11. Divide-and-conquer und Rekursionsgleichungen

Divide-and-conquer:

- Problem aufteilen in Teilprobleme.
- Teilprobleme (rekursiv) lösen
- Lösungen der Teilprobleme zusammensetzen.

Typische Beispiele: Binäre Suche, Mergesort, Quicksort.

Mergesort($A[1,\dots,n]$):

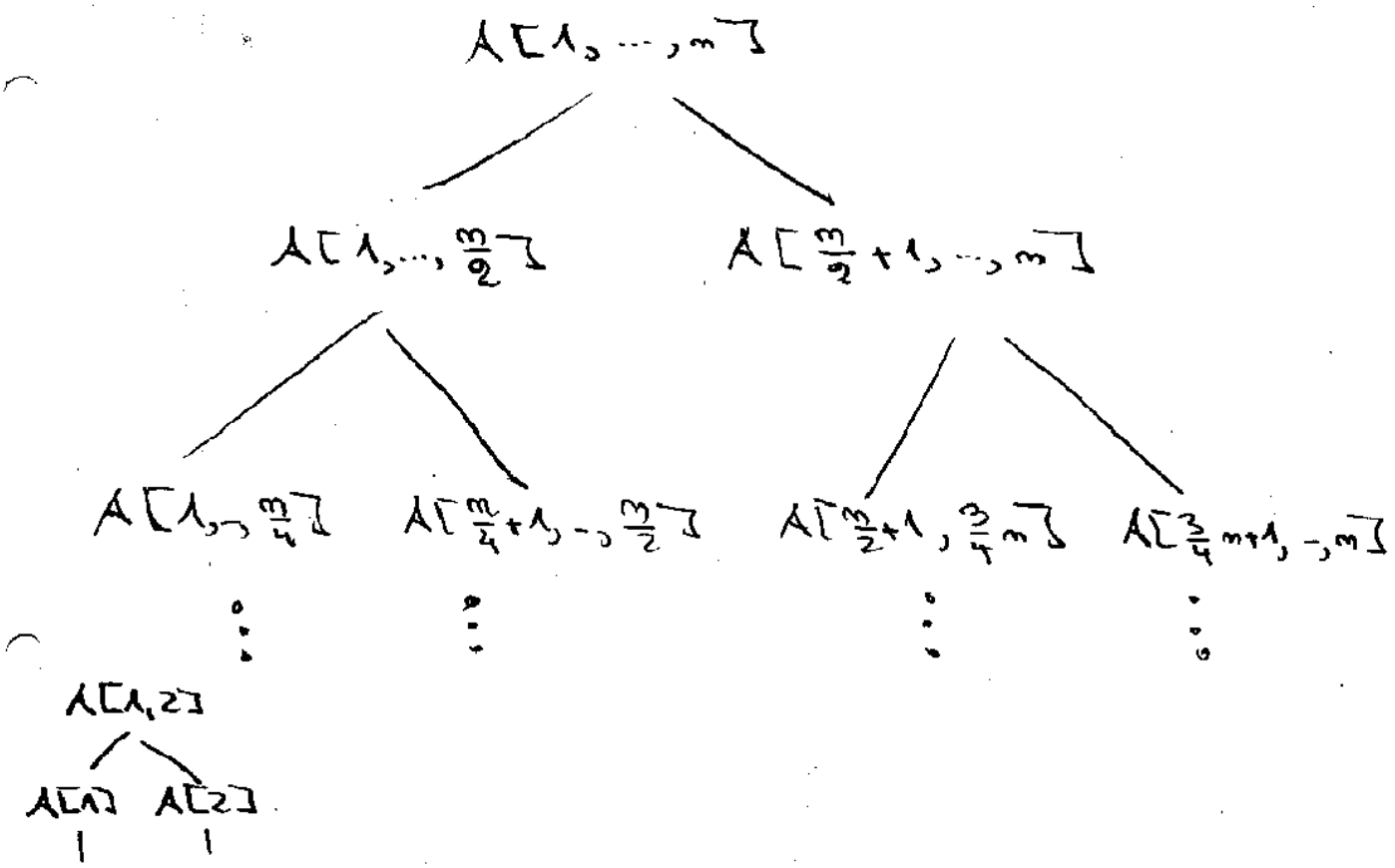
1. If $n=1$ oder $n=0$ return A
2. $B_1 := \text{Mergesort}(A[1,\dots,\lfloor \frac{n}{2} \rfloor])$;
3. $B_2 := \text{Mergesort}(A[\lfloor \frac{n}{2} \rfloor + 1,\dots,n])$;

// 2.3 divide & conquer

4. „Mischung von B_1 und B_2"

// Mischung bilden,
// compress Schritt

Aufbau bei m 2-er Potenz



Tiefe: genau $\log_2 m$.

Laufzeit:

Blätter: $\Theta(m)$

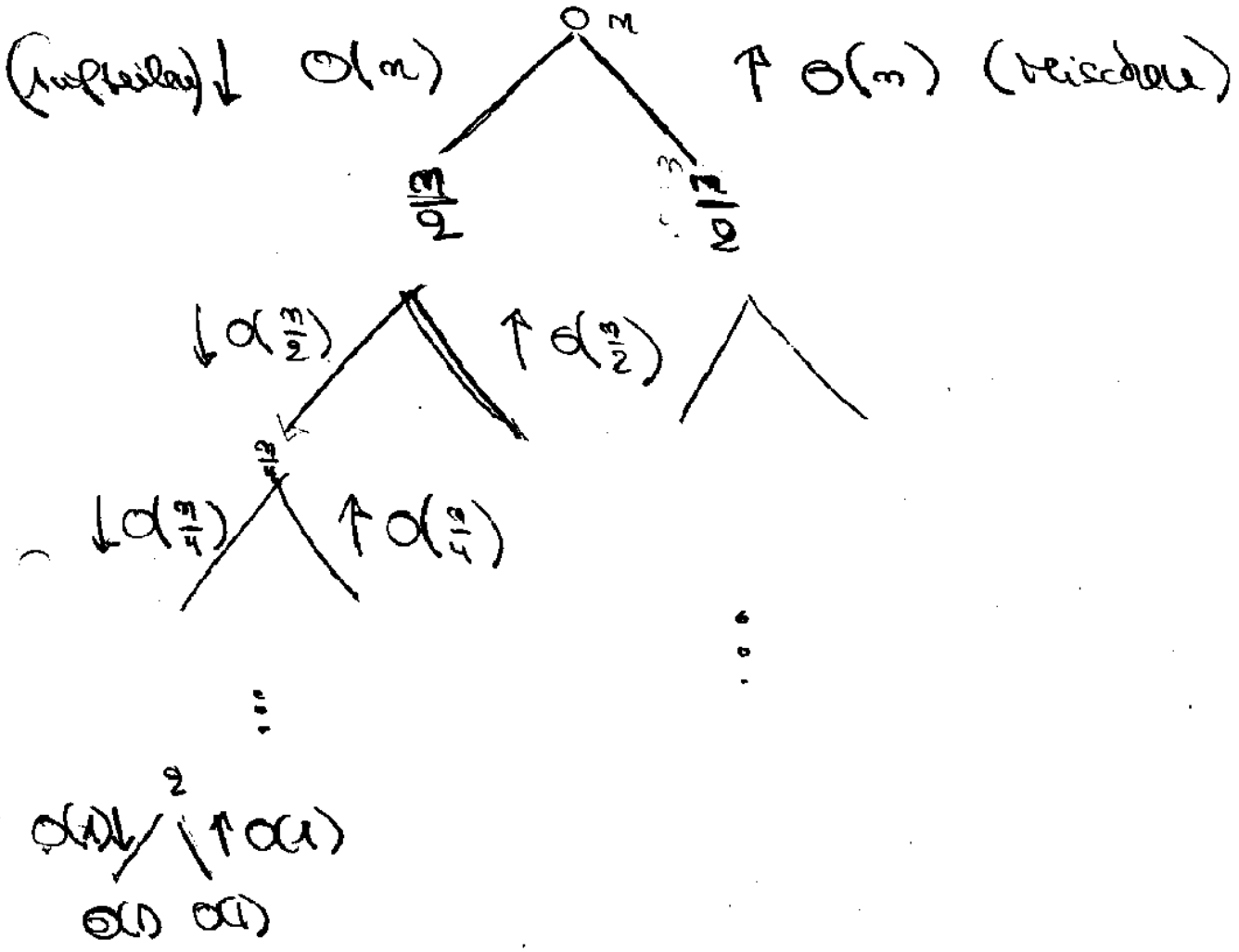
Aufteilen: bei m Elementen $\Theta(m)$

11.3

Zusammensetzen von 2 mal $\frac{n}{2}$ Elemente:

$O(n)$

Damit insgesamt:



(11.4)

Für die Blätter: $n \cdot O(1) = O(n)$

Für das Aufteilen:

$$c \cdot n + c \cdot \frac{n}{2^1} + c \cdot \frac{n}{4^1} + \dots + c \cdot 1$$

$$+ c \cdot \frac{n}{2^1} + \dots$$ Wo soll das addiert

Wir addieren in einer anderen Reihenfolge:

1 $c \cdot n +$

2 $+ c \cdot \frac{n}{2^1} + c \cdot \frac{n}{2^1} =$

3 $+ c \cdot \left(\frac{n}{4^1} + \frac{n}{4^1} + \frac{n}{4^1} + \frac{n}{4^1}\right)$

4 $+ c \cdot \left(\frac{n}{8^1} + \dots + \frac{n}{8^1}\right)$

$+$

$\log n$ $+ c \cdot (\underbrace{1 + \dots + 1}_{n\text{-mal}})$

$$= \log n \cdot c \cdot n = O(n \cdot \log n)$$

Ebenso für das Mischen

$O(n \cdot \log n)$

Insgesamt $O(n \cdot \log n) + O(n) = O(n \cdot \log n)$

- Wichtig: Richtige Rekursionsfolge beim Zusammenaddieren! Bei Bäumen oft stufenweise!
- Der eigentliche Arbeit beim divide-and-conquer geschieht im Aufteilen und im Zusammenfügen. Das Rest ist Rekursion.

- Mergesort einfach bottom-up ohne Rekursion, da starre Aufrufbaum.

  $A[1,2], A[3,4], \ldots, A[n-1, n]$ sortieren

  $A[1, \ldots, 4], A[5, \ldots, 8], \ldots$ sortieren

  usw. $O(n \cdot \log n)$.

- Ebenfalls Heapsort sortiert in $O(n \cdot \log n)$.
- Bubblesort, Insertionsort, Selectionsort $O(n^2)$

zu Quicksort

array $A[1, \ldots, n]$ of „beliebiger Datentyp"

Quicksort ($A[m, \ldots, n]$)

1. If m = ein zu sortierendes
2. a_i = ein $A[i]$;
3. Lösche $A[i]$ aus A.
2. for $j \neq 1$ to n {
3. if $A[j] \leq a$ {
   $A[j]$ als nächstes Element zu B_1;
   break
   }
4. if $A[j] > a$ {
   $A[j]$ zu B_2 }
   }

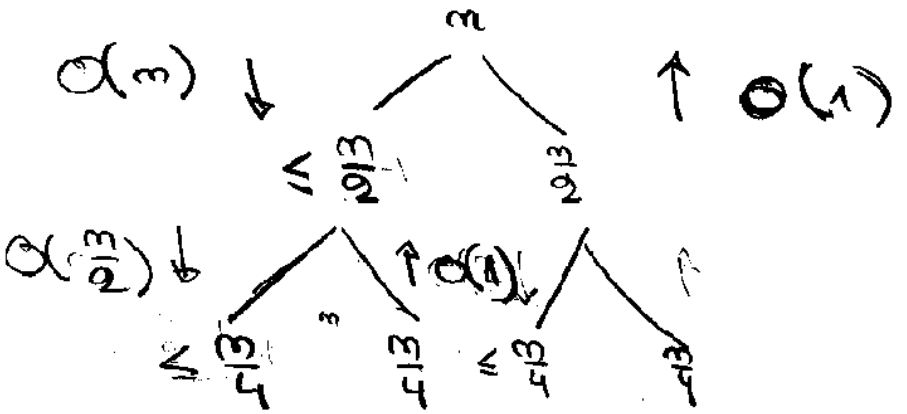
5. $A := B_1 \circ B_2$

6. if $B_1 \neq \phi$ Quicksort(B_1) // B_1 als // Teil von A

7. if $B_2 \neq \phi$ Quicksort(B_2) // B_2 als // Teil von A.

Bsp: Die Prozedur Partition aus dem 1. Semester erlaubt es mit dem Array A alleine auszukommen.

Prozedurbäume:



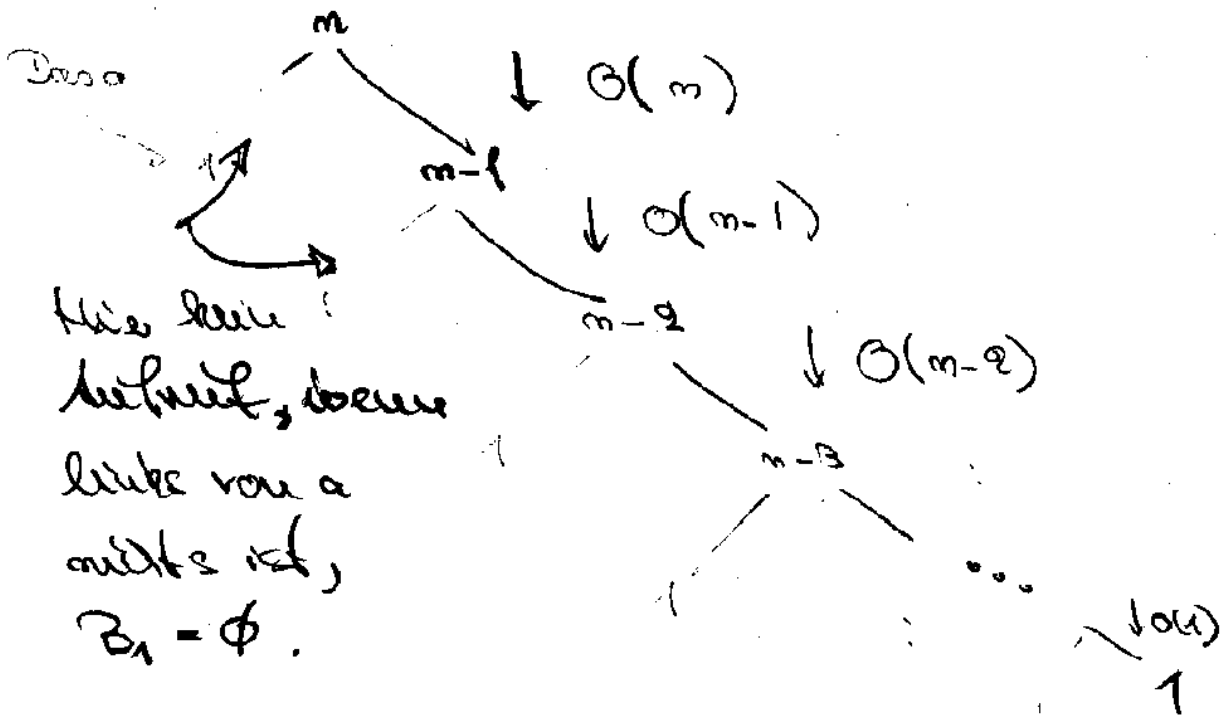
Blätter

Hier $O(n \cdot \log n)$:

$O(n) + 2 \cdot O(\frac{n}{2}) +$

$4 \cdot O(\frac{n}{4}) + \ldots + n \cdot O(1) + O(n)$

Aber auch:



Hier kann
kürzer, denn
links von a
nichts ist,
$B_1 = \phi$.

Laufzeit:

$$n + n-1 + \dots + 3+2+1 = \frac{n(n+1)}{2} = O(n^2)$$

Aufteilen.

$$1+1+\dots+1+1+1 = O(n)$$

Zusammensetzen

Also $O(n^2)$ und auch $\Omega(n^2)$.

Wenn Quicksort trotzdem in
der Praxis häufig? In der Regel
hat ein gutartiges Baum auf
und wir haben $O(n \cdot \log n)$.
Vergleiche immer mit festem a,
a in einem Register → schnelles
Vergleichen. Dagegen keine
Kosten von Mergesort höher
beide Elemente des Vergleichs neu.

Aufwandsanalyse von Quicksort vorher
nicht zu erkennen. Keine
nicht-rekursive Implementierung
wie bei Mergesort. Nur mit
Hilfe eines (Rekursions-) Kellers.

Noch einmal zu Mergesort. Wir betrachten den Fall, daß n eine Zweierpotenz ist, dann $\frac{n}{2}, \frac{n}{4}, \ldots, 1 = 2^0$ ebenfalls. Sei

$T(n) :=$ Worst-case Zeit bei $A[1,\ldots,n]$.

Dann gilt für ein geeignetes c:

$$T(n) \leq c \cdot n + 2 \cdot T\left(\frac{n}{2}\right)$$

$$T(1) \leq c$$

- Einmaliges Teilen $O(1)$
- Mischen $O(n)$
- Rekursionsverwaltung kostet $O(n)$

Betrachten wenn

$$T(n) = cn + 2 \cdot T\left(\frac{n}{2}\right)$$

$$T(1) = c$$

(1.11)

Dann durch Abwickeln der rekursiven Gleichung:

$$T(n) = cn + 2 \cdot T\left(\frac{n}{2}\right)$$

$$= cn + 2c\frac{n}{2} + 2 \cdot 2 \cdot T\left(\frac{n}{4}\right)$$

$$= c \cdot n + cn + 4 \cdot c \cdot \frac{n}{4} + 2 \cdot 2 \cdot 2 \cdot T\left(\frac{n}{8}\right)$$

$$= \vdots$$

$$= cn \cdot \log n + 2 \cdot 2 \cdots 2 \cdot T(1)$$

$$= c \cdot n \cdot \log n + c \cdot n$$

$$= \mathcal{O}(c \cdot n \log n) = \mathcal{O}(n \cdot \log n)!$$

Schleifenbeweis mit der Methode

Induktionsbeweis, daß $T(n) = O(n \cdot \log n)$

Ind.-Anf.:

$$T(1) = O(1 \cdot \log 1) = 0 \; !$$

stimmt so nicht. Also besser

direkt bei $T(2)$ an.

$$T(2) \leq d \cdot 2$$

für ein d geeignet.

Ind.-Schluß

$$T(n) = cn + 2T\left(\frac{n}{2}\right)$$

$$\leq cn + 2 \cdot d \cdot \frac{n}{2} \cdot \log \frac{n}{2}$$

$$\leq cn + 2 \cdot d \cdot \frac{n}{2} (\log n - 1)$$

$$= cn + d \cdot n \log n - d \cdot n$$

$$\leq d \cdot n \log n$$

gilt, wenn $c \geq d$ gewählt wird.

Wie sieht das bei Quicksort aus?

$T(n) :=$ worst case Zeit von Quicksort auf $A[1, \ldots, n]$.

Dann

$$T(n) \leq cn + T(n-1),$$

$$T(n) \leq cn + T(1) + T(n-2),$$

$$T(n) \leq cn + T(2) + T(n-3),$$

$$\vdots$$

$$T(n) \leq cn + T(n-2) + T(1)$$

$$T(n) \leq cn + T(n-1)$$

Also

$$T(1) \leq c$$

$$T(n) \leq c \cdot n + \max\{T(n-1), \; T(i) + T(n-i-1) \mid 1 \leq i \leq n-2\}$$

Dann $T(n) \leq d \cdot n^2$ für
d geeignet.

Ind.-Anf.: ✓

Ind.-Schritt:

$$T(n) \leq cn + \max\{d(n-1)^2\}$$
$$\cup \{di^2 + d(n-i-1)^2 \mid 1 \leq i \leq n-1\})$$

$i + n - i - 1 \leq n-1$

$$\leq cn + d(n-1)^2$$
$$\leq d \cdot n^2 \quad \text{für } d \geq c.$$

Aufgabe: Stellen Sie die Rekursionsgleichung für eine rek. Version des binären Suchens auf und schätzen sie diese bestmöglich ab.

Im folgenden betrachten wir
das Problem der Multiplikation
großer Zahlen. Bisher haben wir
angenommen: Zahlen in ein Speicherwort
⇒ arithmetische Ausdrücke in $O(1)$.
Man spricht vom uniformen Kostenmaß.
Bei größeren Zahlen kommt
es auf den Multiplikationsalgorithmus
an. Man mißt die Laufzeit
in Abhängigkeit von der # Bits,
die das Rechnen verarbeiten muß.
Das ist bei einer Zahl n etwa
$\log_2 n$ (genauer für $n \geq 1$ $\lfloor \log_2 n \rfloor + 1$).
Man mißt nicht in der Zahl selbst!

Die normale Methode 2
Zahlen zu addieren läuft auf
ein $O(m)$ Verfahren hinaus
bei Zahlen der Länge n:

$$\begin{array}{r} a_1 \dots a_m \\ + \; b_1 \dots b_m \\ \hline \dots \; a_m + b_m \end{array}$$

Multiplikation in $O(n^2)$:

$$(a_1 - a_0) \cdot (b_1 - b_0)$$

$$\begin{array}{ll} (a_1 - a_0) \cdot b_1 & O(m) \\ (a_1 - a_0) \cdot b_2 & O(m) \\ \vdots & \\ (a_1 - a_0) b_m & O(m) \end{array}$$

Rekursionsbildung

$n-1$ Additionen mit Zahlen
der Länge $\leq 2n \Rightarrow O(n^2)$!

(11.17)

Mit divide-and-conquer geht es besser! Nehmen wir zunächst einmal an, n ist eine Zweierpotenz. Dann

$$\underbrace{a_1 \dots a_n}_{a:=} = \underbrace{a_1 \dots a_{\frac{n}{2}}}_{a':=} \underbrace{a_{\frac{n}{2}+1} \dots a_n}_{a'':=}$$

$$\underbrace{b_1 \dots b_n}_{b:=} = \underbrace{b_1 \dots b_{\frac{m}{2}}}_{b':=} \underbrace{b_{\frac{m}{2}+1} \dots b_m}_{b'':=}$$

Nun ist ($\frac{n}{2}$ Nullen anhängen)

$$a \cdot b = \underbrace{\left(a' \cdot 2^{\frac{n}{2}} + a''\right)}_{a=} \underbrace{\left(b' \cdot 2^{\frac{n}{2}} + b''\right)}_{b=}$$

$$= a' \cdot b' \cdot 2^{n} + a' b'' 2^{\frac{n}{2}} + a'' b' 2^{\frac{n}{2}} + a'' b''$$

(2^{n}: n Nullen; $2^{\frac{n}{2}}$: $\frac{n}{2}$ Nullen)

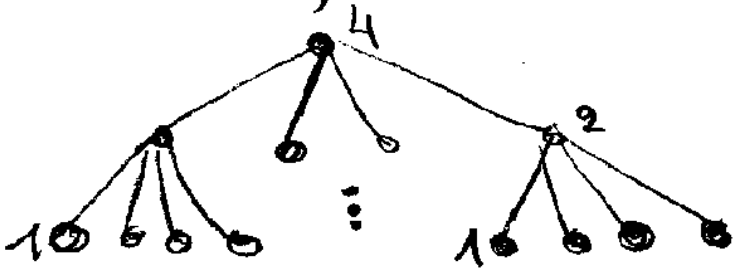
Mit drei Produkten wäre es weiter.

11.18

Das Programm sieht in etwa so aus:

```
Mult (a, b, n) { // a = a_1 ... a_n, b = b_1 ... b_n
1. if n = 1 ; return a·b // n 2-er Potenz
2. a', a'', b', b'' wie oben. // Divide
3. m_1 := Mult(a', b', n/2);
4. m_2 := Mult(a', b'', n/2);
5. m_3 := Mult(a'', b', n/2);
6. m_4 := Mult(a'', b'', n/2);
8. return (m_1 · 2^n + (m_2 + m_3) · 2^{n/2} + m_4)
                                    // Conquer
```

Aufrufbaum, $n = 4$



Tiefe $\log_2 4$.

Laufzeit T_4 bei Rekursion (0,1,n) (14.19)

braucht der divide-Schritt

und der Zeit für die Aufrufe selbst

insgesamt $\Theta(n)$. Der

conquer-Schritt auch $O(n)$.

Zählen wir wieder pro Stufe:

1. Stufe $d \cdot n$ (von $O(n)$)

2. Stufe $4 \cdot d \cdot \frac{n}{2}$

3. Stufe $4 \cdot 4 \cdot d \cdot \frac{n}{4}$

$\vdots$

$\log_2 n$'te Stufe $4^{\log_2 n - 1} \cdot d \cdot \frac{n}{4^{\log_2 n - 1}}$

Blätter $4^{\log_2 n} \cdot d$

(11.20)

Das gibt

$$T(u) \leq \sum_{i=0}^{\log_2 u} 4^i \cdot d \cdot \frac{m}{2^i}$$

$$= d \cdot m \cdot \sum_{i=0}^{\log_2 u} 2^i$$

$$= d \cdot m \cdot \frac{2^{\log_2 u + 1} - 1}{2 - 1}$$

$$= O(m^2) \quad \text{mit} \quad \sum_{i=0}^{k} x^i = \frac{x^{k+1} - 1}{x - 1}$$

für alle $x \neq 1$ ($x > 0, x < 0$, egal!),

geometrische Reihe.