



## Datenstrukturen, Algorithmen und Programmierung 2 (DAP2)

## Laufzeitanalyse

### *O-Notation*

- $O(f(n)) = \{g(n) : \exists c > 0, n_0 > 0, \text{ so dass für alle } n \geq n_0 \text{ gilt } g(n) \leq c \cdot f(n)\}$
- (wobei  $f(n), g(n) > 0$ )

### *Interpretation*

- $g(n) \in O(f(n))$  bedeutet, dass  $g(n)$  für  $n \rightarrow \infty$  höchstens genauso stark wächst wie  $f(n)$
- Beim Wachstum ignorieren wir Konstanten

## Laufzeitanalyse

### *Beispiele*

- $10n \in O(n)$
- $10n \in O(n^2)$
- $n^2 \notin O(1000n)$
- $O(1000n) = O(n)$

### *Hierarchie*

- $O(\log n) \subseteq O(\log^2 n) \subseteq O(\log^c n) \subseteq O(n^\varepsilon) \subseteq O(\sqrt{n}) \subseteq O(n)$
- $O(n) \subseteq O(n^2) \subseteq O(n^c) \subseteq O(2^n)$
- (für  $c \geq 2$  und  $0 < \varepsilon \leq 1/2$ )

## Laufzeitanalyse

### *$\Omega$ -Notation*

- $\Omega(f(n)) = \{g(n) : \exists c > 0, n_0 > 0, \text{ so dass für alle } n \geq n_0 \text{ gilt } g(n) \geq c \cdot f(n)\}$
- (wobei  $f(n), g(n) > 0$ )

### *Interpretation*

- $g(n) \in \Omega(f(n))$  bedeutet, dass  $g(n)$  für  $n \rightarrow \infty$  **mindestens** so stark wächst wie  $f(n)$
- Beim Wachstum ignorieren wir Konstanten

## Laufzeitanalyse

### *Beispiele*

- $10n \in \Omega(n)$
- $1000n \notin \Omega(n^2)$
- $n^2 \in \Omega(n)$
- $\Omega(1000n) = \Omega(n)$
- $f(n) = \Omega(g(n)) \Leftrightarrow g(n) = O(f(n))$

## Laufzeitanalyse

### *$\Theta$ -Notation*

- $g(n) \in \Theta(f(n)) \Leftrightarrow g(n) = O(f(n))$  und  $g(n) = \Omega(f(n))$

### *Beispiele*

- $1000n \in \Theta(n)$
- $10n^2 + 1000n \in \Theta(n^2)$
- $n^{1-\sin n} \notin \Theta(n)$

## Laufzeitanalyse

### *o-Notation*

- $o(f(n)) \in \{g(n): \forall c > 0 \exists n_0 > 0, \text{ so dass für alle } n \geq n_0 \text{ gilt } c \cdot g(n) < f(n)\}$
- $(f(n), g(n) > 0)$

### *$\omega$ -Notation*

- $f(n) \in \omega(g(n)) \Leftrightarrow g(n) \in o(f(n))$

## Laufzeitanalyse

### *Beispiele*

- $n \in o(n^2)$
- $n \notin o(n)$

### *Eine weitere Interpretation*

- Grob gesprochen sind  $O, \Omega, \Theta, o, \omega$  die „asymptotischen Versionen“ von  $\leq, \geq, =, <, >$  (in dieser Reihenfolge)

### *Schreibweise*

- Wir schreiben häufig  $f(n) = O(g(n))$  anstelle von  $f(n) \in O(g(n))$

## 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) (siehe Vollversion)*

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

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

## Teile & Herrsche

MergeSort(Array A, p, r)

- |    |                                        |                                   |
|----|----------------------------------------|-----------------------------------|
| 1. | <b>if</b> $p < r$ <b>then</b>          | ➤ 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  $A[1 \dots n]$

## Teile & Herrsche

### *Erweiterte Induktion*

- (I.A.)  $A(1)$  ist richtig
- (I.V.)  $A(m)$  gilt für alle  $1 \leq m \leq n$
- (I.S.) Aus (I.V.) folgt  $A(n+1)$
  
- 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).  $A(m)$  gilt für alle  $m \in \{1 \dots n\}$  für ein beliebiges, festes  $n$ .

## Teile & Herrsche

### Satz 7

- Algorithmus MergeSort( $A, p, r$ ) sortiert das Feld  $A[p..r]$  korrekt.

### Beweis

- Wir zeigen die Korrektheit per Induktion über  $n=r-p$ . Ist  $m < 0$ , so ist nichts zu zeigen.
- (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 MergeSort( $A, p, r$ ) das Feld  $A[p..r]$  korrekt.

## Teile & Herrsche

### Satz 7

- Algorithmus 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 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. Nun folgt die Korrektheit aus der Tatsache, dass Merge die beiden Bereiche korrekt zu einer sortierten Liste zusammenfügt.

## Teile & Herrsche

MergeSort(Array A, p, r)

- |                                           |                                   |
|-------------------------------------------|-----------------------------------|
| 1. <b>if</b> $p < r$ <b>then</b>          | ➤ 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(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	1
$1+T(n/2)$	$1+T(n/2)$
$1+T(n/2)$	$1+T(n/2)$
$\leq cn$	$\leq cn$
$2T(n/2) + cn$	

### *Aufruf des Algorithmus*

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

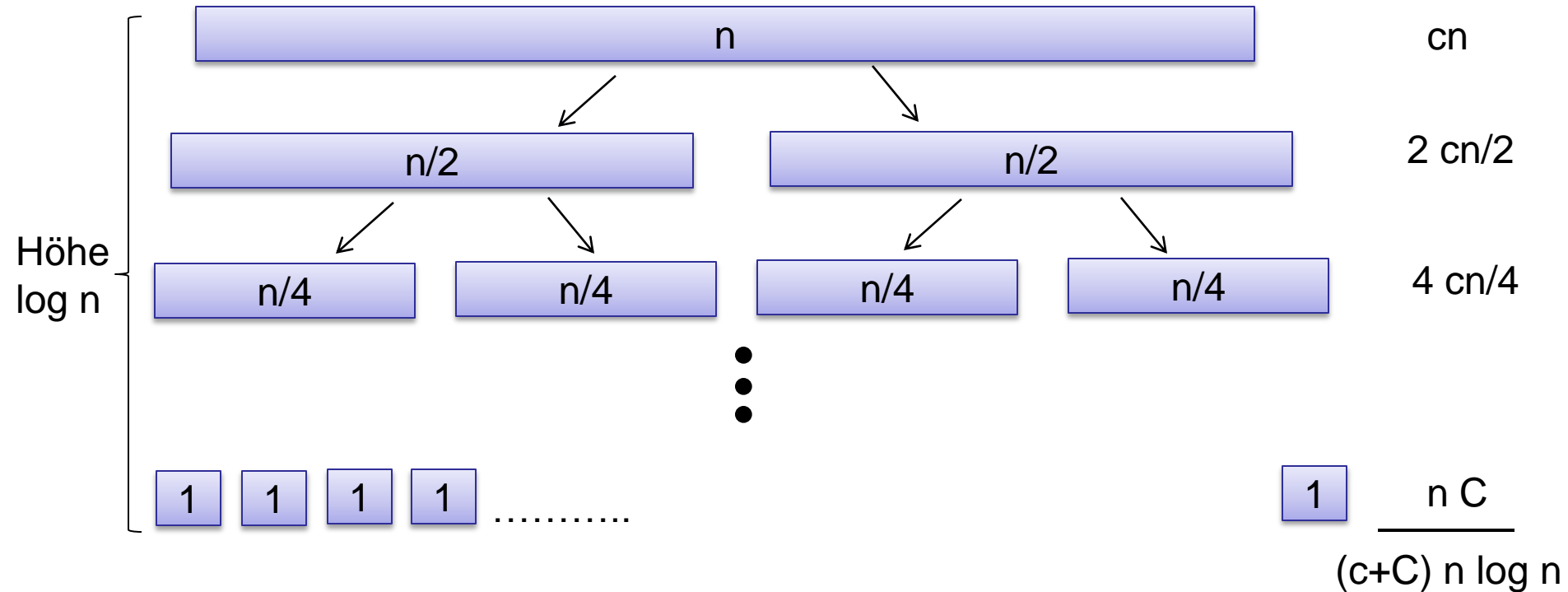
## Teile & Herrsche

### *Laufzeit 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.

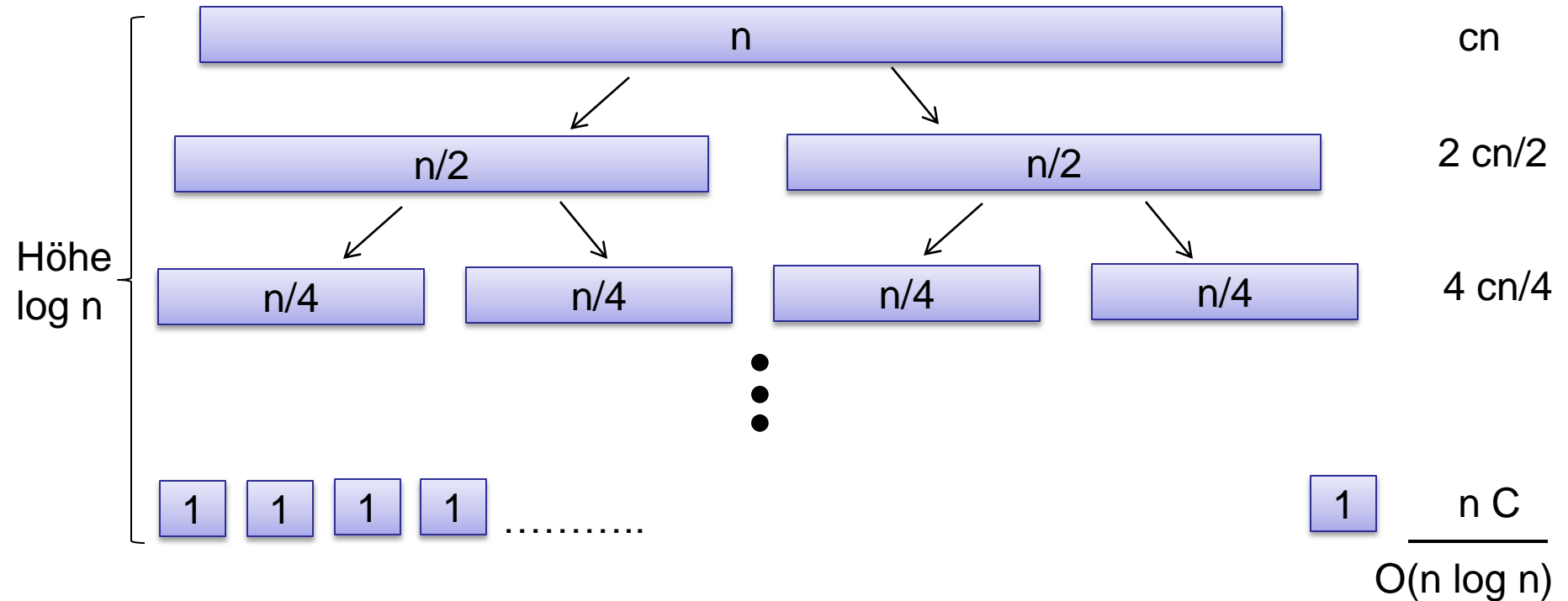
## Teile & Herrsche

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



## Teile & Herrsche

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



## Teile & Herrsche

### Satz 8

- Algorithmus MergeSort hat eine Laufzeit von  $\Theta(n \log n)$ .

### Beweis

- Sei  $T(2) \leq C'$  und  $C^* \geq \max\{c, C'\}$ . Wir zeigen per Induktion,  $T(n) \leq C^* n \log n$ .
- (I.A.) für  $n=2$  gilt  $T(2) \leq C' \leq C^* 2 \log 2$ .
- (I.V.) Für Eingabelänge  $2 < m < n$  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
$$\begin{aligned} T(n) &\leq 2 C^* n/2 \log(n/2) + cn \\ &\leq C^* n (\log(n)-1) + cn \\ &\leq C^* n (\log(n)-1) + C^* n \\ &\leq C^* n \log(n) \\ &= O(n \log n). \end{aligned}$$
- $\Omega(n \log n)$  -> Übung.

## Teile & Herrsche

### *Weiteres Beispiel*

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

BinäreSuche( $A, b, p, r$ )

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

## Teile & Herrsche

BinäreSuche(A,b,p,r)

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

### Aufruf

- BinäreSuche(A,b,1,n)

## Teile & Herrsche

BinäreSuche(A,b,p,r)

1. **if** p=r **then return** p
2. **else**
3.     q ←  $\lfloor (p+r)/2 \rfloor$
4.     **if** b ≤ 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
---	---	----	----	----	----	----

Suche b=23 (siehe Vollversion)

## Teile & Herrsche

### Satz 9

- Algorithmus 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 BinäreSuche( $A, b, p, r$ ) den Index einer Zahl  $b$  in einem sortierten Feld  $A[p..r]$ .

## Teile & Herrsche

### Satz 9

- Algorithmus 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 BinäreSuche( $A, b, p, r$ ) den Index einer Zahl  $b$  in einem sortierten Feld  $A[p..r]$ .
- (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. Es gilt  $q \geq p$  und  $q < r$ . Ist  $b \leq A[q]$ , so wird 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 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.      $q \leftarrow \lfloor (p+r)/2 \rfloor$
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=5

Suche b=23; Gefunden!

## Teile & Herrsche

BinäreSuche(A,b,p,r)

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

Laufzeit:

1

1

1

$1+T(\lceil n/2 \rceil)$

$1+T(\lfloor n/2 \rfloor)$

---

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

### 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.     q ← ⌊(p+r)/2⌋
4.     **if** b ≤ A[q] **then return** BinäreSuche(A,b,p,q)
5.     **else return** BinäreSuche(A,b,q+1,r)

Laufzeit:

1

1

1

1+T(⌈n/2⌉)

1+T(⌊n/2⌋)

---

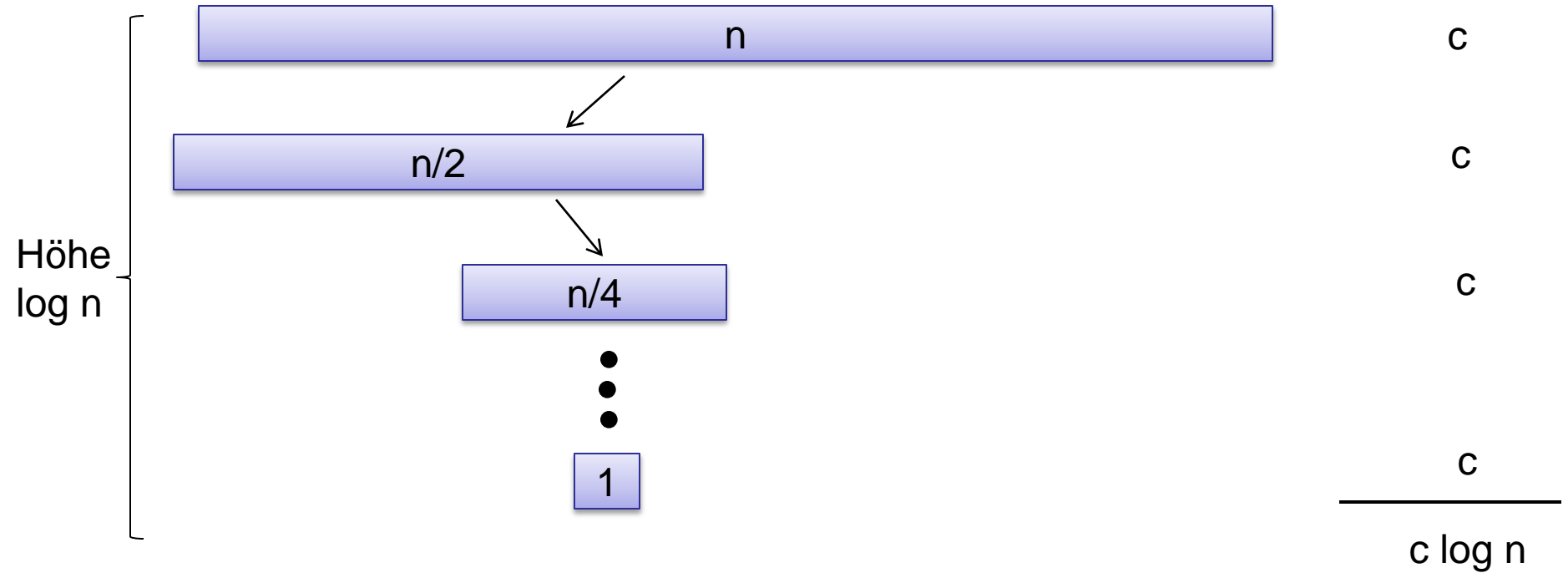
5+ max{T(⌈n/2⌉), T(⌊n/2⌋)}

### 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

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



## Teile & Herrsche

### Satz 10

- Algorithmus BinäreSuche hat eine Laufzeit von  $O(\log n)$ .

### 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, dass  $T(n) \leq T(\lceil n/2 \rceil) + 5$ . Nach (I.V.) gilt somit  $T(n) \leq 5 \lceil \log(\lceil n/2 \rceil) \rceil + 1 + 5$ . 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  $T(n) \leq 5 \lceil \log n \rceil + 1$ .

## Zusammenfassung

### *Teile & Herrsche*

- Teile die Eingabe in mehrere Teile
- Löse das Problem rekursiv auf den Teilen
- Füge die Teile geschickt wieder zusammen

### *Laufzeitrekursionen*

- Weglassen von Runden liefert normalerweise das richtige Ergebnis in O-Notation
- Intuition über Rekursionsbaum
- Eigentlicher Beweis per Induktion