ALGORITHMISCHES PRINZIP

TEILE UND HERRSCHE (DIVIDE AND CONQUER)

Teile & Herrsche – Mergesort

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) – Mergesort

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

Teile & Herrsche – Mergesort

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 – Mergesort

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

INTERNET NETWORK ARCHITECTURES

Teile & Herrsche – Mergesort

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**	**25**

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

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

INTERNET NETWORK ARCHITECTURES

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
- Wovon hängt die Laufzeit ab?
  - Anzahl der Teilprobleme
  - Größe der Teilprobleme
  - Algorithmus für das Zusammensetzen der Teilprobleme

Teile & Herrsche – Rekursionsgleichung

Laufzeiten als Rekursionsgleichung in der Form

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

- (und $T(1) = const$)

Teile & Herrsche – Rekursionsgleichung

Laufzeiten als Rekursionsgleichung in der Form

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

Anzahl Unterprobleme

- (und T(1) = const)

INTERNET NETWORK ARCHITECTURES

Teile & Herrsche – Rekursionsgleichung

Laufzeiten als Rekursionsgleichung in 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 – Rekursionsgleichung

Laufzeiten als Rekursionsgleichung in 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 – Rekursionsgleichung

Laufzeiten als Rekursionsgleichung in der Form

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

Anzahl Unterprobleme

Größe der Unterprobleme
(bestimmt Höhe des Rekursionsbaums)

Aufwand für Aufteilen und Zusammenfügen

- (und T(1) = const)

Welche unterschiedlichen Fälle gibt es?

Teile & Herrsche – Rekursionsgleichung Mergesort

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 $T(1) = \text{const}$)

(n Zweierpotenz)

Teile & Herrsche – Binäre Suche

Weiteres Beispiel:

- Problem: Finde Element x in sortiertem Feld
- Algorithmus: Binäre Suche

- Wie:
  - Suche:
    - Vergleiche Element x mit dem mittleren Element des Feldes
    - Wenn gefunden, gib Feldindex zurück
    - Wenn kleiner, dann suche rekursiv im linken Teilfeld
    - Wenn größer, dann suche rekursiv im rechten Teilfeld
  - Abbruchbedingung:
    - Wenn Feldgröße == 0 gib Element nicht gefunden zurück

Teile & Herrsche – Binäre Suche

Weiteres Beispiel

- Problem: Finde Element b 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) // A Feld, b Element, p, r Feldanfang, -ende

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 – Binäre Suche

BinäreSuche(A,b,p,r) // A Feld, b Element, p, r Feldanfang, -ende

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 – Binäre Suche

BinäreSuche(A,b,p,r) // A Feld, b Element, p, r Feldanfang, -ende

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**

Suche b=23

Teile & Herrsche – Binäre Suche

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=1 r=7

Suche b=23

Teile & Herrsche – Binäre Suche

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 – Binäre Suche

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=7

Suche b=23

Teile & Herrsche – Binäre Suche

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	q=6	r=7

Suche b=23

Teile & Herrsche – Binäre Suche

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 $\le$ 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	

Suche b=23

Teile & Herrsche – Binäre Suche

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 ≤ 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 q=5	r=6	

Suche b=23

Teile & Herrsche – Binäre Suche

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 – Binäre Suche

BinäreSuche(A,b,p,r)

1. $\boxed{\textbf{if } p=r \textbf{ then return } p}$
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 – Binäre Suche Beweisidee

Satz

- 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.

Beweisidee

- Wir zeigen die Korrektheit per Induktion
- Annahme laut Satz: b ist in A[p..r] vorhanden
- Induktionsvoraussetzung:
  - b wird in einem Teilfeld gefunden **// Rekursionsschritt**
- Induktionsschritt:
  - Finden des passenden Teilfeldes, durch Vergleich des mittleren Elementes mit b **// Auswahlschritt**
  - Nutzen der Induktionsvoraussetzung

Teile & Herrsche – Binäre Suche Beweis

Satz

- 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.

Teile & Herrsche – Binäre Suche Beweis

Satz

- 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.

Teile & Herrsche – Binäre Suche Beweis

Satz

- 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], sofern b im Feld vorhanden ist.

Teile & Herrsche – Binäre Suche Beweis

Satz

- 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≤m≤n findet 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 – Binäre Suche Beweis

Satz

- 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 $0 \le m \le n$ findet 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 – Binäre Suche Beweis

Satz

- 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 $0 \le m \le n$ findet 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. Es gilt $q \ge p$ und q<r. Ist $b \le A[q]$, so wird BinäreSuche rekursiv für A[p..q] aufgerufen.

Teile & Herrsche – Binäre Suche
Beweis

Satz

- 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 $0 \le m \le n$ findet 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. Es gilt $q \ge p$ und $q<r$. Ist $b \le A[q]$, 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.

Teile & Herrsche – Binäre Suche Beweis

Satz

- 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], 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. 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].

Teile & Herrsche – Binäre Suche Beweis

Satz

- 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 $0 \le m \le n$ findet 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. Es gilt $q \ge p$ und q<r. Ist $b \le A[q]$, 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 – Binäre Suche Beweis

Satz

- 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 $0 \le m \le n$ findet 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. Es gilt $q \ge p$ und q<r. Ist $b \le A[q]$, 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 – Binäre Suche Laufzeit

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 $\le$ A[q] **then return** BinäreSuche(A,b,p,q)
5. **else return** BinäreSuche(A,b,q+1,r)

Laufzeit:

Laufzeit

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

Teile & Herrsche – Binäre Suche Laufzeit

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)

Laufzeit:

1

Laufzeit

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

Teile & Herrsche – Binäre Suche Laufzeit

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)

Laufzeit:

1

1

Laufzeit

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

Teile & Herrsche – Binäre Suche Laufzeit

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

Laufzeit

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

Teile & Herrsche – Binäre Suche Laufzeit

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

Laufzeit

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

Teile & Herrsche – Binäre Suche Laufzeit

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

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

Teile & Herrsche – Binäre Suche Laufzeit

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

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

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

Teile & Herrsche – Binäre Suche Laufzeit

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

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

INTERNET NETWORK ARCHITECTURES

Teile & Herrsche – Binäre Suche Rekursionsgleichung

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 – Binäre Suche Rekursionsgleichung

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

n	c

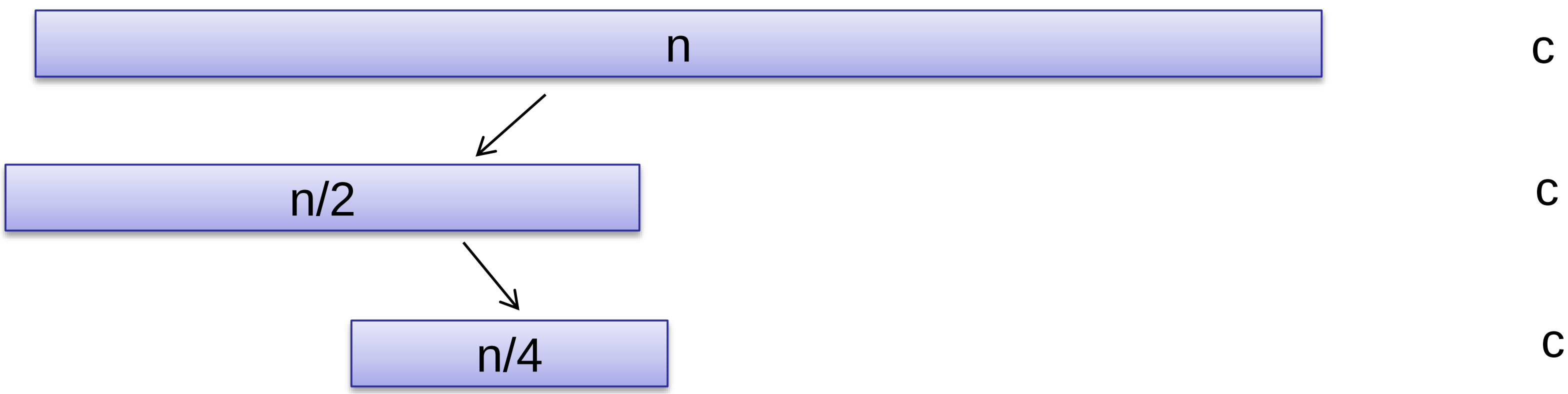
Teile & Herrsche – Binäre Suche Rekursionsgleichung

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

n	c
$n/2$	c

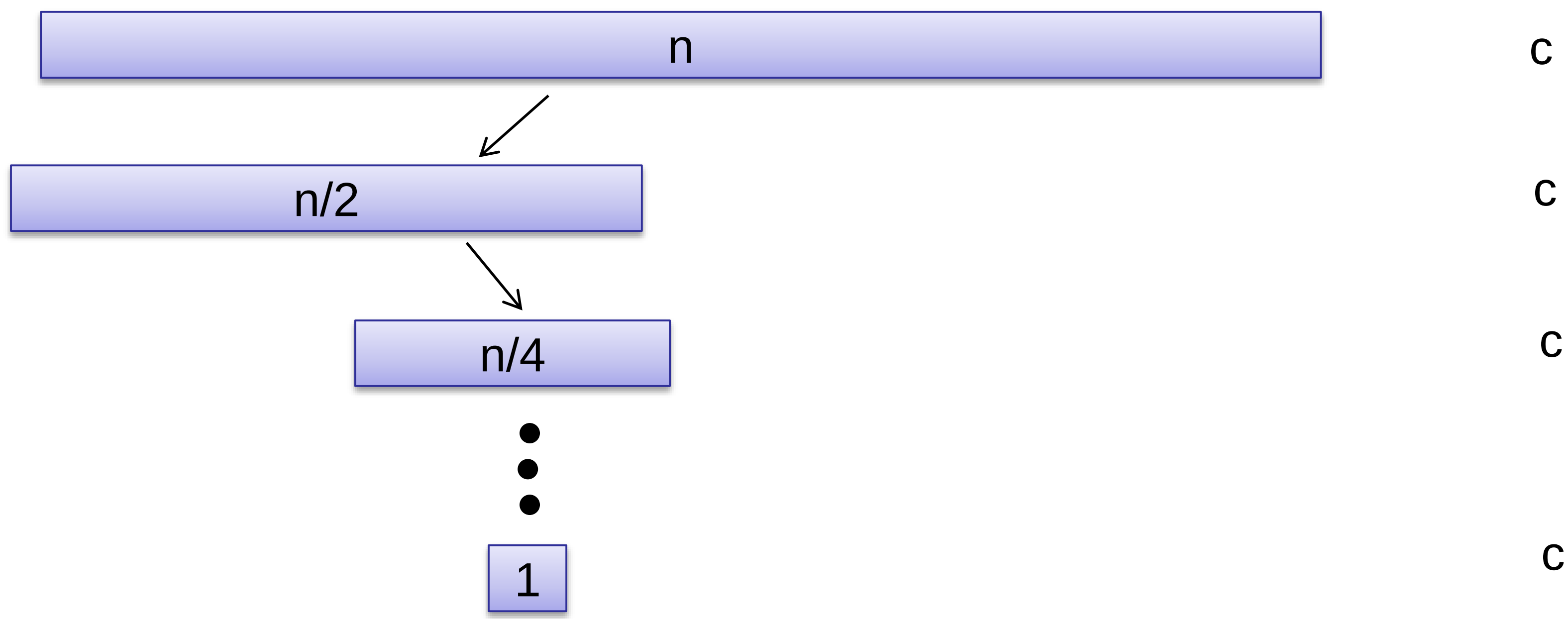
Teile & Herrsche – Binäre Suche Rekursionsgleichung

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



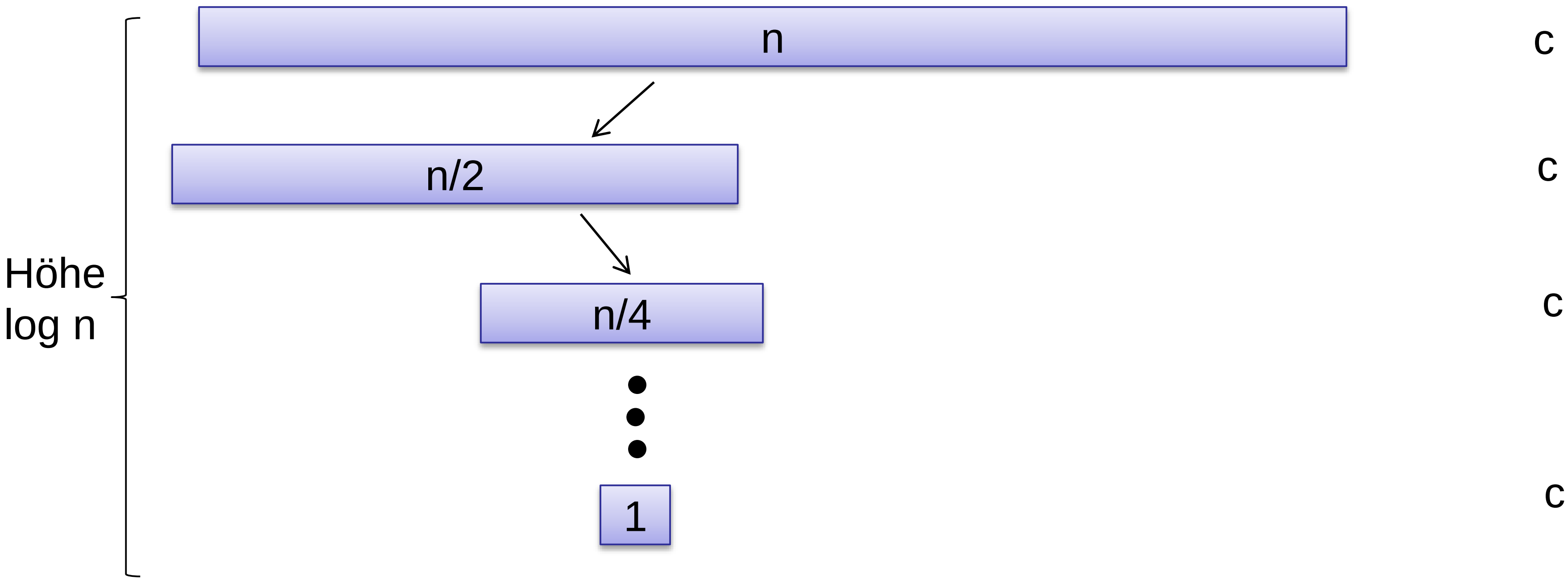
Teile & Herrsche – Binäre Suche Rekursionsgleichung

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



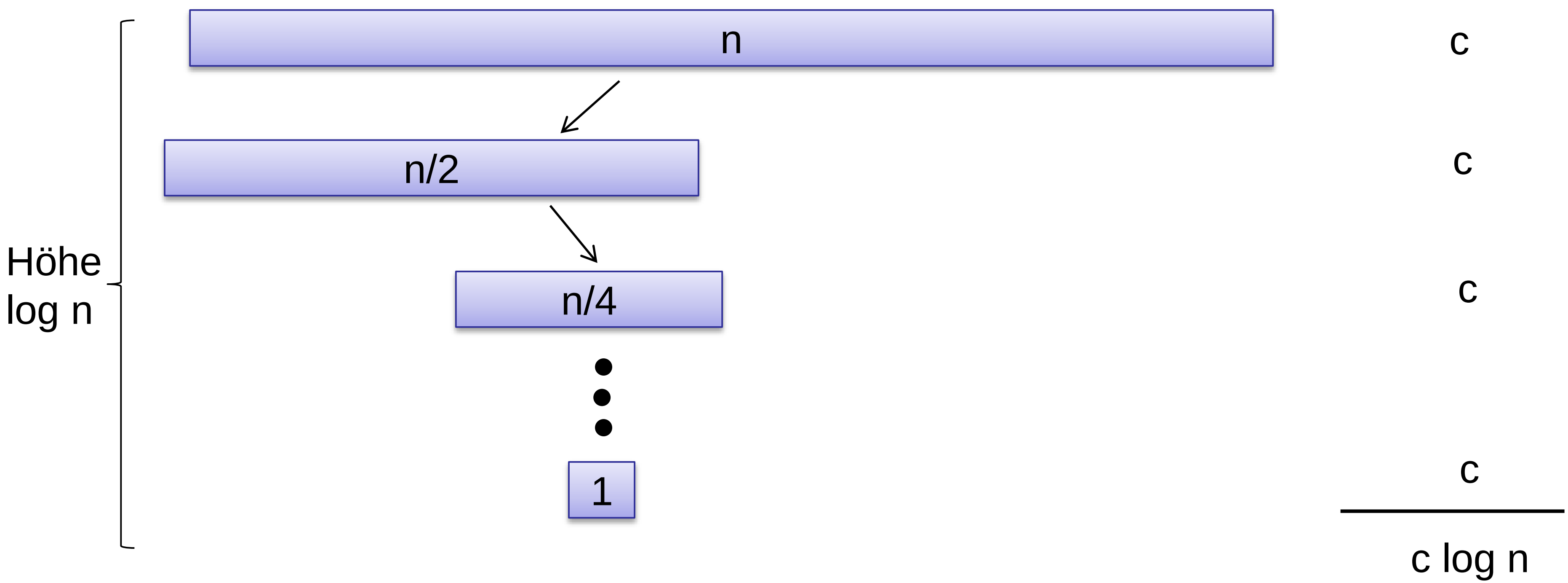
Teile & Herrsche – Binäre Suche
Rekursionsgleichung

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



Teile & Herrsche – Binäre Suche
Rekursionsgleichung

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



Teile & Herrsche – Binäre Suche Laufzeit

Satz

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

Beweis

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

Teile & Herrsche – Binäre Suche Laufzeit

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

Teile & Herrsche – Integer Multiplikation

Integer Multiplikation

- Problem: Multipliziere zwei n-Bit Integer
- Eingabe: Zwei n-Bit Integer X,Y
- Ausgabe: 2n-Bit Integer Z mit Z=XY

Annahmen:

- Wir können n-Bit Integer in $\Theta(n)$ (worst case) Zeit addieren
- Wir können n-Bit Integer in $\Theta(n+k)$ (worst case) Zeit mit 2^k multiplizieren (durch Shift)

Teile & Herrsche – Integer Multiplikation Schulmethode

Schulmethode: (13·11)

Teile & Herrsche – Integer Multiplikation Schulmethode

Schulmethode: (13·11)

$$\begin{array}{c} 1101 \cdot 1011 \\ \hline \end{array}$$

Teile & Herrsche – Integer Multiplikation Schulmethode

Schulmethode: (13·11)

$$\begin{array}{r} 1101 \cdot 1011 \\ \hline 1101 \end{array}$$

INTERNET NETWORK ARCHITECTURES

Teile & Herrsche – Integer Multiplikation Schulmethode

Schulmethode: (13·11)

$$
\begin{array}{r}
1101 \cdot 1011 \\
\hline
1101 \\
1101
\end{array}
$$

Teile & Herrsche – Integer Multiplikation Schulmethode

Schulmethode: (13·11)

```
  1101 · 1011
 -------------
        1101
       1101
     1101
```

Teile & Herrsche – Integer Multiplikation Schulmethode

Schulmethode: (13·11)

$$\begin{array}{r}
1101 \cdot 1011 \\
\hline
1101 \\
1101 \\
1101 \\
\hline
\end{array}$$

Teile & Herrsche – Integer Multiplikation Schulmethode

Schulmethode: (13·11)

$$
\begin{array}{r}
1101 \cdot 1011 \\
\hline
1101 \\
1101 \\
1101 \\
\hline
10001111
\end{array}
$$

Teile & Herrsche – Integer Multiplikation
Schulmethode – Laufzeit

Laufzeit Schulmethode

- n Multiplikationen mit 2^k für ein $k \leq n$
- n-1 Additionen im worst-case:

$$\underbrace{11\ldots111}_{\text{n-Bit}} \cdot \underbrace{11\ldots111}_{\text{n-Bit}}$$

- Jede Addition $\Theta(n)$ Zeit
- Insgesamt $\Theta(n^2)$ Laufzeit

Teile & Herrsche – Integer Multiplikation Schulmethode – Laufzeit

Laufzeit Schulmethode

- n Multiplikationen mit 2^k für ein k≤n
- n-1 Additionen im worst-case:

$$\underbrace{11\ldots111}_{\text{n-Bit}} \cdot \underbrace{11\ldots111}_{\text{n-Bit}}$$

- Jede Addition $\Theta(n)$ Zeit
- Insgesamt $\Theta(n^2)$ Laufzeit

Bessere Laufzeit mit Teile & Herrsche?

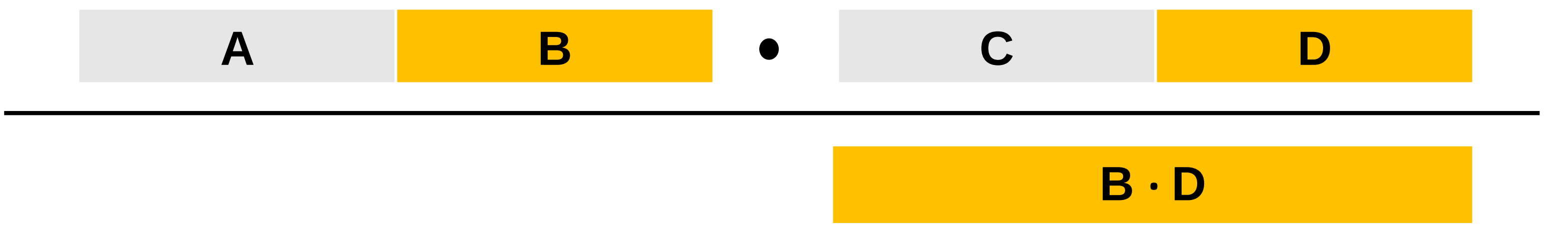
Teile & Herrsche – Integer Multiplikation
Erster Versuch

Integer Multiplikation

A	B	•	C	D

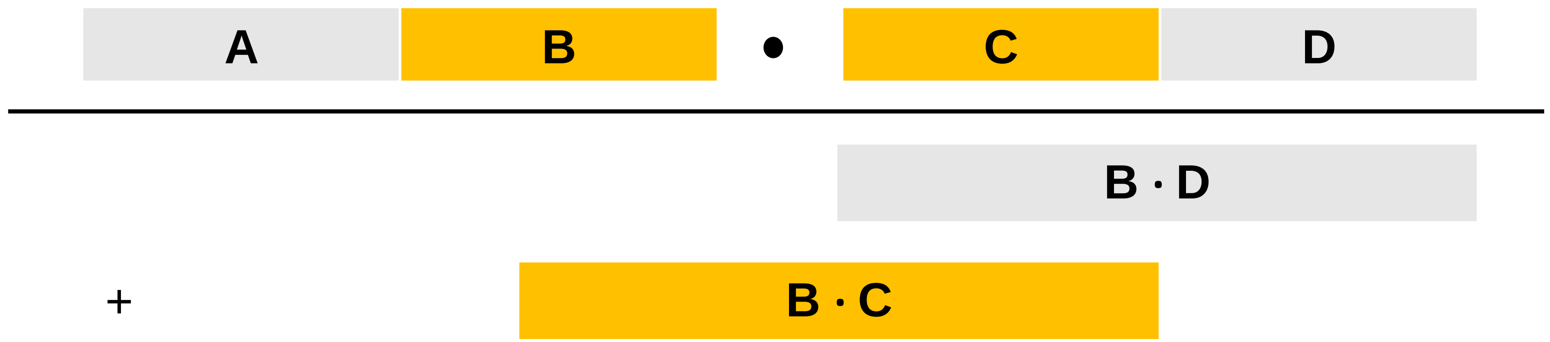
Teile & Herrsche – Integer Multiplikation
Erster Versuch

Integer Multiplikation



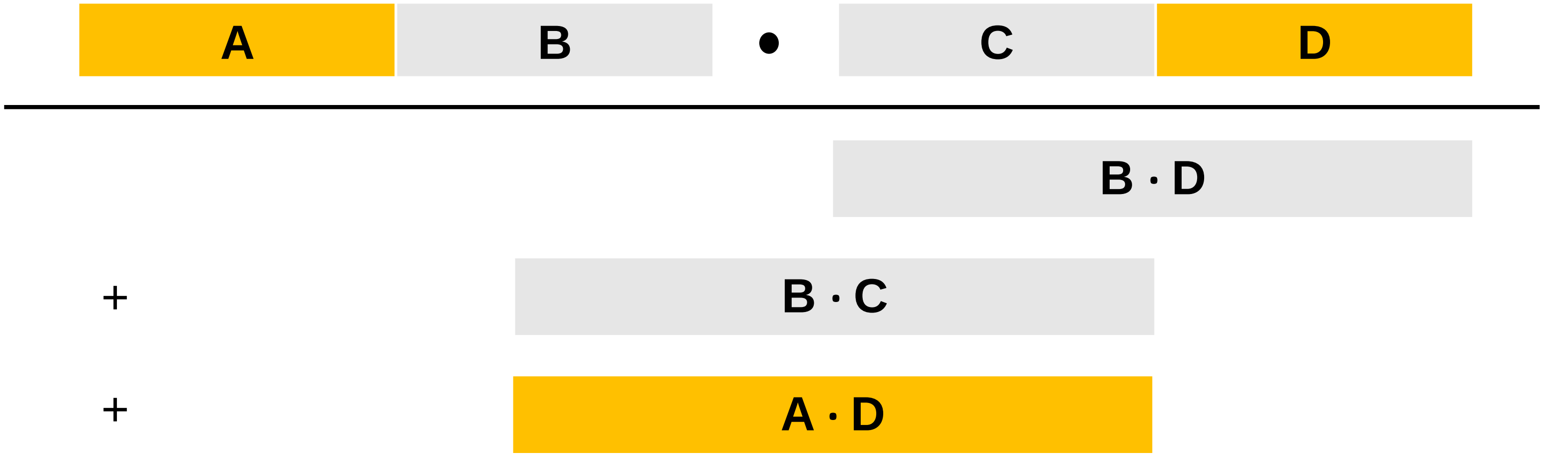
Teile & Herrsche – Integer Multiplikation Erster Versuch

Integer Multiplikation



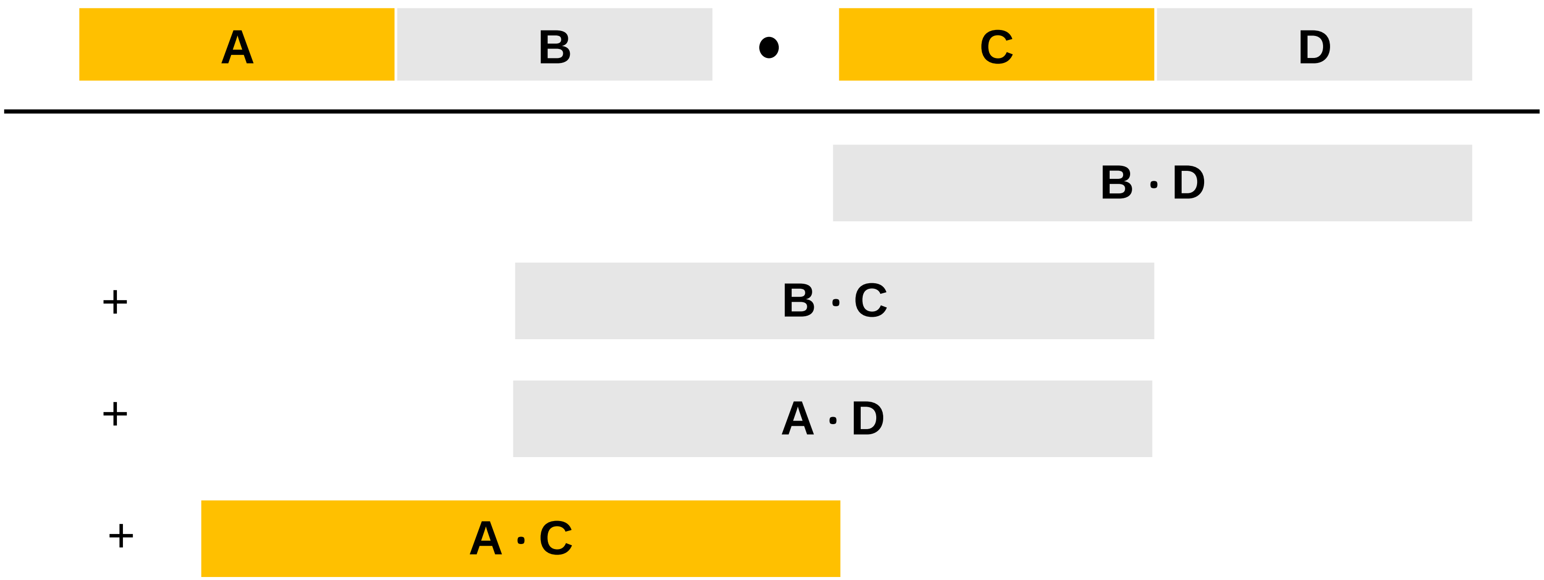
Teile & Herrsche – Integer Multiplikation Erster Versuch

Integer Multiplikation



Teile & Herrsche – Integer Multiplikation Erster Versuch

Integer Multiplikation



INTERNET NETWORK ARCHITECTURES

Teile & Herrsche – Integer Multiplikation Erster Versuch

Integer Multiplikation

A B • C D

B · D

\+ B · C

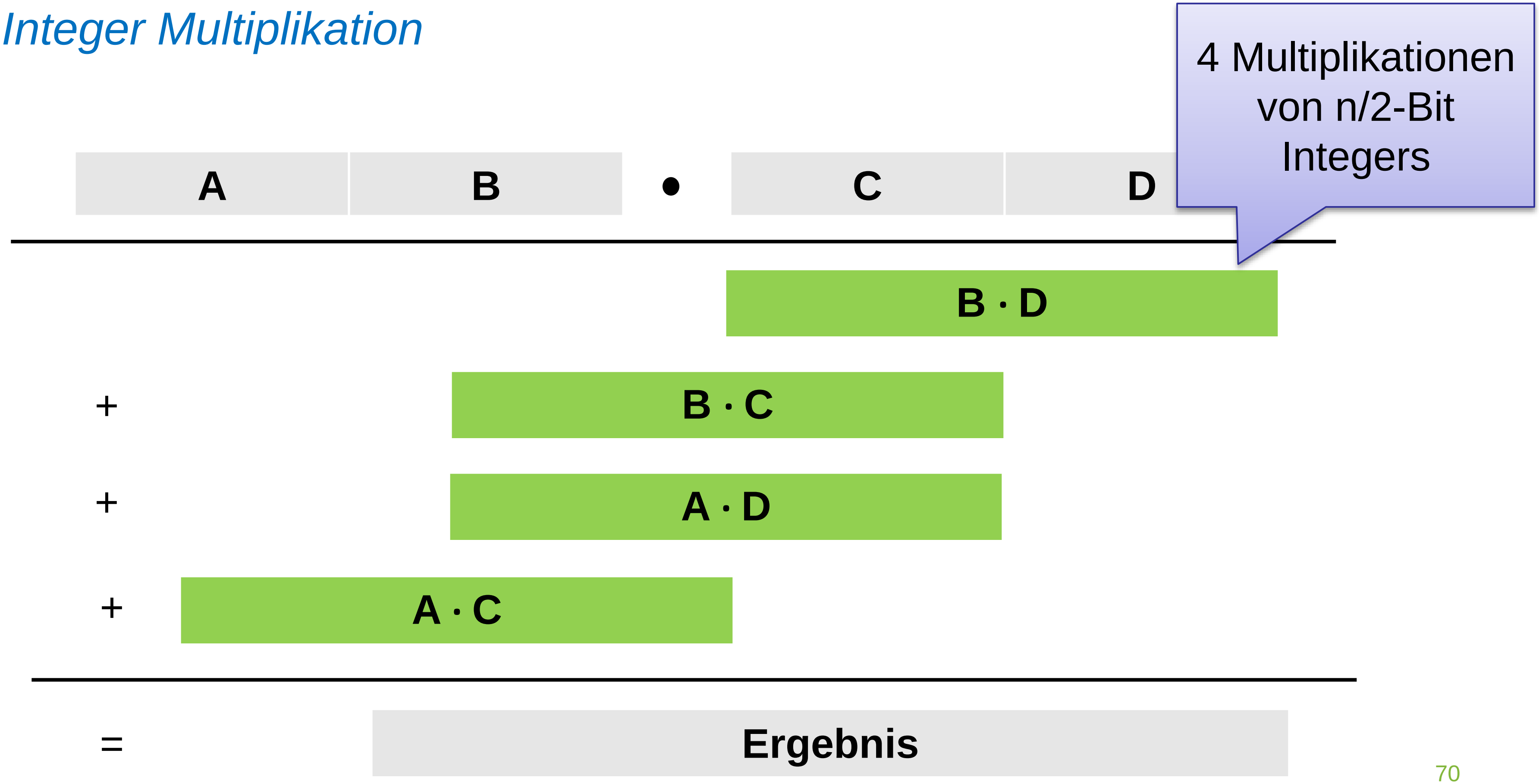
\+ A · D

\+ A · C

= **Ergebnis**

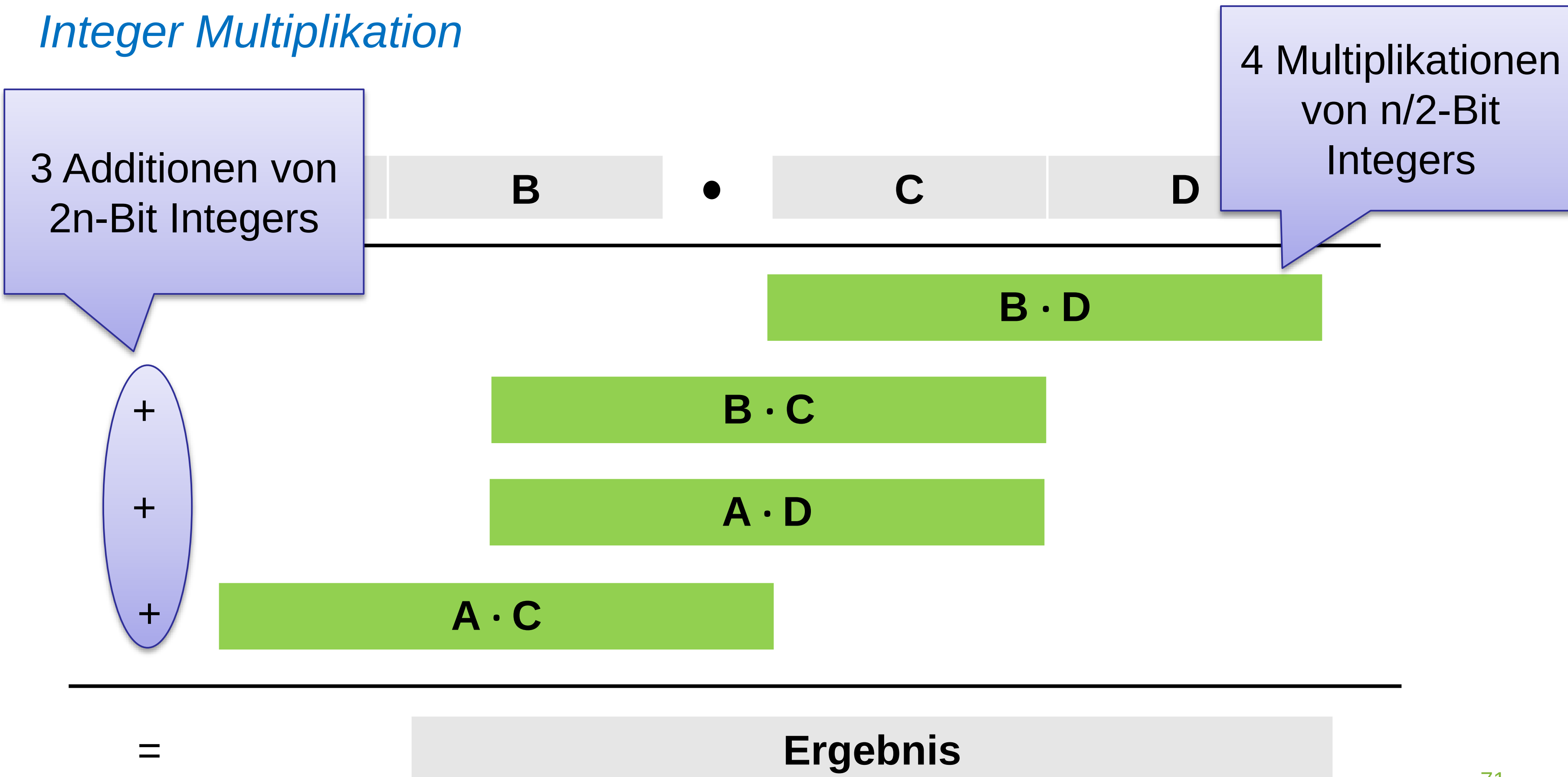
Teile & Herrsche – Integer Multiplikation Erster Versuch

Integer Multiplikation



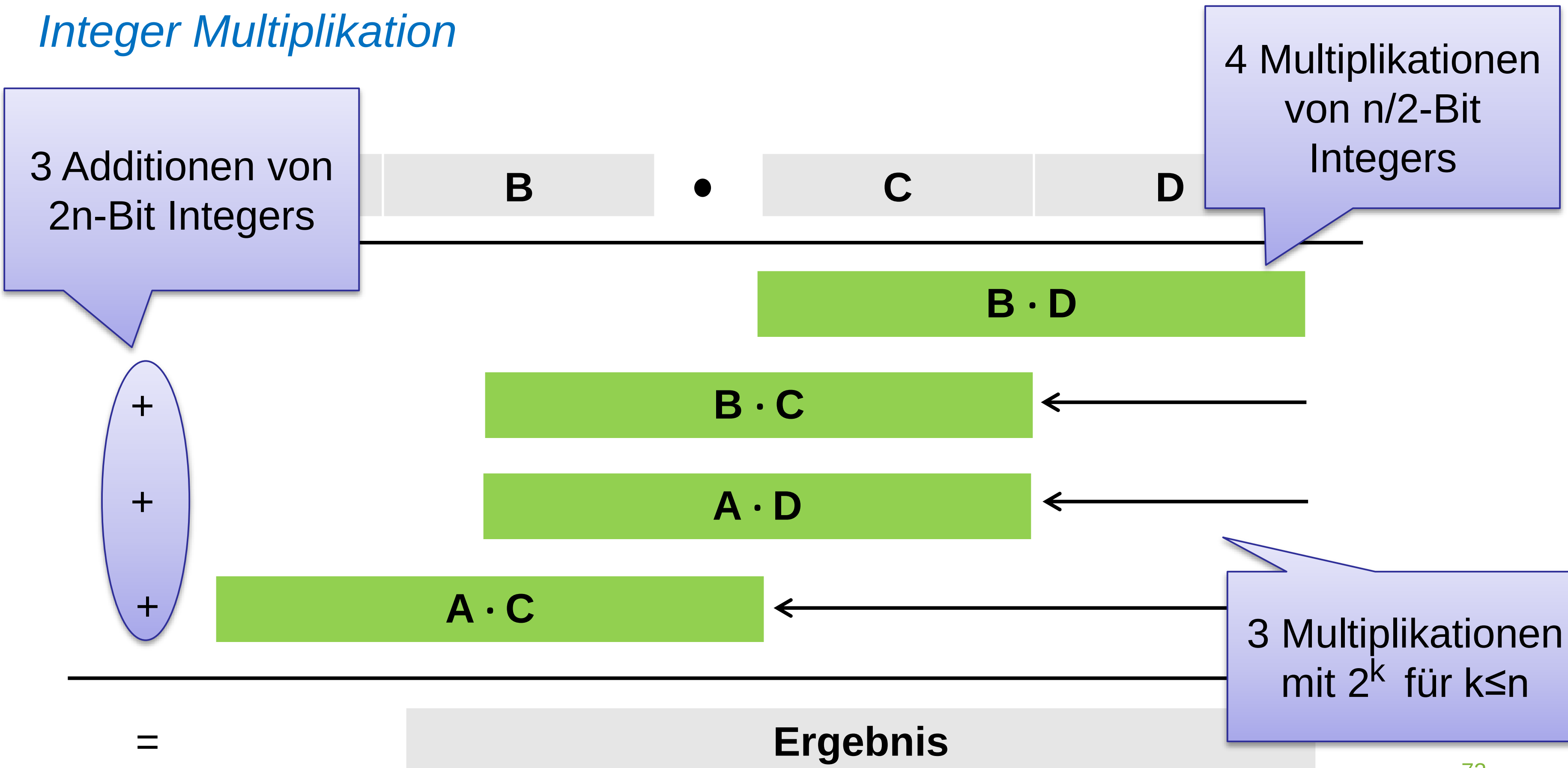
Teile & Herrsche – Integer Multiplikation Erster Versuch

Integer Multiplikation



Teile & Herrsche – Integer Multiplikation
Erster Versuch

Integer Multiplikation



Teile & Herrsche – Integer Multiplikation Erster Versuch – Rekursionsgleichung

Beispiel Multiplikation Schulmethode

$$T(n) = 4 \cdot T(n/2) + cn$$

Anzahl Unterprobleme

Aufwand für Aufteilen und Zusammenfügen

Größe der Unterprobleme
(bestimmt Höhe des Rekursionsbaums)

- (und T(1) = const)

(n Zweierpotenz)

Teile & Herrsche – Integer Multiplikation
Erster Versuch – Rekursionsgleichung

Laufzeit einfaches Teile & Herrsche

$$T(n) \leq \begin{cases} 4\,T(n/2) + cn & , n>1 \\ c & , n=1 \end{cases}$$

c geeignete Konstante

Teile & Herrsche – Integer Multiplikation Erster Versuch – Rekursionsgleichung

Laufzeit einfaches Teile & Herrsche

$$T(n) \leq \begin{cases} 4\,T(n/2) + cn & , n>1 \\ c & , n=1 \end{cases}$$

c geeignete Konstante

n

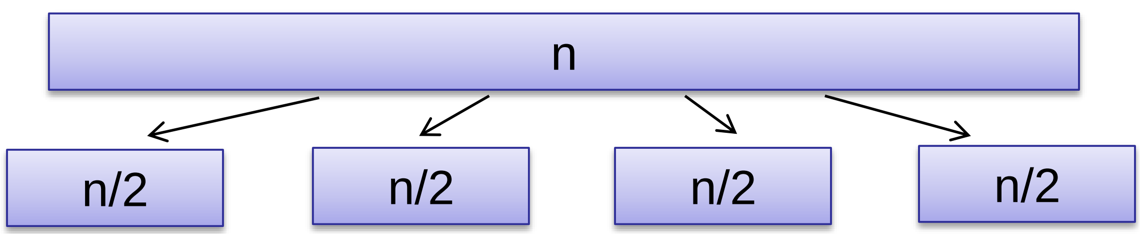
cn

Teile & Herrsche – Integer Multiplikation
Erster Versuch – Rekursionsgleichung

Laufzeit einfaches Teile & Herrsche

$$T(n) \le \begin{cases} 4\,T(n/2) + cn & , n>1 \\ c & , n=1 \end{cases}$$

c geeignete Konstante



cn

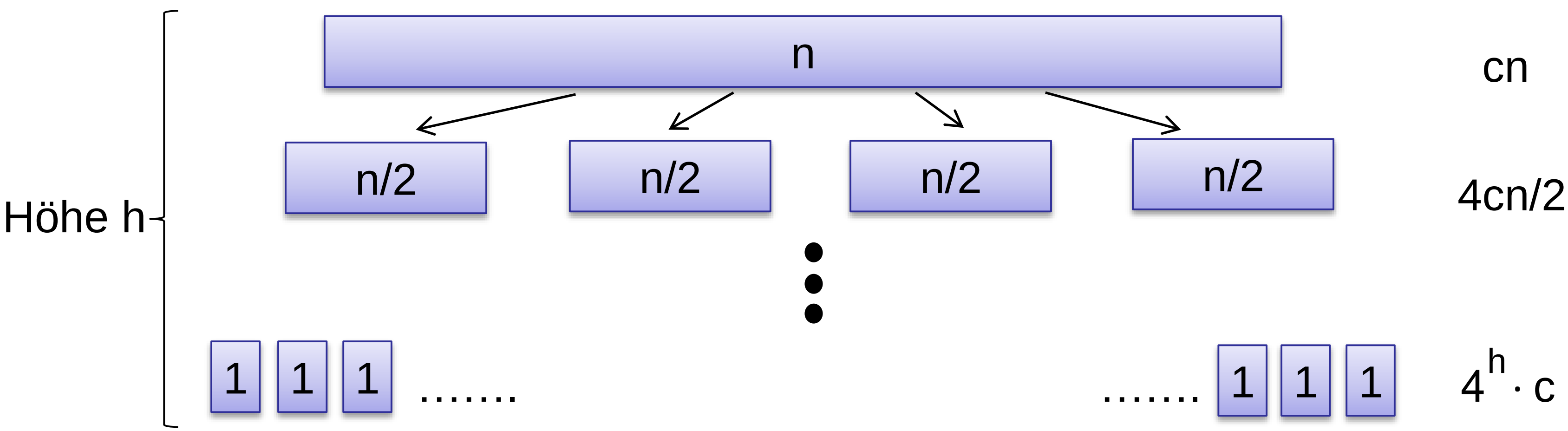
4cn/2

Teile & Herrsche – Integer Multiplikation Erster Versuch – Rekursionsgleichung

Laufzeit einfaches Teile & Herrsche

$$T(n) \leq \begin{cases} 4\,T(n/2) + cn & , n>1 \\ c & , n=1 \end{cases}$$

c geeignete Konstante

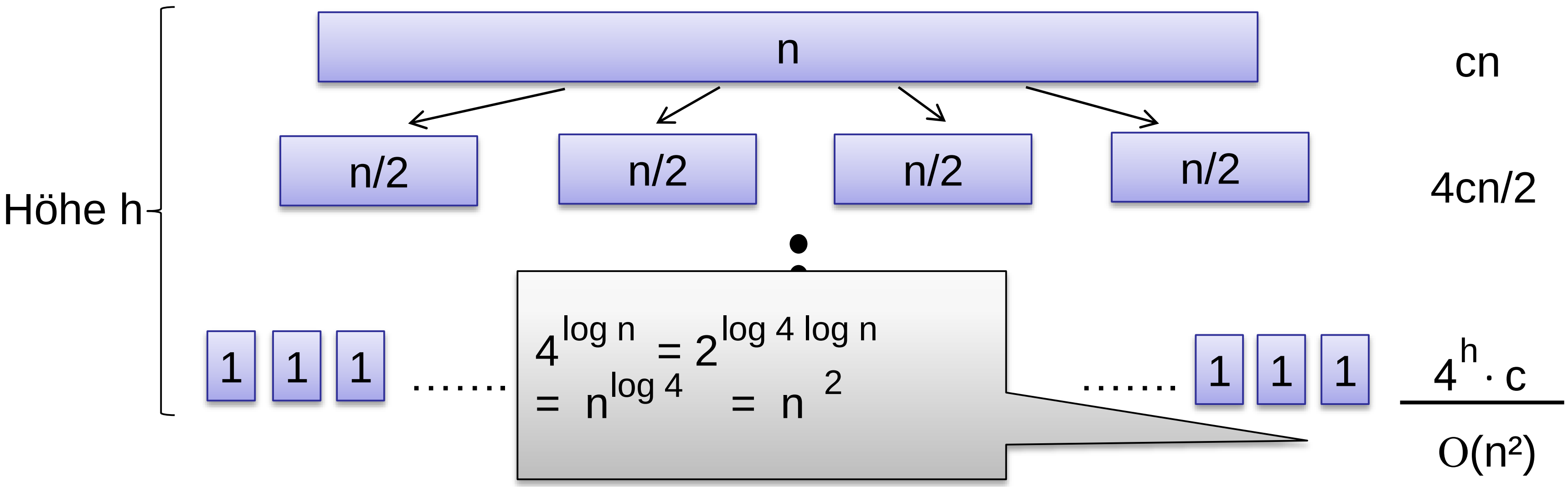


Teile & Herrsche – Integer Multiplikation
Erster Versuch – Rekursionsgleichung

Laufzeit einfaches Teile & Herrsche

$$T(n) \leq \begin{cases} 4\,T(n/2) + cn & , n>1 \\ c & , n=1 \end{cases}$$

c geeignete Konstante



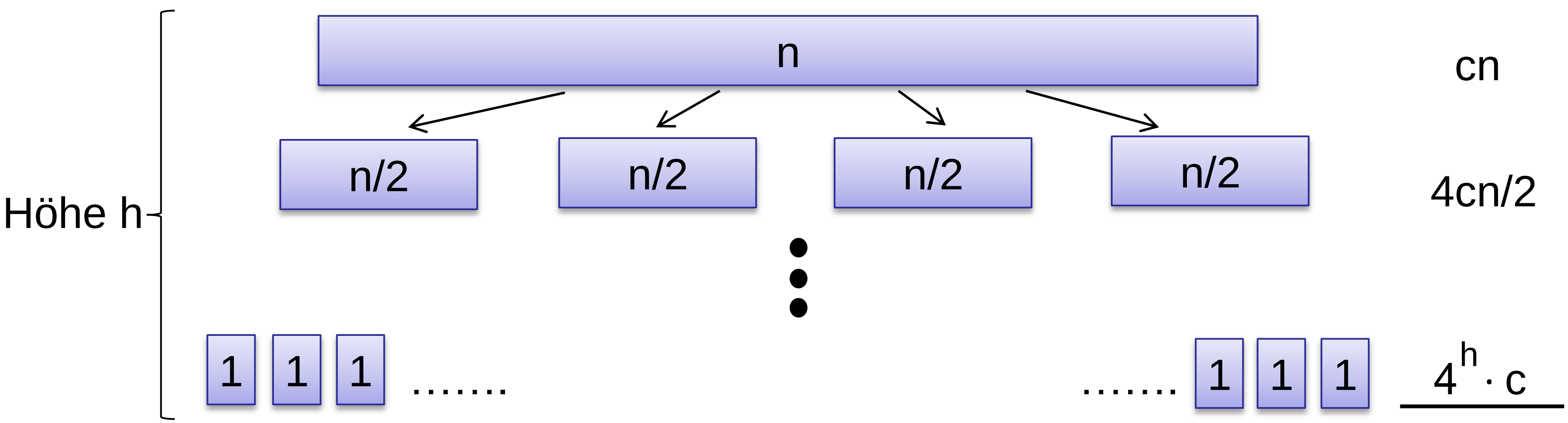
Höhe des Baums: h = log n

Teile & Herrsche – Integer Multiplikation
Erster Versuch – Rekursionsgleichung

Laufzeit einfaches Teile & Herrsche

$$T(n) \leq \begin{cases} 4\,T(n/2) + cn & , n>1 \\ c & , n=1 \end{cases}$$

c geeignete Konstante



O(n²)

Nichts gewonnen!

Höhe des Baums: h = log n

Teile & Herrsche – Integer Multiplikation Zweiter Versuch

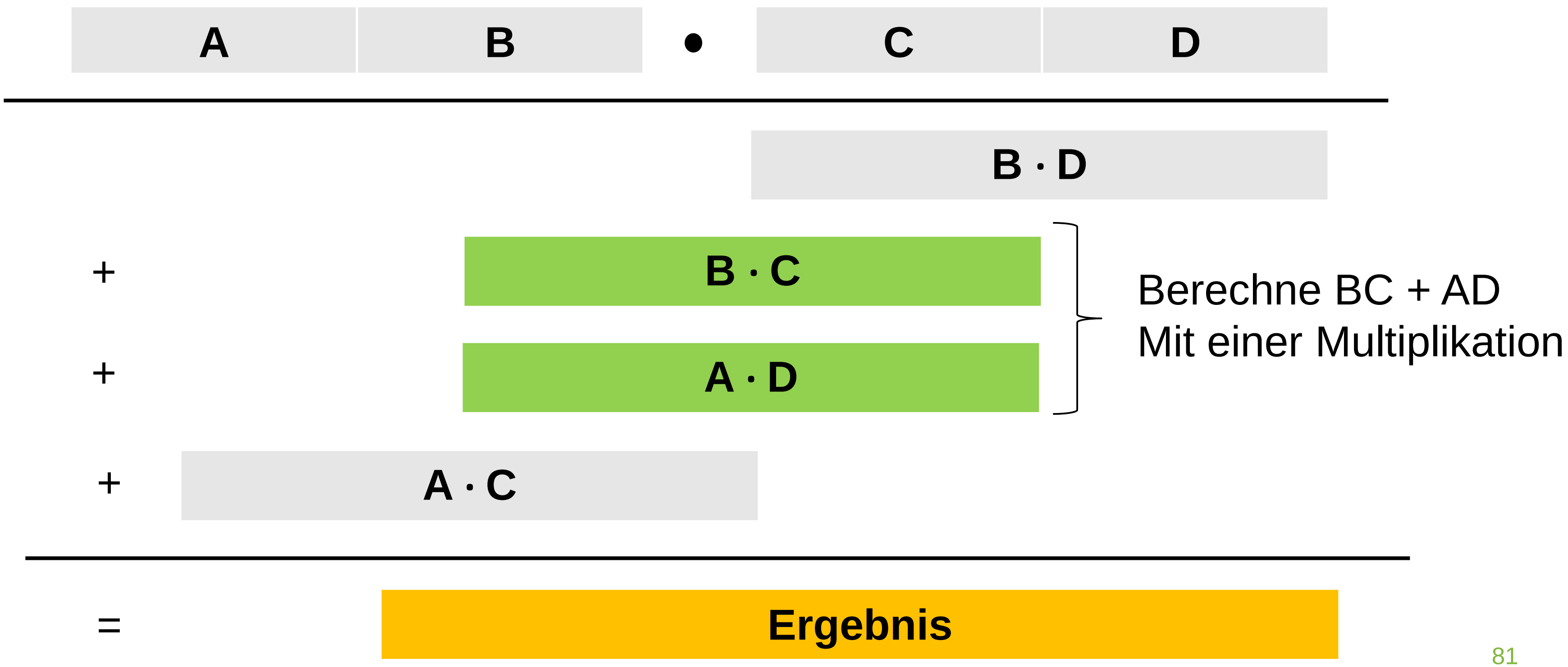
Verbesserte Integer Multiplikation

A	B	•	C	D

$$
\begin{array}{lr}
 & B \cdot D \\
+ & B \cdot C \quad\quad \\
+ & A \cdot D \quad\quad \\
+ & A \cdot C \quad\quad\quad\quad \\
\hline
= & \text{Ergebnis}
\end{array}
$$

Teile & Herrsche – Integer Multiplikation
Zweiter Versuch

Verbesserte Integer Multiplikation



INTERNET NETWORK ARCHITECTURES

Teile & Herrsche – Integer Multiplikation
Zweiter Versuch

Verbesserte Integer Multiplikation

A	B	•	C	D

	B · D
+	B · C
+	A · D
+	A · C
=	**Ergebnis**

Berechne BC + AD
Mit einer Multiplikation

Trick:
(A+B) (C+D) – AC – BD
= BC + AD

Teile & Herrsche – Integer Multiplikation Zweiter Versuch – Rekursionsgleichung

Beispiel Schnelle Multiplikation

$$T(n) = 3 \cdot T(n/2) + cn$$

Anzahl Unterprobleme

Aufwand für Aufteilen und Zusammenfügen

Größe der Unterprobleme (bestimmt Höhe des Rekursionsbaums)

- (und T(1) = const)

(n Zweierpotenz)

Teile & Herrsche – Integer Multiplikation
Zweiter Versuch – Rekursionsgleichung

Aufwand Verbesserte Integer Multiplikation

- 3 Multiplikationen der Länge n/2
- [AC, BD, (A+B) (C+D)]
- Konstant viele Additionen und Multiplikationen mit Zweierpotenzen

Laufzeit

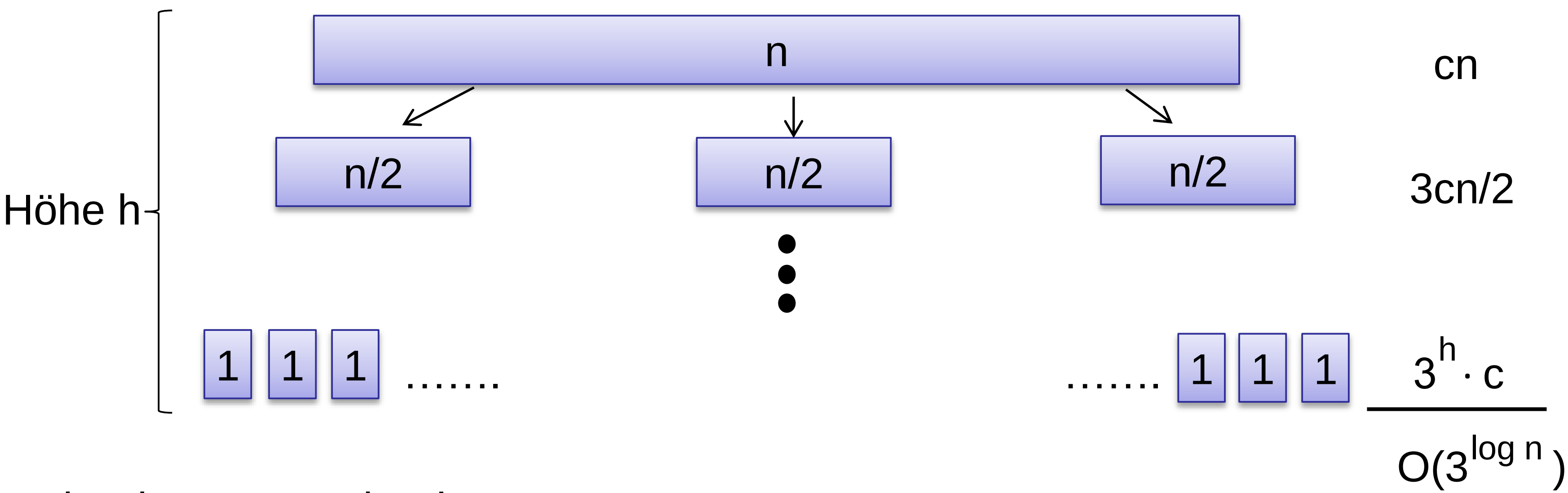
$$T(n) = \begin{cases} 3\,T(n/2) + cn & , n>1 \\ c & , n=1 \end{cases} \qquad c \text{ geeignete Konstante}$$

Teile & Herrsche – Integer Multiplikation Zweiter Versuch – Rekursionsgleichung

Laufzeit verbesserte Integer Multiplikation

$$T(n) \leq \begin{cases} 3\,T(n/2) + cn & , n>1 \\ c & , n=1 \end{cases}$$

c geeignete Konstante



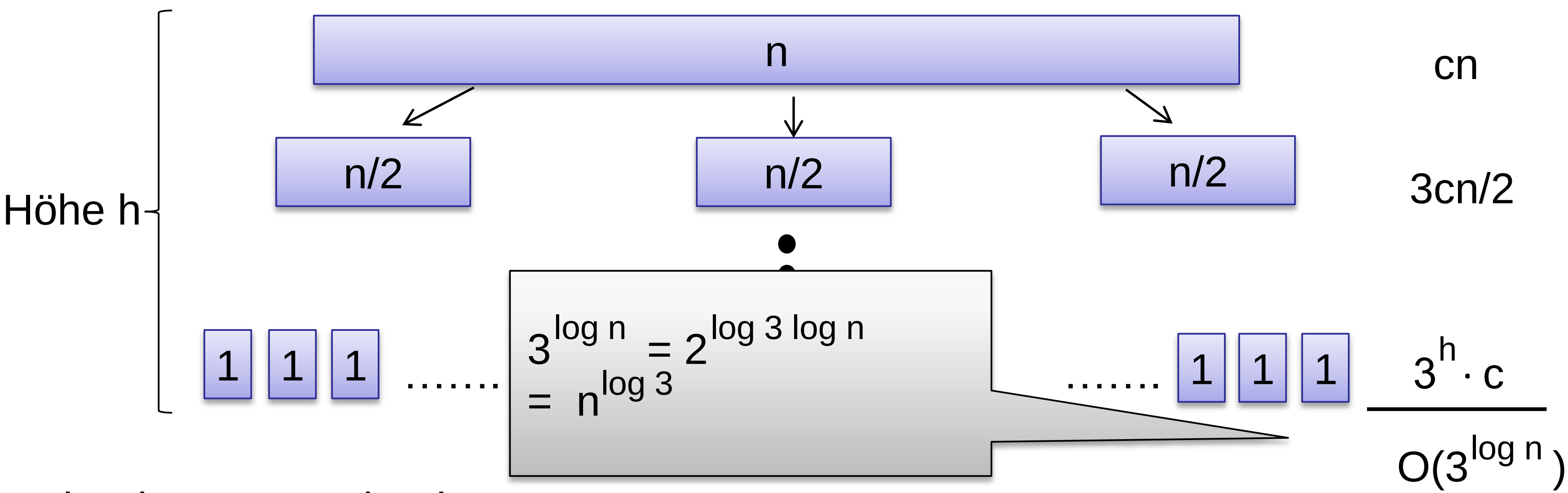
Höhe des Baums: $h = \log n$

Teile & Herrsche – Integer Multiplikation
Zweiter Versuch – Rekursionsgleichung

Laufzeit verbesserte Integer Multiplikation

$$T(n) \leq \begin{cases} 3\,T(n/2) + cn & , n>1 \\ c & , n=1 \end{cases}$$

c geeignete Konstante



Höhe des Baums: h = log n

Teile & Herrsche – Integer Multiplikation Zweiter Versuch – Laufzeit

Satz

- Die Laufzeit der verbesserten Integer Multiplikation ist

$$O(3^{\log n}) = O(n^{\log 3}) = O(n^{1.59}).$$

Beweis

- Per Induktion über n. Zu zeigen ist, dass $T(n) \leq c\, 3^{\log n} - 2cn$.

Teile & Herrsche
Allgemeine Rekursionsgleichung

Laufzeiten als Rekursionsgleichung in der Form

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

Anzahl Unterprobleme

Aufwand für das Aufteilen und Zusammensetzen

Größe der Unterprobleme (bestimmt Höhe des Rekursionsbaums)

- wobei T(1) konstant ist

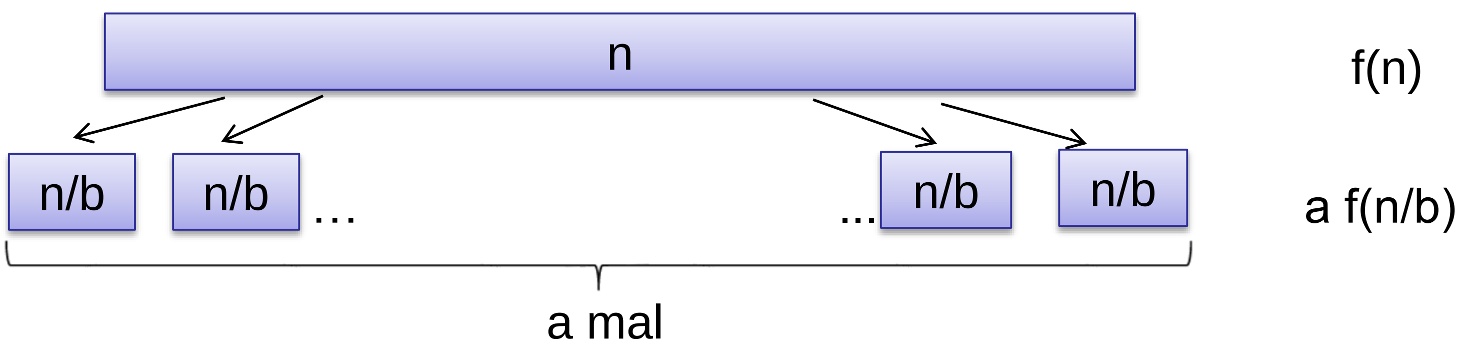
Es gibt das Mastertheorem was eine Aussage über die Laufzeit für viele Fälle erlaubt. Wir betrachten es kurz.

Teile & Herrsche
Allgemeine Rekursionsgleichung

Laufzeiten als Rekursionsgleichung in der Form

$f(n) = O(n^k)$ für Konstante k.

$$T(n) = \begin{cases} a\,T(n/b) + f(n) & , n>1 \\ 1 & , n=1 \end{cases}$$

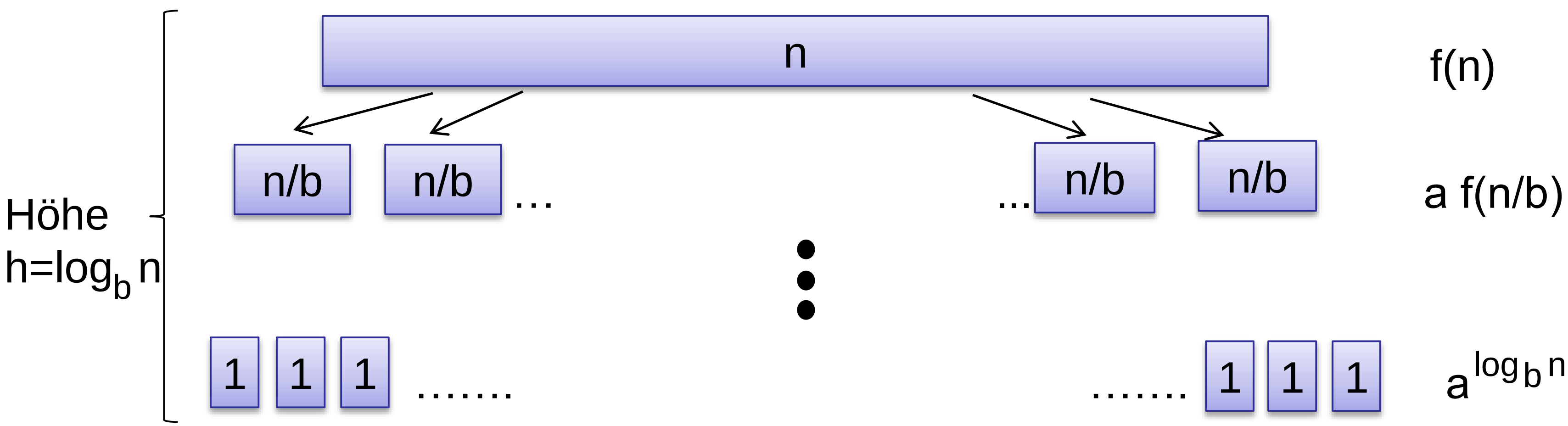


Teile & Herrsche
Allgemeine Rekursionsgleichung

Laufzeiten als Rekursionsgleichung in der Form

$f(n) = O(n^k)$ für Konstante k.

$$T(n) = \begin{cases} a\,T(n/b) + f(n) & , n>1 \\ 1 & , n=1 \end{cases}$$



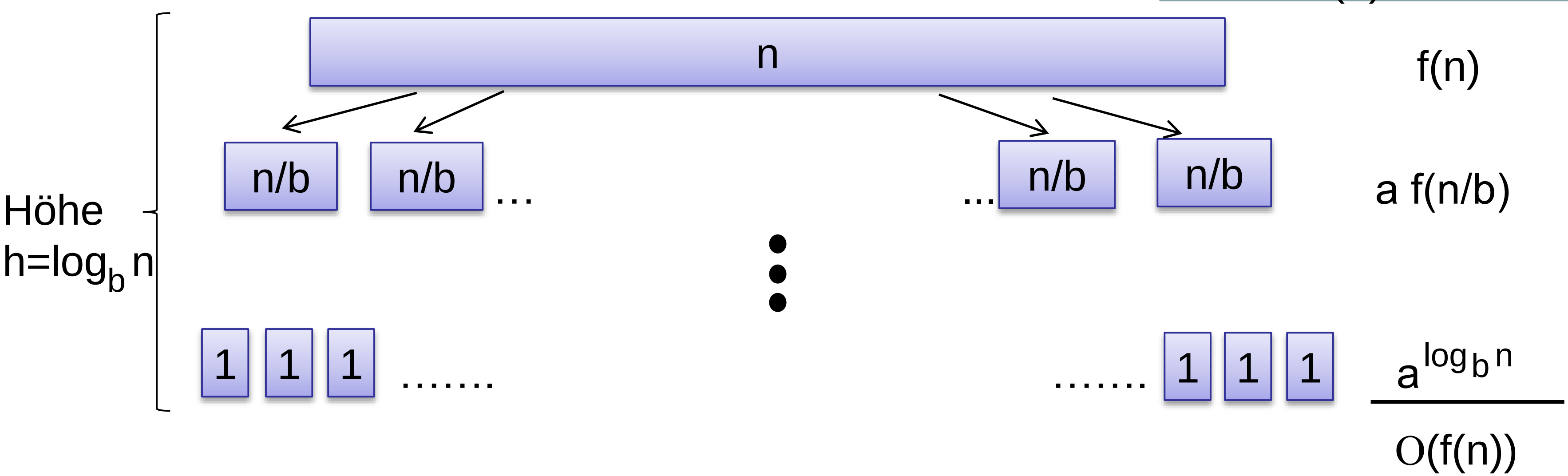
Teile & Herrsche – Mastertheorem
Abhängigkeit vom dominanten Laufzeitfaktor

Laufzeiten als Rekursionsgleichung in der Form

f(n) = O(n^k) für Konstante k.

$$T(n) = \begin{cases} a\,T(n/b) + f(n) & , n>1 \\ 1 & , n=1 \end{cases}$$

Fall 1:
Laufzeit
Zusammensetzen,
d.h. f(n) dominiert



INTERNET NETWORK ARCHITECTURES

Teile & Herrsche – Mastertheorem Abhängigkeit vom dominanten Laufzeitfaktor

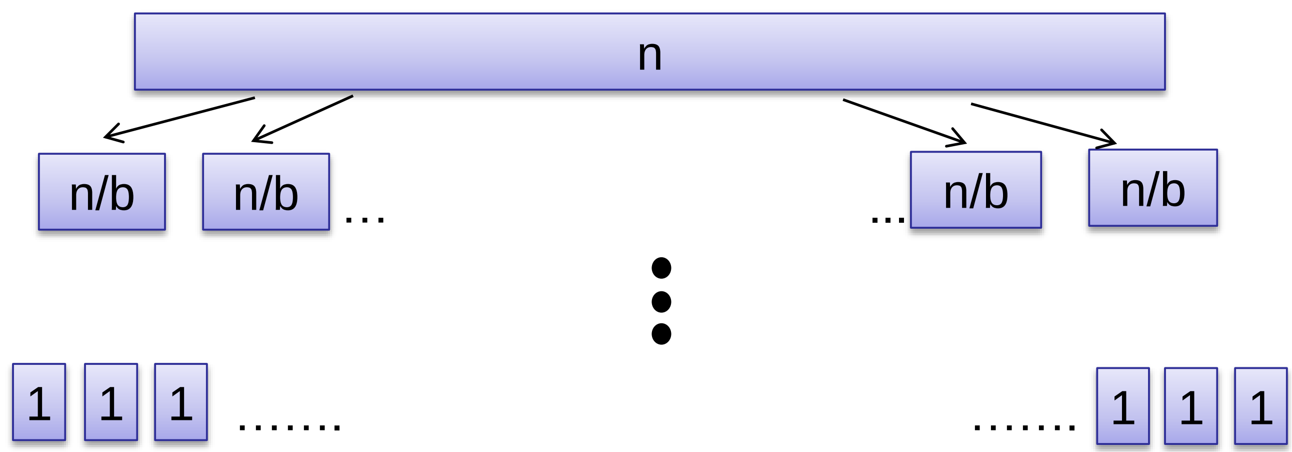
Laufzeiten als Rekursionsgleichung in der Form

$f(n) = O(n^k)$ für Konstante k.

$$T(n) = \begin{cases} a\,T(n/b) + f(n) & , n>1 \\ 1 & , n=1 \end{cases}$$

Fall 2:
Laufzeit auf unterster Rekursionsstufe dominiert

Höhe $h=\log_b n$



$f(n)$

$a\,f(n/b)$

$$\frac{a^{\log_b n}}{O(n^{\log_b a})}$$

Teile & Herrsche – Mastertheorem
Abhängigkeit vom dominanten Laufzeitfaktor

Laufzeiten als Rekursionsgleichung in der Form

$f(n) = O(n^k)$ für Konstante k.

Fall 3:
Laufzeiten
Zusammensetzen und
Rekursion vergleichbar

$$T(n) = \begin{cases} a\,T(n/b) + f(n) & , n>1 \\ 1 & , n=1 \end{cases}$$