

## Lösungen von Übungsblatt 2

### Algorithmen (WS 2018, Ulrike von Luxburg)

#### Lösungen zu Aufgabe 1

Btw: Schöne Schreibweise für den Fall  $d \neq \log_b(a)$  ist auch:  $T(n) \in \mathcal{O}(n^{\max\{d, \log_b(a)\}})$ .

- a)  $a = 4, b = 4, d = 1$ .  $\log_4 4 = 1 = d \Rightarrow T(n) \in \mathcal{O}(n \log(n))$   
 b)  $a = 1, b = 42, d = 1/3, \log_{42} 1 = 0 < d \Rightarrow T(n) \in \mathcal{O}(\sqrt[3]{n})$   
 c)  $a = 10^{11}, b = 10^3, d = e, \log_b a = 11/3 > d \Rightarrow T(n) \in \mathcal{O}(n^{11/3})$

#### Lösungen zu Aufgabe 2

- (a) Zeigen Sie, dass für  $c \geq 1$  die folgende Rekursion  $T(n) \in \Theta(n^{c+1})$  ist.

$$\begin{aligned} T(1) &= 0 \\ T(n) &= T(n-1) + n^c. \end{aligned}$$

Wir haben

$$T(n) = T(n-1) + n^c = T(n-2) + (n-1)^c + n^c \stackrel{?}{=} \sum_{i=0}^{n-2} (n-i)^c.$$

Das Fragezeichen bedeutet, dass wir an dieser Stelle die Lösung raten. Wir müssen sie natürlich noch formal beweisen, was wir mit vollständiger Induktion machen werden.

I.A.: Für  $n = 1$  stimmt es, weil die leere Summe gleich 0 ist, so wie  $T(1)$ .

I.S.: Also nehmen wir an, dass unsere Formel stimmt für ein  $n \geq 1$  beliebig, aber fest. Dann erhalten wir

$$\begin{aligned} T(n+1) &= T(n) + (n+1)^c \\ &= \sum_{i=0}^{n-2} (n-i)^c + (n+1)^c \\ &= \sum_{i=1}^{n-1} (n-(i-1))^c + (n+1)^c \\ &= \sum_{i=1}^{n-1} ((n+1)-i)^c + (n+1)^c \\ &= \sum_{i=0}^{(n+1)-2} ((n+1)-i)^c. \end{aligned}$$

Somit stimmt die Aussage.

Nun können wir nach oben und unten abschätzen, um die Behauptung der Aufgabe zu erhalten.

$$T(n) = \sum_{i=0}^{n-2} (n-i)^c \leq \sum_{i=0}^{n-2} (n-0)^c = n^c \cdot (n-1) = n^{c+1} - n^c \in \mathcal{O}(n^{c+1})$$

und

$$T(n) = \sum_{i=0}^{n-2} (n-i)^c \geq \sum_{i=0}^{n/2} (n-i)^c \geq \sum_{i=0}^{n/2} (n - \frac{n}{2})^c = \sum_{i=0}^{n/2} (\frac{n}{2})^c \in \Omega(n^{c+1}).$$

Da  $T(n) \in \mathcal{O}(n^{c+1})$  und  $T(n) \in \Omega(n^{c+1})$ , erhalten wir  $T(n) \in \Theta(n^{c+1})$ .

(b) Sei  $n = \left(\frac{4}{3}\right)^k$  mit  $k \in \mathbb{N}$ . Folgende Rekursion ist für die Funktion  $T$  gegeben:

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 2 \cdot T\left(\frac{3}{4}n\right) + 1 \end{aligned}$$

Finden Sie für  $T(n)$  eine geschlossene Form ohne das Mastertheorem zu verwenden und beweisen Sie die Korrektheit Ihrer geschlossenen Form mit vollständiger Induktion.

Wir setzen  $a := \frac{4}{3}$  und ersetzen  $n$  mit  $a^k$ . Dann erhalten wir

$$\begin{aligned} T(a^k) &= 2T(a^{k-1}) + 1 \\ &= 2(2T(a^{k-2}) + 1) + 1 \\ &= 2^2T(a^{k-2}) + 2 + 1 \\ &= 2^2(2T(a^{k-3}) + 1) + 2 + 1 \\ &= 2^3T(a^{k-3}) + 2^2 + 2^1 + 2^0 \\ &\stackrel{?}{=} 2^k + 2^{k-1} + \dots + 2^2 + 2^1 + 2^0 \\ &= \sum_{i=0}^k 2^i \\ &= 2^{k+1} - 1 \end{aligned}$$

Wir müssen das Fragezeichen wieder mit Induktion beweisen. Für  $k = 0$  haben wir

$$T(a^k) = T(1) = 1 = 2^{0+1} - 1 = 2^{k+1} - 1.$$

Schneller Induktionsschritt:

$$T(a^{k+1}) = 2T(a^k) + 1 = 2(2^{k+1} - 1) + 1 = 2^{k+2} - 2 + 1 = 2^{k+2} - 1.$$

### Lösungen zu Aufgabe 3

- Jeder Baum mit  $n$  Knoten hat *immer* genau  $n - 1$  Kanten, insbesondere  $k$ -näre Bäume irgendeiner Höhe  $h$ . Zähle so: Jeder Knoten außer der Wurzel hat genau eine Kante zum Elternknoten (das ergibt eine bijektive Zuordnung zwischen den Kanten und den  $n - 1$  Nicht-Wurzelknoten).
- In Level 0 liegt maximal  $1 = k^0$  Knoten. Da in einem  $k$ -nären Baum jeder Knoten maximal  $k$  Kinder hat, liegen in Level 1 maximal  $1 \cdot k = k^1$ , in Level 2 maximal  $1 \cdot k \cdot k = k^2$  Kinder, usw. In Level  $\ell$  liegen also maximal  $k^\ell$  Knoten.
- Aufsummieren über alle Level liefert  $\sum_{i=0}^{\ell} k^i = \frac{k^{\ell+1}-1}{k-1}$  gemäß der geometrischen Summenformel.
- Falls das letzte Level voll besetzt ist, handelt es sich tatsächlich um einen vollen Baum und der Baum enthält  $\frac{k^{\ell+1}-1}{k-1}$  viele Knoten gemäß Aufgabenteil (c). Da das letzte Level mindestens 1 und höchstens  $k^\ell$  Knoten enthält, können von obigem Wert höchstens  $k^\ell - 1$  Knoten fehlen, d.h., ein solcher Baum hat mindestens  $\frac{k^{\ell+1}-1}{k-1} - (k^\ell - 1) = \frac{k^\ell-1}{k-1} + 1$  viele Knoten.

Dieser Wert ist natürlich der gleiche, den man aus dem Ergebnis von (c) für nur  $\ell - 1$  Level zzgl. 1 weiteren Knoten erhält.

### Lösungen zu Aufgabe 4

Diese drei prominenten Tree-Walks heißen Pre-Order, In-Order und Post-Order, siehe z.B. den Abschnitt "Traversierung" im Wikipedia-Eintrag zu Binärbäumen.

- Alle drei Varianten haben die gleiche Laufzeit, da sie bis auf die Position der PRINT-Operation übereinstimmen.

Unabhängig davon, welche Variante wir betrachten, findet ein initialer Funktionsaufruf für den Wurzelknoten statt - das ist schon einmal ein Aufruf. Weiterhin erzeugt jeder Knoten, mit dem die Funktion aufgerufen wird, zwei weitere rekursive Aufrufe, auch wenn die Kinder nullbind. Zusätzlich ist zu beachten, dass die Funktion mit jedem Knoten als Argument *genau einmal* aufgerufen wird. Damit ergibt sich die Anzahl an Funktionsaufrufen zu  $1 + 2n$  - einen Aufruf mit der Wurzel als Eingabe und 2 rekursive Aufrufe der Funktion für jeden der  $n$  Knoten. Jeder Funktionsaufruf benötigt konstante Zeit  $\Theta(1)$  (eine if-Abfrage und eventuell eine PRINT-Operation sowie Übergabe an die rekursiven Aufrufe). Folglich ergibt sich eine Gesamtlaufzeit von  $\Theta(1 + 2n) = \Theta(n)$  (unabhängig von der konkreten Eingabe).

b) Wie in (a) gezeigt ist die Laufzeit  $\Theta(n)$  unabhängig von der konkreten Eingabe und damit worst-case-Laufzeit und best-case-Laufzeit zugleich.

c) 1: 

S	R	I	T	E	H	G	I	E	M	O	L	W	K	A	L
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

  
 2: 

E	T	I	H	R	I	G	E	S	L	O	W	M	A	K	L
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

  
 3: 

E	T	H	I	I	E	G	R	L	W	O	A	L	K	M	S
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

d) 

S	V	T	N	R	I	Y	U	I	E
---	---	---	---	---	---	---	---	---	---

e) 

W	E	L	I	K	E	A	L	G	O	R	I	T	H	M	S
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

### Lösungen zu Aufgabe 5

Sei  $a_1 \leq a_2 \leq \dots \leq a_n$  das sortierte Array und  $z$  die gesuchte Zahl. Betrachten wir folgendes Verfahren:

(1) Fall  $n = 1$ , teste ob  $a_1 = z$  und gebe das Ergebnis aus.

(2) Falls  $n \geq 2$ , teste, ob  $a_{\lfloor n/2 \rfloor} \geq z$ .

a) Falls ja, verfare rekursiv mit der "linken" Hälfte,  $a_1, \dots, a_{\lfloor n/2 \rfloor}$ , und  $z$  fort.

b) Ansonsten gehen wir in die rechte Hälfte,  $a_{\lfloor n/2 \rfloor + 1}, \dots, a_n$ .

Sei  $T(n)$  die Anzahl der Vergleiche. Es gilt  $T(1) = 1$ . Da die rechte Hälfte größer als die linke sein kann, gilt für  $n \geq 2$

$$T(n) \leq T(n - \lfloor n/2 \rfloor) + 1 = T(\lceil n/2 \rceil) + 1.$$

Zusammengefasst: Ein Vergleich und ein rekursiver Aufruf. Wie schön in anderen Aufgaben, kann die Gleichung aufgelöst werden, und ergibt  $T(n) \leq \lceil \log_2 n \rceil + 1$ . Dies muss mit vollständiger Induktion bewiesen werden:

- I.A.:  $n = 1$ :  $T(1) = 1 = 1 + \log_2 1$ .
- I.V.: Angenommen  $T(n) \leq \lceil \log_2 n \rceil + 1$  für  $n < n_0$  beliebig, aber fest.
- I.S.: Sei  $n_0 \geq 2$ , dann gilt

$$\begin{aligned} T(n_0) &\leq T(\lceil n_0/2 \rceil) + 1 \\ &\stackrel{I.V.}{\leq} \lceil \log_2 \lceil n_0/2 \rceil \rceil + 1 + 1 \\ &\leq \lceil \log_2 n_0 - 1 \rceil + 2 \\ &= \lceil \log_2 n_0 \rceil + 1. \end{aligned}$$

Damit folgt die Behauptung. Dieses Verfahren ist auch bekannt unter binärer Suche.