technische universität dortmund

LS 2 / Informatik

Datenstrukturen, Algorithmen und Programmierung 2 (DAP2)

Organisatorisches zum 1. Mai

Ausweichtermine am 3. Mai:

- Fr 8-10 (2), Fr. 10-12 (2), Fr 14-16
- (siehe auch Praktikumswebseite)

- Ich möchte hier auch nochmal an die Anwesenheitspflicht im Praktikum erinnern!
- Falls Sie aus wichtigen Gründen nicht an Ihrer Übung teilnehmen können, informieren Sie Ihren Gruppenleiter vorher, an welchem Ausweichtermin Sie anwesend sein werden

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)

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

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

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

Schritt 2:
Rekursiv Sortieren

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**	**25**

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

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

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** $b \le A[q]$ **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. 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 $\leftarrow \lfloor (p+r)/2 \rfloor$
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**

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=1 $\qquad\qquad$ 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 $\le$ 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=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. 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

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

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=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 $\le$ 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

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. $q \leftarrow \lfloor (p+r)/2 \rfloor$
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

Satz 6

- 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

Satz 6

- 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

Satz 6

- 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

Satz 6

- 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.

Teile & Herrsche

Satz 6

- 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

Satz 6

- 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

Satz 6

- 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

Satz 6

- 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].

Teile & Herrsche

Satz 6

- 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

Satz 6

- 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

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:

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

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

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:
1
1
1

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. $q \leftarrow \lfloor (p+r)/2 \rfloor$	1
4. **if** $b \le A[q]$ **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

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

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** $b \le A[q]$ **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

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:

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

	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 $\le$ 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

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

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)

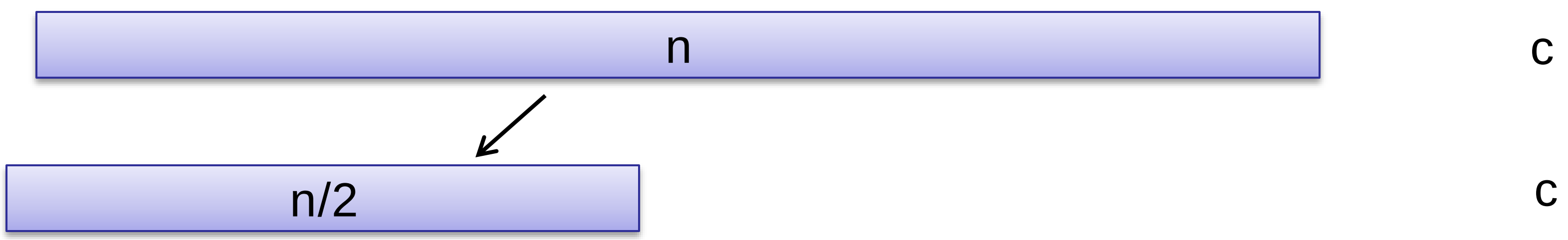
Teile & Herrsche

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

n	c

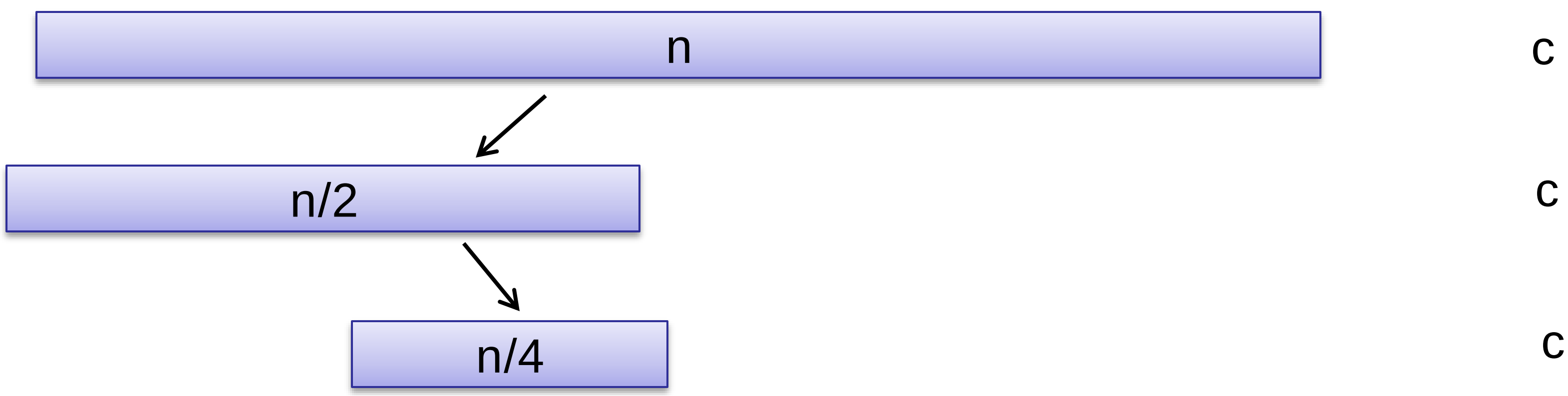
Teile & Herrsche

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



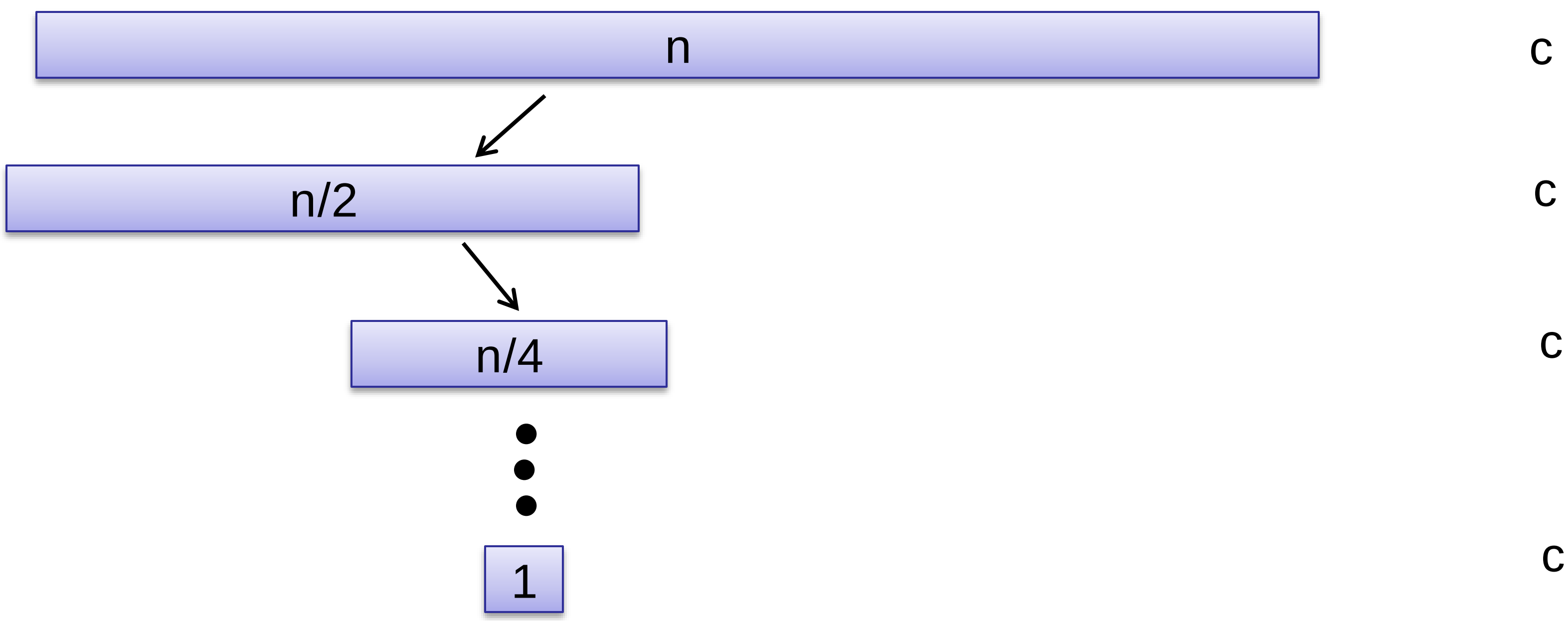
Teile & Herrsche

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



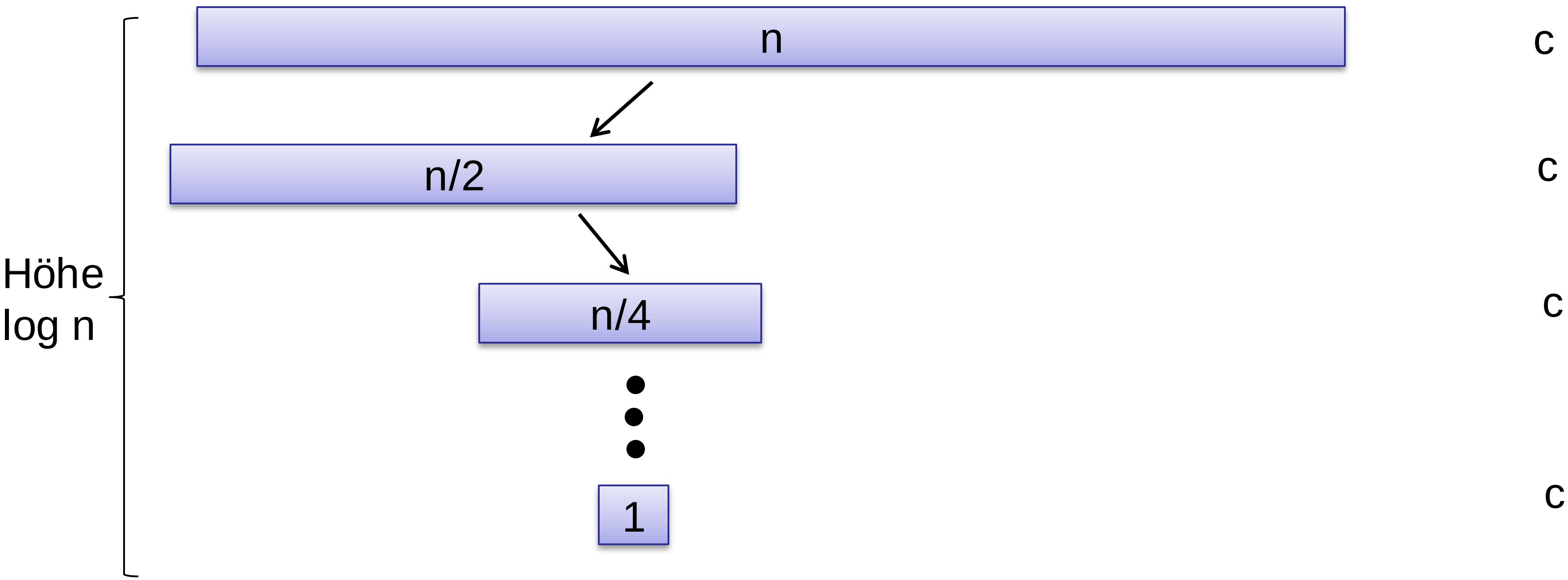
Teile & Herrsche

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



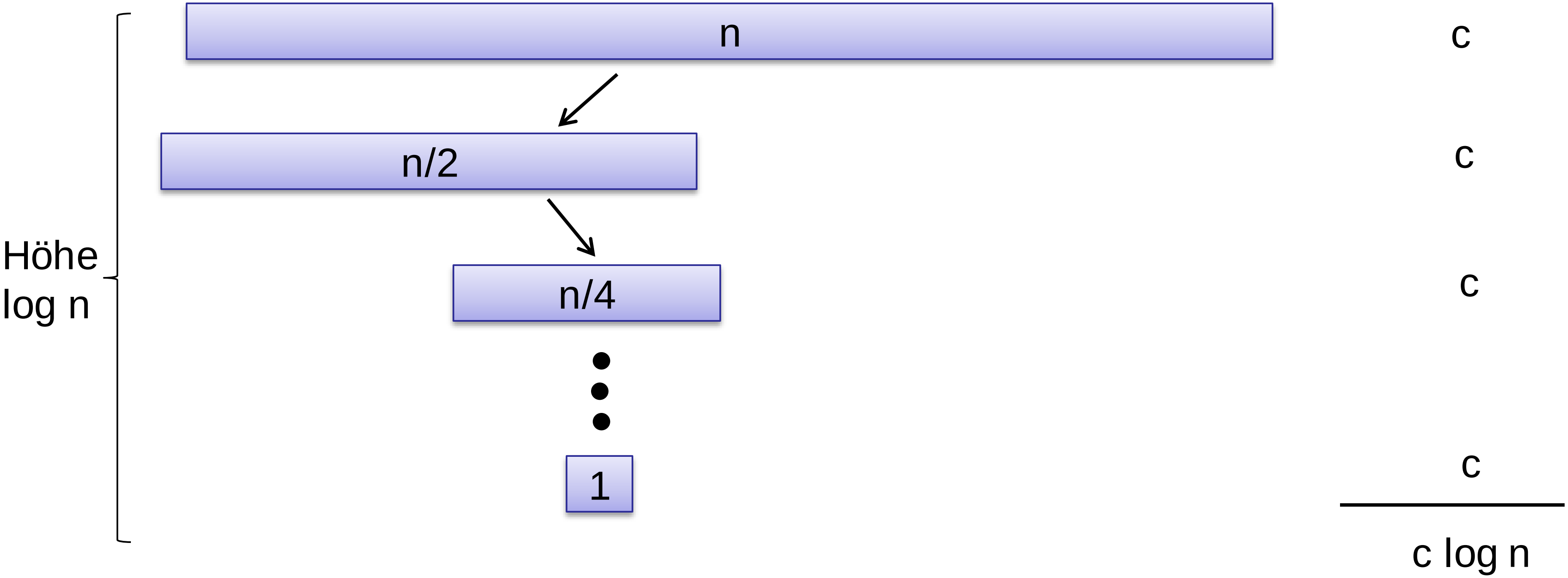
Teile & Herrsche

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

Beweis

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

Teile & Herrsche

Satz 7

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

Beweis

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

Teile & Herrsche

Satz 7

- Algorithmus 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 $T(m) \leq 5\lceil \log m \rceil + 1$.

Teile & Herrsche

Satz 7

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

Beweis

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

Teile & Herrsche

Satz 7

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

Beweis

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

Teile & Herrsche

Satz 7

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

Beweis

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

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

Teile & Herrsche

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

Teile & Herrsche

Schulmethode: (13·11)

Teile & Herrsche

Schulmethode: (13·11)

$$\underline{\;1101 \cdot 1011\;}$$

Teile & Herrsche

Schulmethode: (13·11)

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

Teile & Herrsche

Schulmethode: (13·11)

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

Teile & Herrsche

Schulmethode: *(13·11)*

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

Teile & Herrsche

Schulmethode: (13·11)

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

Teile & Herrsche

Schulmethode: (13·11)

```
 1101 · 1011
 -----------
        1101
       1101
     1101
 -----------
    10001111
```

Teile & Herrsche

Laufzeit Schulmethode

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

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

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

Teile & Herrsche

Laufzeit Schulmethode

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

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

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

Bessere Laufzeit mit Teile & Herrsche?

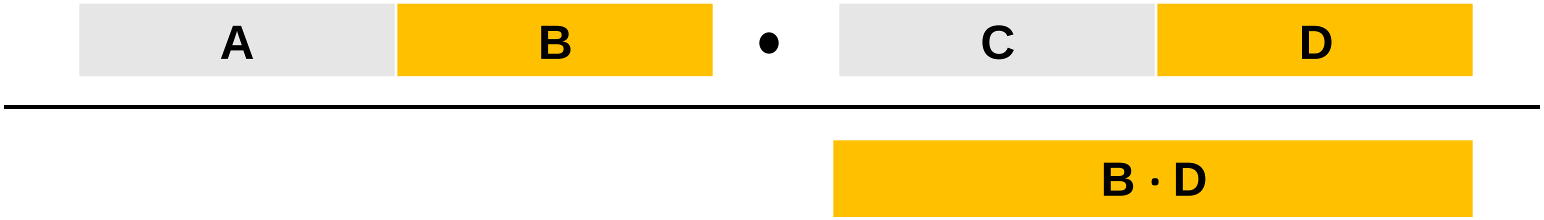
Teile & Herrsche

Integer Multiplikation

A	B	•	C	D

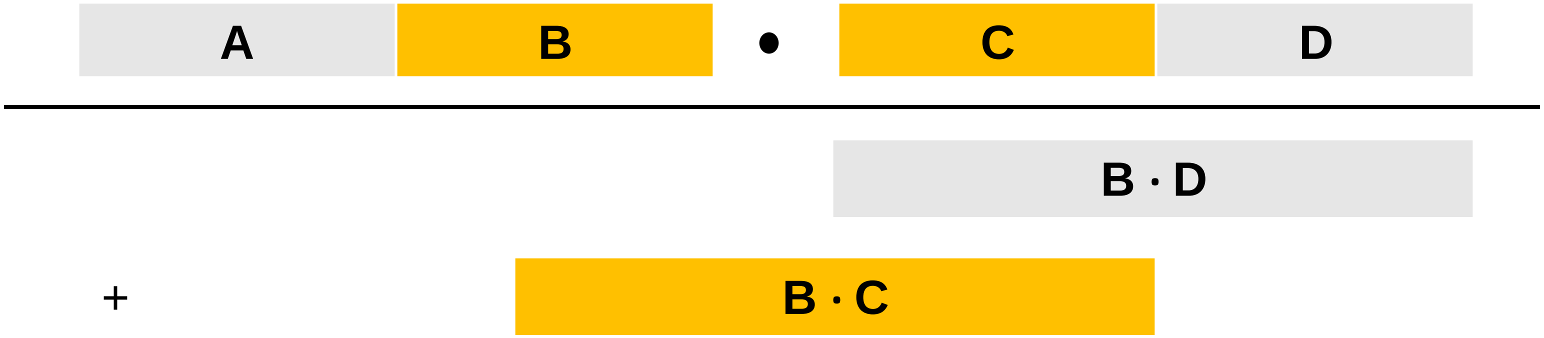
Teile & Herrsche

Integer Multiplikation



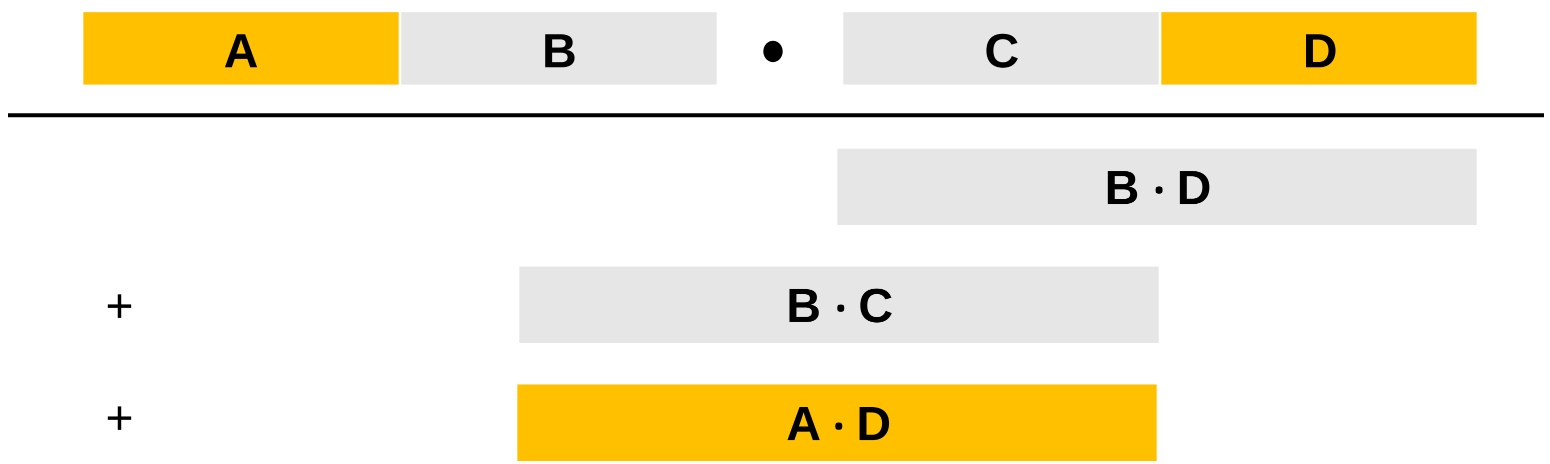
Teile & Herrsche

Integer Multiplikation



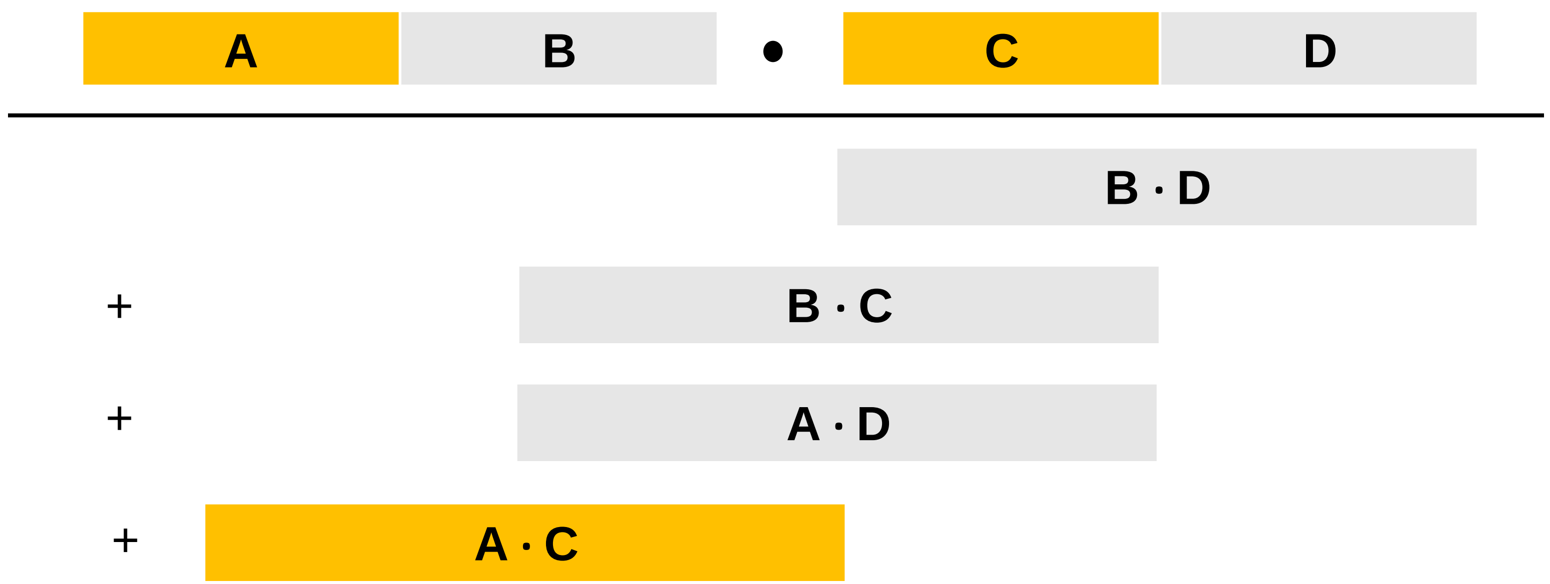
Teile & Herrsche

Integer Multiplikation



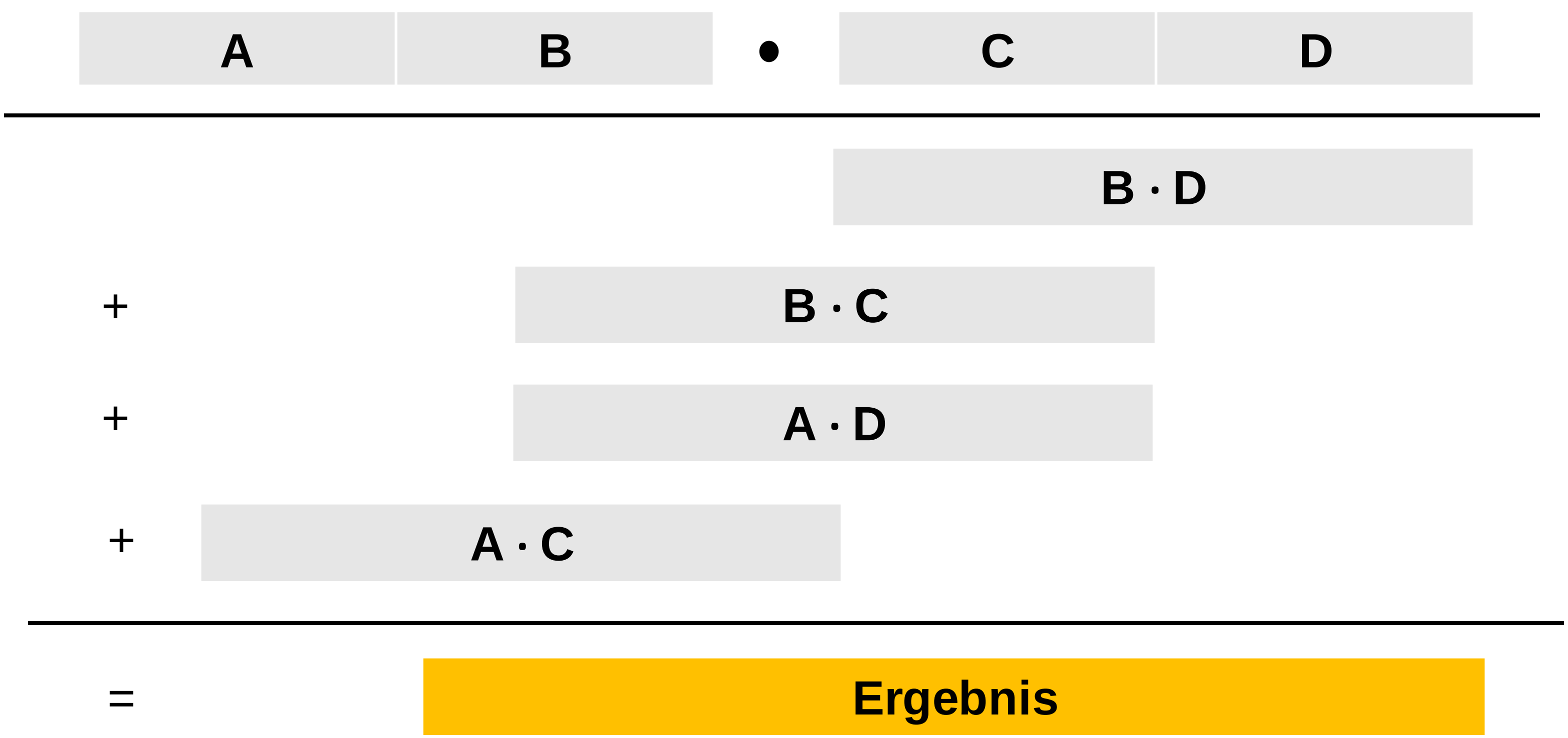
Teile & Herrsche

Integer Multiplikation



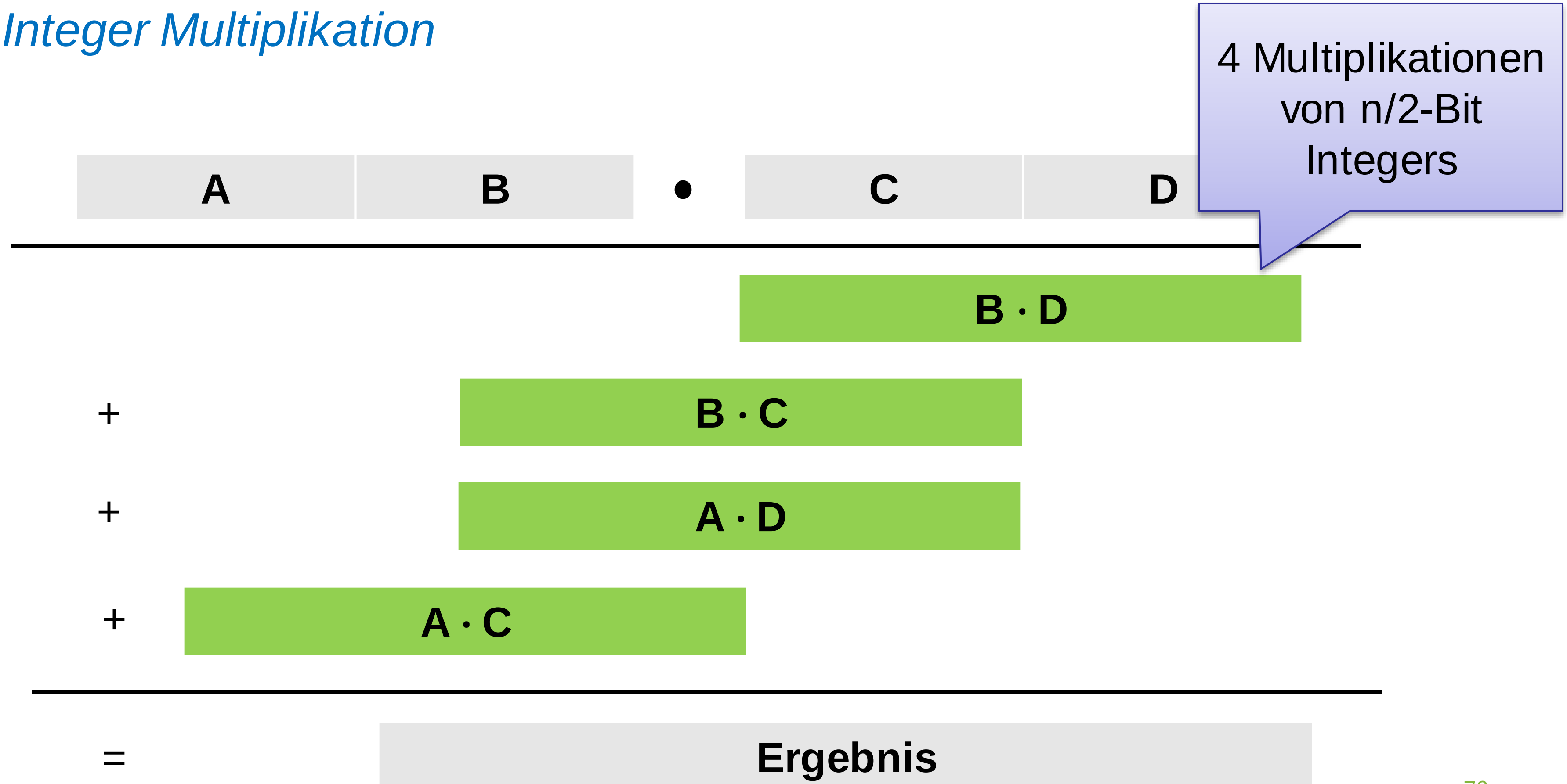
Teile & Herrsche

Integer Multiplikation



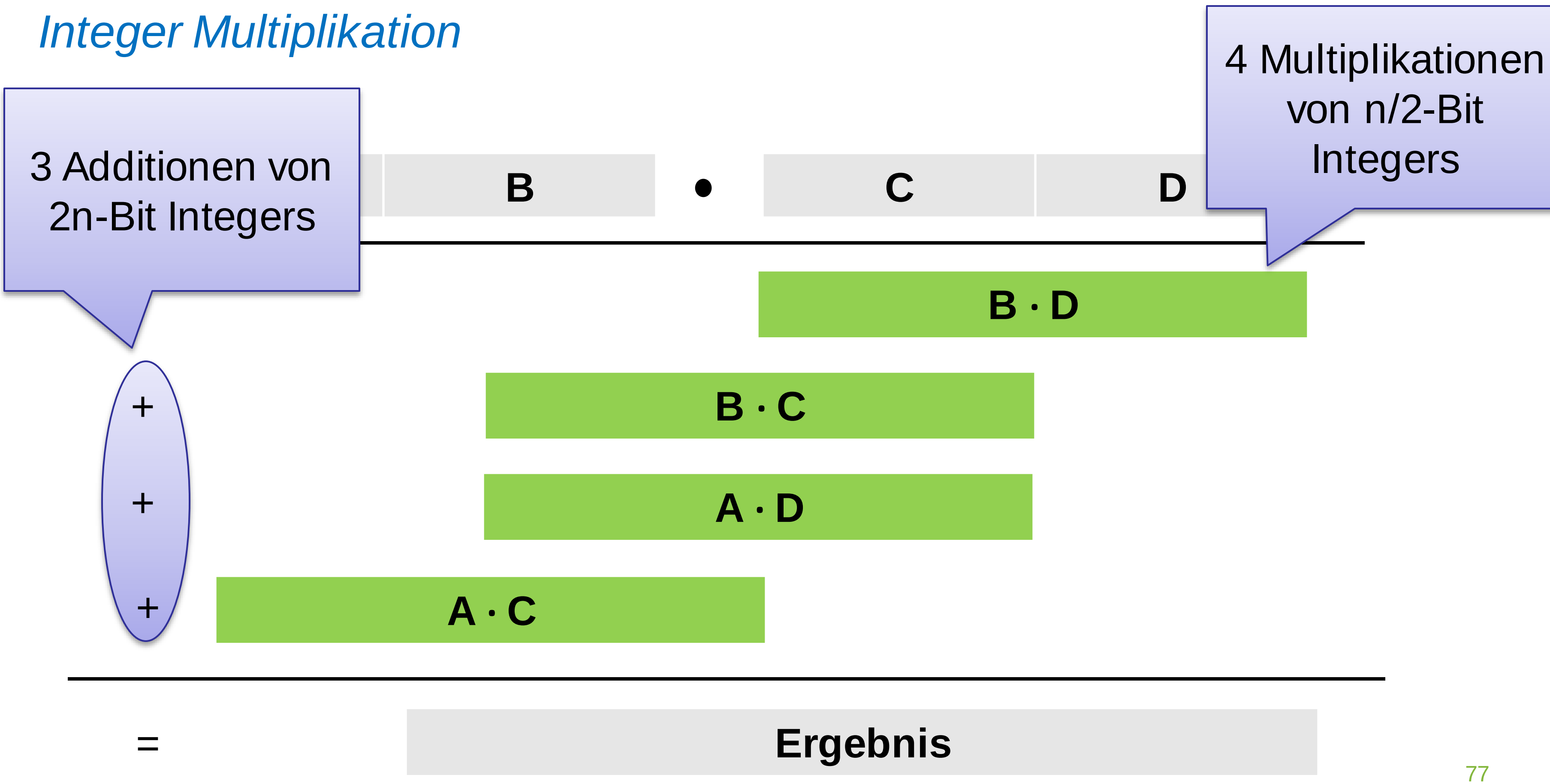
Teile & Herrsche

Integer Multiplikation



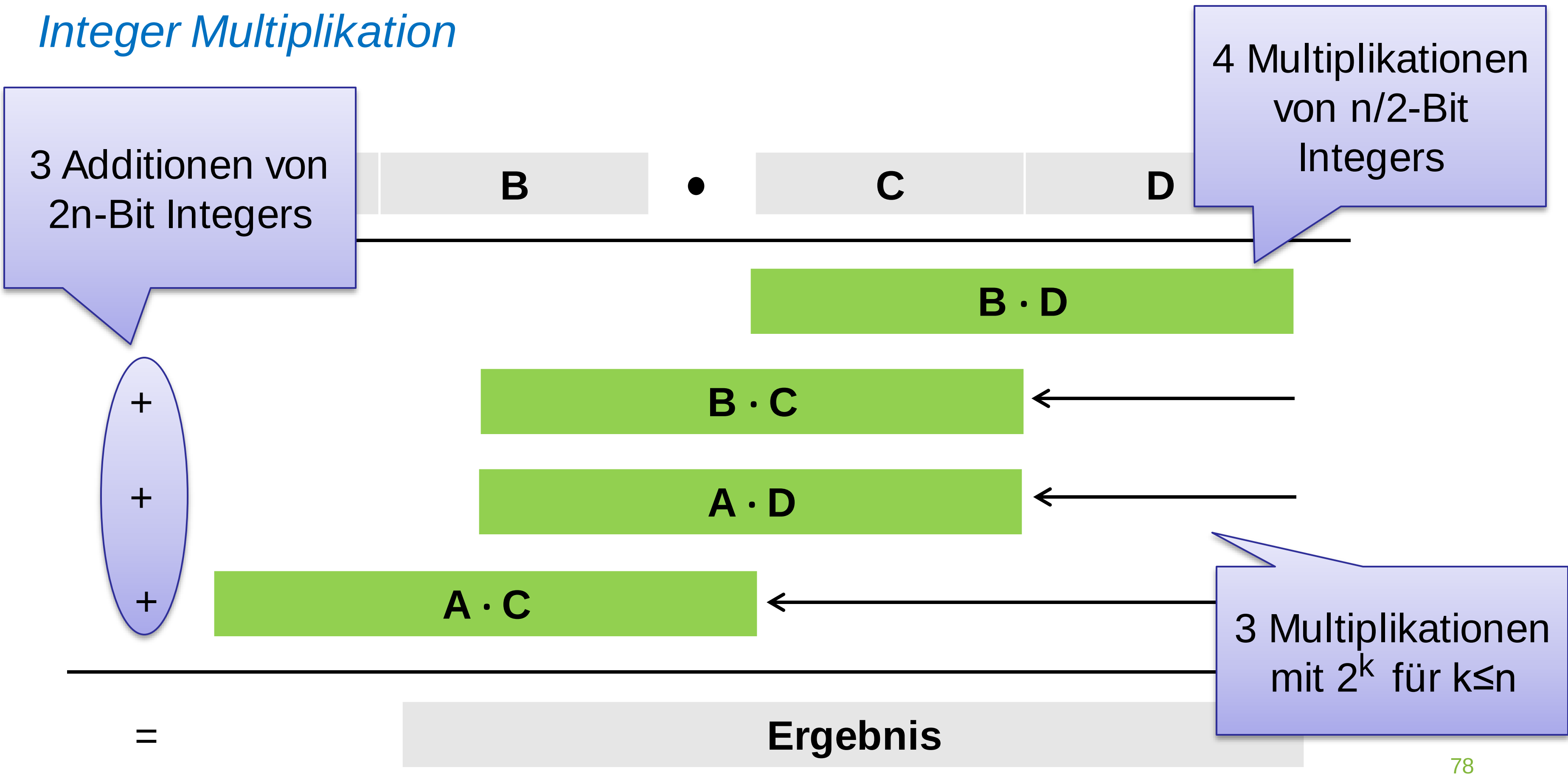
Teile & Herrsche

Integer Multiplikation



Teile & Herrsche

Integer Multiplikation



Teile & Herrsche

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

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

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

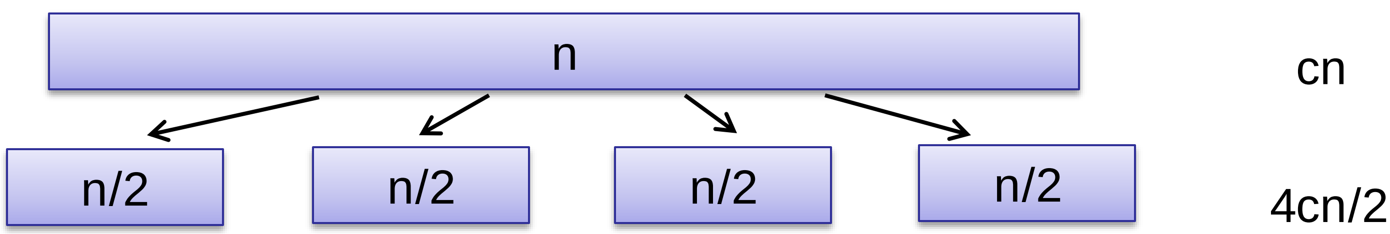
cn

Teile & Herrsche

Laufzeit einfaches Teile & Herrsche

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

c geeignete Konstante

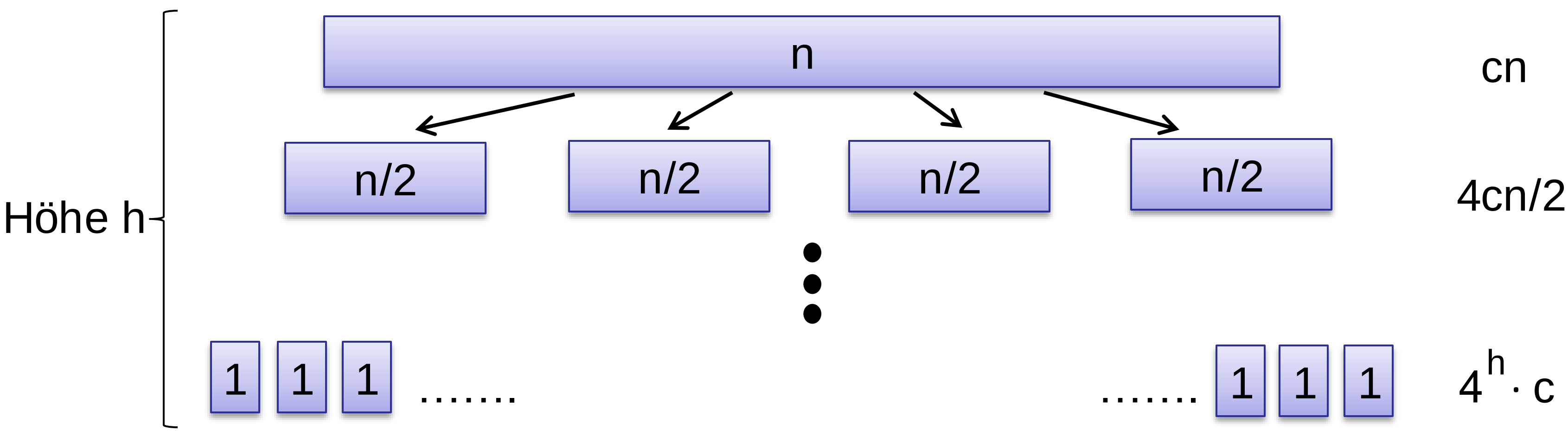


Teile & Herrsche

Laufzeit einfaches Teile & Herrsche

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

c geeignete Konstante

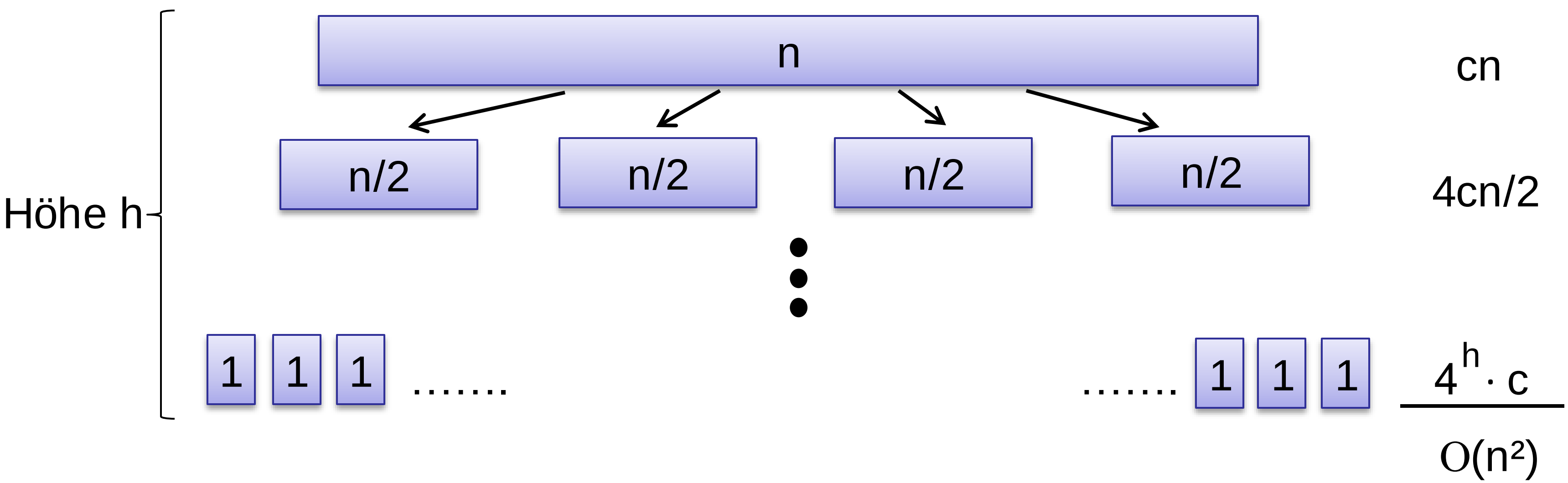


Teile & Herrsche

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

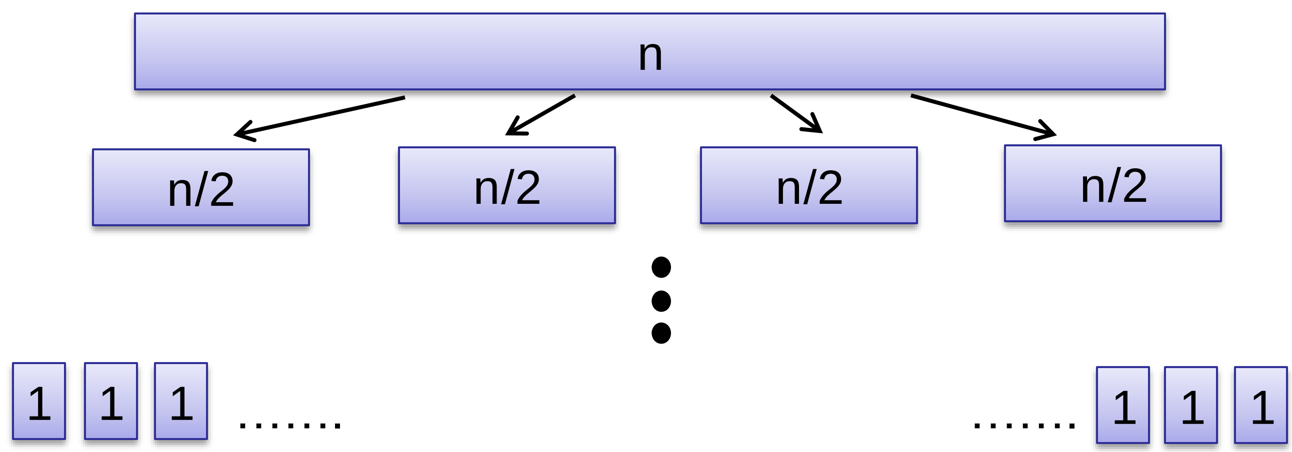
Teile & Herrsche

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 h



cn

4cn/2

$4^h \cdot c$

O(n²)

Nichts gewonnen!

Höhe des Baums: h = log n

Teile & Herrsche

Satz 8

Die Multiplikation zweier n-Bit Zahlen mit dem einfachen Teile & Herrsche Verfahren hat Laufzeit $O(n^2)$.

Beweis

- Wir nehmen an, dass n eine Zweierpotenz ist. Induktion über n. Wir zeigen $T(n) \le cn^2$.

Teile & Herrsche

Satz 8

Die Multiplikation zweier n-Bit Zahlen mit dem einfachen Teile & Herrsche Verfahren hat Laufzeit $O(n^2)$.

Beweis

- Wir nehmen an, dass n eine Zweierpotenz ist. Induktion über n. Wir zeigen $T(n) \le cn^2$.
- (I.A.) Die Laufzeit zur Multiplikation von zwei 1-Bit Zahlen ist höchstens c.

Teile & Herrsche

Satz 8

Die Multiplikation zweier n-Bit Zahlen mit dem einfachen Teile & Herrsche Verfahren hat Laufzeit $O(n^2)$.

Beweis

- Wir nehmen an, dass n eine Zweierpotenz ist. Induktion über n. Wir zeigen $T(n) \le cn^2$.
- (I.A.) Die Laufzeit zur Multiplikation von zwei 1-Bit Zahlen ist höchstens c.
- (I.V.) Für jedes $m<n$ ist die Laufzeit zur Multiplikation von zwei m-Bit Zahlen cm^2.

Teile & Herrsche

Satz 8

Die Multiplikation zweier n-Bit Zahlen mit dem einfachen Teile & Herrsche Verfahren hat Laufzeit $O(n^2)$.

Beweis

- Wir nehmen an, dass n eine Zweierpotenz ist. Induktion über n. Wir zeigen $T(n) \le cn^2$.
- (I.A.) Die Laufzeit zur Multiplikation von zwei 1-Bit Zahlen ist höchstens c.
- (I.V.) Für jedes m<n ist die Laufzeit zur Multiplikation von zwei m-Bit Zahlen cm^2.
- (I.S.) Betrachte eine Multiplikation von zwei n-Bit Zahlen. Es gilt $T(n) \le 4\,T(n/2) + cn$. Nach (I.V.) gilt dann $T(n) \le 4\,c\,(n/2)^2 + cn = cn^2+cn$.

Teile & Herrsche

Satz 8

Die Multiplikation zweier n-Bit Zahlen mit dem einfachen Teile & Herrsche Verfahren hat Laufzeit $O(n^2)$.

Beweis

- Wir nehmen an, dass n eine Zweierpotenz ist. Induktion über n. Wir zeigen $T(n) \le cn^2$.
- (I.A.) Die Laufzeit zur Multiplikation von zwei 1-Bit Zahlen ist höchstens c.
- (I.V.) Für jedes m<n ist die Laufzeit zur Multiplikation von zwei m-Bit Zahlen c m^2.
- (I.S.) Betrachte eine Multiplikation von zwei n-Bit Zahlen. Es gilt $T(n) \le 4\,T(n/2) + cn$. Nach (I.V.) gilt dann $T(n) \le 4\,c\,(n/2)^2 + cn = \boxed{cn^2+cn.}$

Funktioniert nicht !!!!!

Teile & Herrsche

Satz 8

Die Multiplikation zweier n-Bit Zahlen mit dem einfachen Teile & Herrsche Verfahren hat Laufzeit $O(n^2)$.

Beweis (neuer Versuch)

- Wir nehmen an, dass n eine Zweierpotenz ist. Induktion über n. O.b.d.A. sei $c \geq T(2)$. Wir zeigen $\boxed{T(n) \leq cn^2 - cn.}$

Trick: Die Funktion etwas verkleinern!!

Teile & Herrsche

Satz 8

Die Multiplikation zweier n-Bit Zahlen mit dem einfachen Teile & Herrsche Verfahren hat Laufzeit $O(n^2)$.

Beweis (neuer Versuch)

- Wir nehmen an, dass n eine Zweierpotenz ist. Induktion über n. O.b.d.A. sei $c \geq T(2)$. Wir zeigen $T(n) \leq cn^2 - cn$.
- (I.A.) Die Laufzeit zur Multiplikation von zwei 2-Bit Zahlen ist höchstens $T(2) \leq c \leq 2c$.

Teile & Herrsche

Satz 8

Die Multiplikation zweier n-Bit Zahlen mit dem einfachen Teile & Herrsche Verfahren hat Laufzeit $O(n^2)$.

Beweis (neuer Versuch)

- Wir nehmen an, dass n eine Zweierpotenz ist. Induktion über n. O.b.d.A. sei $c \geq T(2)$. Wir zeigen $T(n) \leq cn^2 - cn$.
- (I.A.) Die Laufzeit zur Multiplikation von zwei 2-Bit Zahlen ist höchstens $T(2) \leq c \leq 2c$.
- (I.V.) Für jedes $m<n$ ist die Laufzeit zur Multiplikation von zwei m-Bit Zahlen $c\, m^2 - cm$.

Teile & Herrsche

Satz 8

Die Multiplikation zweier n-Bit Zahlen mit dem einfachen Teile & Herrsche Verfahren hat Laufzeit $O(n^2)$.

Beweis (neuer Versuch)

- Wir nehmen an, dass n eine Zweierpotenz ist. Induktion über n. O.b.d.A. sei $c \geq T(2)$. Wir zeigen $T(n) \leq cn^2 - cn$.
- (I.A.) Die Laufzeit zur Multiplikation von zwei 2-Bit Zahlen ist höchstens $T(2) \leq c \leq 2c$.
- (I.V.) Für jedes $m<n$ ist die Laufzeit zur Multiplikation von zwei m-Bit Zahlen $c\, m^2 - cm$.
- (I.S.) Betrachte eine Multiplikation von zwei n-Bit Zahlen. Es gilt $T(n) \leq 4\, T(n/2) + cn$.

Teile & Herrsche

Satz 8

Die Multiplikation zweier n-Bit Zahlen mit dem einfachen Teile & Herrsche Verfahren hat Laufzeit $O(n^2)$.

Beweis (neuer Versuch)

- Wir nehmen an, dass n eine Zweierpotenz ist. Induktion über n. O.b.d.A. sei $c \geq T(2)$. Wir zeigen $T(n) \leq cn^2 - cn$.
- (I.A.) Die Laufzeit zur Multiplikation von zwei 2-Bit Zahlen ist höchstens $T(2) \leq c \leq 2c$.
- (I.V.) Für jedes $m<n$ ist die Laufzeit zur Multiplikation von zwei m-Bit Zahlen $c\, m^2 - cm$.
- (I.S.) Betrachte eine Multiplikation von zwei n-Bit Zahlen. Es gilt $T(n) \leq 4\, T(n/2) + cn$.
  Nach (I.V.) gilt dann $T(n) \leq 4\, c\, (n/2)^2 - 4\, c(n/2) + cn = cn^2 - cn$.

Teile & Herrsche

Satz 8

Die Multiplikation zweier n-Bit Zahlen mit dem einfachen Teile & Herrsche Verfahren hat Laufzeit $O(n^2)$.

Beweis (neuer Versuch)

- Wir nehmen an, dass n eine Zweierpotenz ist. Induktion über n. O.b.d.A. sei $c \geq T(2)$. Wir zeigen $T(n) \leq cn^2 - cn$.
- (I.A.) Die Laufzeit zur Multiplikation von zwei 2-Bit Zahlen ist höchstens $T(2) \leq c \leq 2c$.
- (I.V.) Für jedes $m<n$ ist die Laufzeit zur Multiplikation von zwei m-Bit Zahlen $c\, m^2 - cm$.
- (I.S.) Betrachte eine Multiplikation von zwei n-Bit Zahlen. Es gilt $T(n) \leq 4\, T(n/2) + cn$.
  Nach (I.V.) gilt dann $T(n) \leq 4\, c\, (n/2)^2 - 4\, c(n/2) + cn = cn^2 - cn$.