 \rceil), T(\lfloor n/2 \rfloor)\}$$

Laufzeit

$$T(n) = \begin{cases} 1 & \text{, falls } n = 1 \\ 5 + \max\{T(\lceil n/2 \rceil), T(\lfloor n/2 \rfloor)\} & \text{, falls } n > 1 \end{cases}$$

Teile & Herrsche

Beispiel $BinäreSuche$:

$$T(n) = 1 \cdot T(n/2) + c$$

Anzahl Unterprobleme

Aufwand für Aufteilen und Zusammenfügen

Größe der Unterprobleme
(bestimmt Höhe des Rekursionsbaums)

(und $T(1)$ = const)

(n Zweierpotenz)

Teile & Herrsche

Auflösen von $T(n) \leq T(n/2) + c$ (Intuition; wir ignorieren Runden)

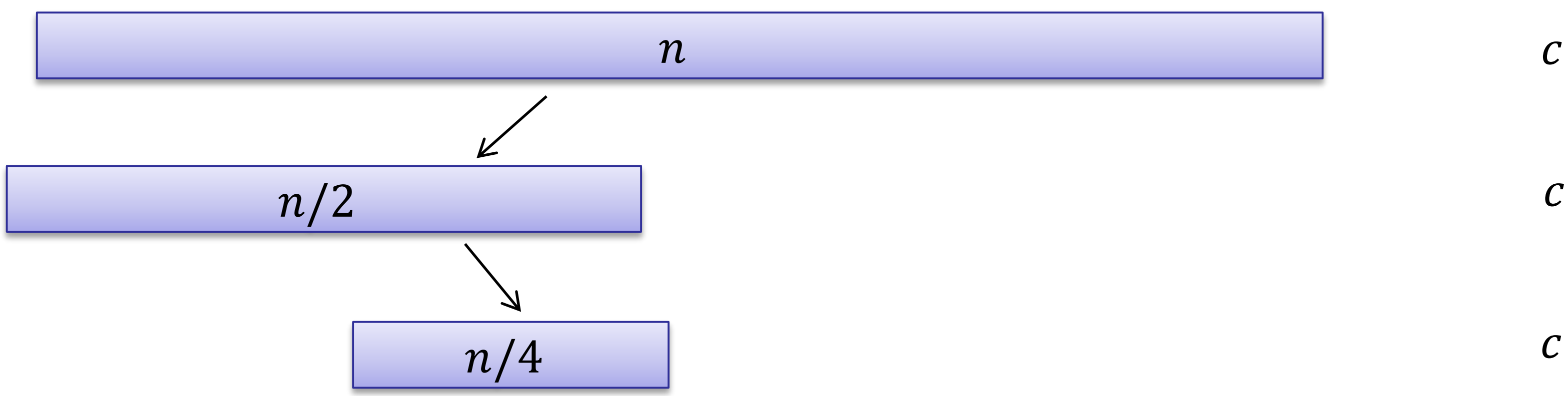
n $\qquad$ c

Teile & Herrsche

Auflösen von $T(n) \leq T(n/2) + c$ (Intuition; wir ignorieren Runden)

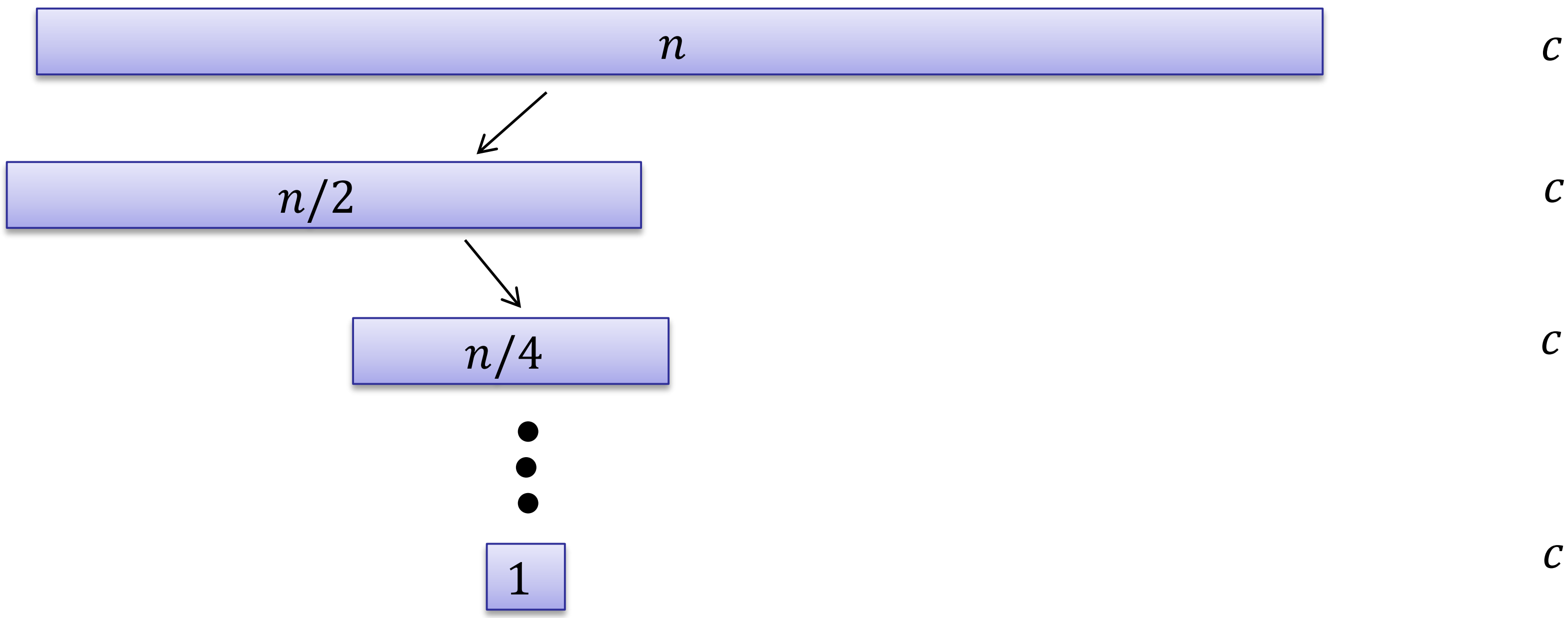
Teile & Herrsche

Auflösen von $T(n) \leq T(n/2) + c$ (Intuition; wir ignorieren Runden)



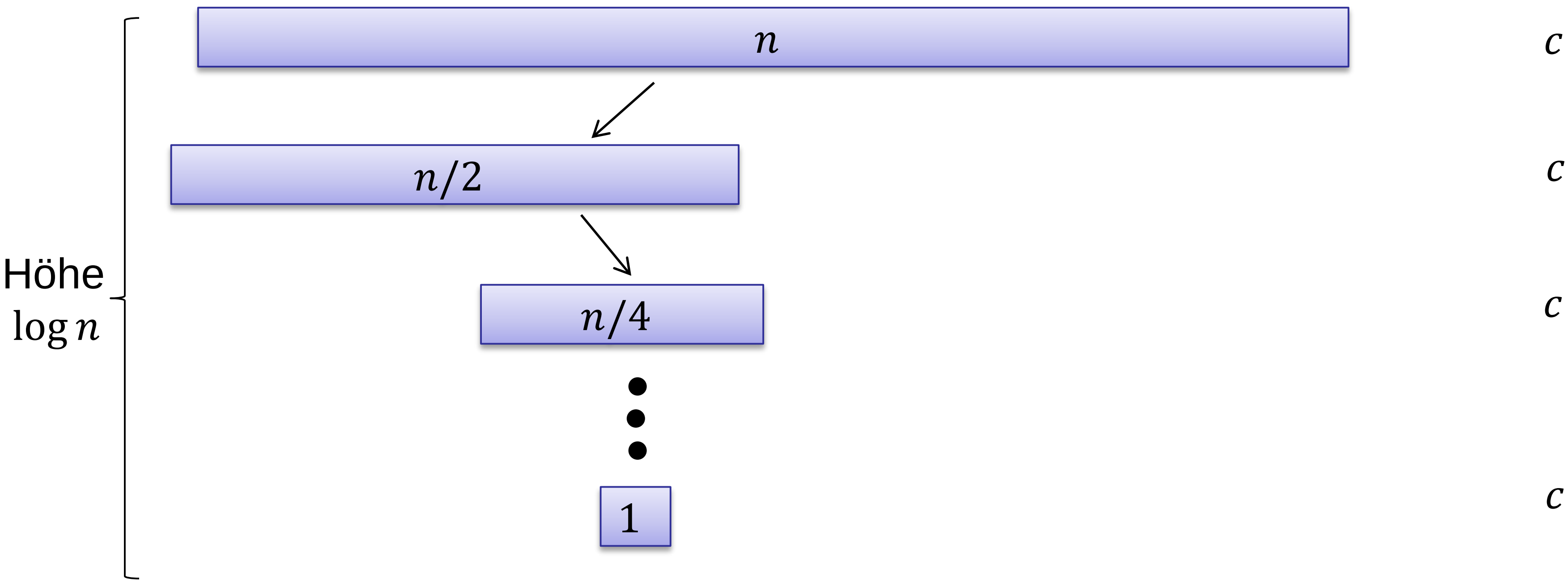
Teile & Herrsche

Auflösen von $T(n) \leq T(n/2) + c$ (Intuition; wir ignorieren Runden)



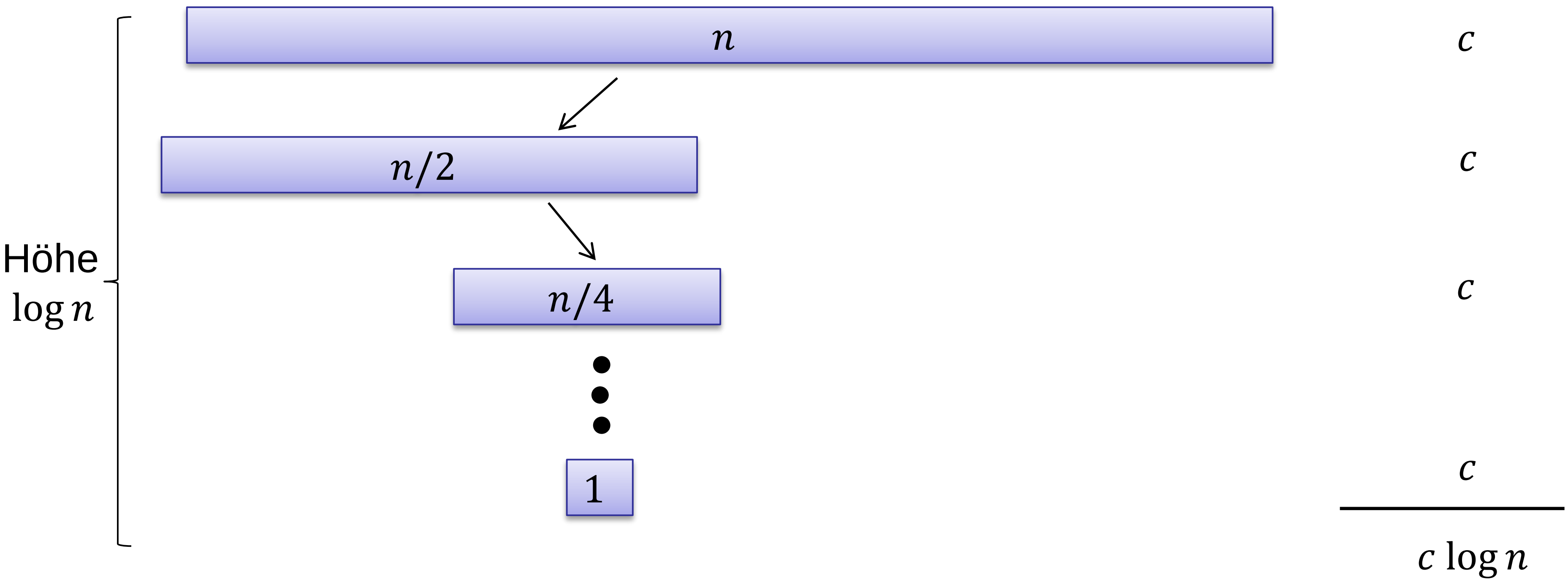
Teile & Herrsche

Auflösen von $T(n) \leq T(n/2) + c$ (Intuition; wir ignorieren Runden)



Teile & Herrsche

Auflösen von $T(n) \leq T(n/2) + c$ (Intuition; wir ignorieren Runden)



Teile & Herrsche

Satz 7

Algorithmus $\mathrm{BinäreSuche}$ hat eine Laufzeit von $\mathbf{O}(\log n)$.

Beweis

- Wir zeigen per Induktion, $T(n) \leq 5 \lceil \log n \rceil + 1$.

Teile & Herrsche

Satz 7

Algorithmus $\mathrm{BinäreSuche}$ hat eine Laufzeit von $\mathbf{O}(\log n)$.

Beweis

- Wir zeigen per Induktion, $T(n) \leq 5\lceil \log n \rceil + 1$.
- (I.A.) für $n = 1$ gilt $T(1) = 1 = 5\lceil \log n \rceil + 1$.

Teile & Herrsche

Satz 7

Algorithmus $\mathrm{BinäreSuche}$ hat eine Laufzeit von $\mathbf{O}(\log n)$.

Beweis

- Wir zeigen per Induktion, $T(n) \leq 5 \lceil \log n \rceil + 1$.
- (I.A.) für $n = 1$ gilt $T(1) = 1 = 5 \lceil \log n \rceil + 1$.
- (I.V.) Für Eingabelänge $m < n$ ist die Laufzeit $T(m) \leq 5\lceil \log m \rceil + 1$.

Teile & Herrsche

Satz 7

Algorithmus $\mathrm{BinäreSuche}$ hat eine Laufzeit von $\mathbf{O}(\log n)$.

Beweis

- Wir zeigen per Induktion, $T(n) \leq 5\,\lceil \log n \rceil + 1$.
- (I.A.) für $n = 1$ gilt $T(1) = 1 = 5\,\lceil \log n \rceil + 1$.
- (I.V.) Für Eingabelänge $m < n$ ist die Laufzeit $T(m) \leq 5\lceil \log m \rceil + 1$.
- (I.S.) Wir wissen: $T(n) \leq \max\{T(\lceil n/2 \rceil), T(\lfloor n/2 \rfloor)\} + 5$. Nach (I.V.) gilt somit: $T(n) \leq \max\{5\lceil \log(\lceil n/2 \rceil) \rceil, 5\lceil \log(\lfloor n/2 \rfloor) \rceil\} + 1 + 5 \leq 5\lceil \log(\lceil n/2 \rceil) \rceil + 1 + 5$

Teile & Herrsche

Satz 7

Algorithmus $\mathrm{BinäreSuche}$ hat eine Laufzeit von $\mathbf{O}(\log n)$.

Beweis

- Wir zeigen per Induktion, $T(n) \leq 5\,\lceil \log n \rceil + 1$.
- (I.A.) für $n = 1$ gilt $T(1) = 1 = 5\,\lceil \log n \rceil + 1$.
- (I.V.) Für Eingabelänge $m < n$ ist die Laufzeit $T(m) \leq 5\lceil \log m \rceil + 1$.
- (I.S.) Wir wissen: $T(n) \leq \max\{T(\lceil n/2 \rceil), T(\lfloor n/2 \rfloor)\} + 5$. Nach (I.V.) gilt somit:
  $T(n) \leq \max\{5\lceil \log(\lceil n/2 \rceil)\rceil, 5\lceil \log(\lfloor n/2 \rfloor)\rceil\} + 1 + 5 \leq 5\lceil \log(\lceil n/2 \rceil)\rceil + 1 + 5$
  Ist n gerade, so gilt $\lceil \log(\lceil n/2 \rceil)\rceil = \lceil \log(n/2)\rceil = \lceil \log(n) - 1\rceil = \lceil \log(n)\rceil - 1$.
  Somit folgt $T(n) \leq 5\lceil \log n \rceil + 1$.

Teile & Herrsche

Satz 7

Algorithmus $\mathrm{BinäreSuche}$ hat eine Laufzeit von $\mathbf{O}(\log n)$.

Beweis

- Wir zeigen per Induktion, $T(n) \leq 5\lceil \log n \rceil + 1$.
- (I.A.) für $n = 1$ gilt $T(1) = 1 = 5\lceil \log n \rceil + 1$.
- (I.V.) Für Eingabelänge $m < n$ ist die Laufzeit $T(m) \leq 5\lceil \log m \rceil + 1$.
- (I.S.) Wir wissen: $T(n) \leq \max\{T(\lceil n/2 \rceil), T(\lfloor n/2 \rfloor)\} + 5$. Nach (I.V.) gilt somit: $T(n) \leq \max\{5\lceil \log(\lceil n/2 \rceil) \rceil, 5\lceil \log(\lfloor n/2 \rfloor) \rceil\} + 1 + 5 \leq 5\lceil \log(\lceil n/2 \rceil) \rceil + 1 + 5$
  Ist n gerade, so gilt $\lceil \log(\lceil n/2 \rceil) \rceil = \lceil \log(n/2) \rceil = \lceil \log(n) - 1 \rceil = \lceil \log(n) \rceil - 1$.
  Somit folgt $T(n) \leq 5\lceil \log n \rceil + 1$. Ist n ungerade, so gilt $\lceil \log(\lceil n/2 \rceil) \rceil = \lceil \log((n+1)/2) \rceil = \lceil \log(n+1) - 1 \rceil = \lceil \log(n+1) \rceil - 1 = \lceil \log(n) \rceil - 1$.

Teile & Herrsche

Satz 7

Algorithmus $\mathrm{BinäreSuche}$ hat eine Laufzeit von $\mathbf{O}(\log n)$.

Beweis

- Wir zeigen per Induktion, $T(n) \leq 5\,\lceil \log n \rceil + 1$.
- (I.A.) für $n = 1$ gilt $T(1) = 1 = 5\,\lceil \log n \rceil + 1$.
- (I.V.) Für Eingabelänge $m < n$ ist die Laufzeit $T(m) \leq 5\lceil \log m \rceil + 1$.
- (I.S.) Wir wissen: $T(n) \leq \max\{T(\lceil n/2 \rceil), T(\lfloor n/2 \rfloor)\} + 5$. Nach (I.V.) gilt somit: $T(n) \leq \max\{5\lceil \log(\lceil n/2 \rceil) \rceil, 5\lceil \log(\lfloor n/2 \rfloor) \rceil\} + 1 + 5 \leq 5\lceil \log(\lceil n/2 \rceil) \rceil + 1 + 5$ Ist n gerade, so gilt $\lceil \log(\lceil n/2 \rceil) \rceil = \lceil \log(n/2) \rceil = \lceil \log(n) - 1 \rceil = \lceil \log(n) \rceil - 1$. Somit folgt $T(n) \leq 5\lceil \log n \rceil + 1$. Ist n ungerade, so gilt $\lceil \log(\lceil n/2 \rceil) \rceil = \lceil \log((n+1)/2) \rceil = \lceil \log(n+1) - 1 \rceil = \lceil \log(n+1) \rceil - 1 \;= \lceil \log(n) \rceil - 1$.
- Somit folgt auch hier $T(n) \leq 5\,\lceil \log n \rceil + 1$.

Teile & Herrsche

Satz 7

Algorithmus BinäreSuche hat eine Laufzeit von $\mathbf{O}(\log n)$.

Beweis

- Wir zeigen per Induktion, $T(n) \leq 5\,\lceil \log n \rceil + 1$.
- (I.A.) für $n = 1$ gilt $T(1) = 1 = 5\,\lceil \log n \rceil + 1$.
- (I.V.) Für Eingabelänge $m < n$ ist die Laufzeit $T(m) \leq 5\lceil \log m \rceil + 1$.
- (I.S.) Wir wissen: $T(n) \leq \max\{T(\lceil n/2 \rceil), T(\lfloor n/2 \rfloor)\} + 5$. Nach (I.V.) gilt somit:
  $T(n) \leq \max\{5\lceil \log(\lceil n/2 \rceil)\rceil, 5\lceil \log(\lfloor n/2 \rfloor)\rceil\} + 1 + 5 \leq 5\lceil \log(\lceil n/2 \rceil)\rceil + 1 + 5$
  Ist n gerade, so gilt $\lceil \log(\lceil n/2 \rceil)\rceil = \lceil \log(n/2)\rceil = \lceil \log(n) - 1\rceil = \lceil \log(n)\rceil - 1$.
  Somit folgt $T(n) \leq 5\lceil \log n \rceil + 1$. Ist n ungerade, so gilt $\lceil \log(\lceil n/2 \rceil)\rceil =$
  $\lceil \log((n+1)/2)\rceil = \lceil \log(n+1) - 1\rceil = \lceil \log(n+1)\rceil - 1 \;= \lceil \log(n)\rceil - 1$.
- Somit folgt auch hier $T(n) \leq 5\,\lceil \log n \rceil + 1$.

Teile & Herrsche

Binäre Suche vs. lineare Suche

Laufzeit	**10**	**100**	**1,000**	**10,000**	**100,000**
n	10	100	1,000	10,000	100,000
$\log n$	3	6	10	13	17

Beobachtung

- n wächst sehr viel stärker als $\log n$
- Binäre Suche effizient für riesige Datenmengen
- In der Praxis ist $\log n$ fast wie eine Konstante

Weiteres Beispiel: Zähle Auftreten eines Schlüssel

Count(Array A, Key k)

$c \leftarrow 0$
for $j \leftarrow 1$ **to** length$[A]$ **do**
    if $A[j] = k$ **then** $c \leftarrow c + 1$
return c

CountRec(Array A, Key k, Int n)

If $n = 1$ **then** $c \leftarrow 0$
else $c \leftarrow$ CountRec$(A, k, n-1)$
If $A[n] = k$ **then** $c \leftarrow c + 1$
return c

CountRec(Array A, Key k, Int p, r)

If $p = r$ **then**
    If $A[p] = k$ **then** $c \leftarrow 1$
    else $c \leftarrow 0$
else
    $q \leftarrow \lfloor (p+r)/2 \rfloor$
    $c \leftarrow$ CountRec(A, k, p, q)
    $c \leftarrow c +$ CountRec$(A, k, q+1, r)$
return c

Was sind die Laufzeiten/Rekurrenzen dieser Algorithmen?