Datenstrukturen und Algorithmen: Blatt 3

Bernhard Dietrich (6256800)
Lars Fernhomberg (6256030)
Sebastian Kniesburges (6257120)
Marcus Köthenbürger (6258550)

Übungsgruppe 11
Mittwoch, 11:00-13:00 Uhr in D1.303
Matthias Ernst

Aufgabe 1

Ordnen Sie die folgenden vier Funktionen gemäß ihrem asymptotischen Wachstums

$$\left(\frac{3}{2}\right)^n \quad n^3 \quad \left(\log(n)\right)^{\log(n)} \quad 4^{\log(n)}$$

Begründen Sie Ihre Antwort ausführlich.

Die Reihenfolge der Funktionen ist $4^{\log(n)}$, n^3, $\left(\log(n)\right)^{\log(n)}$, $\left(\frac{3}{2}\right)^n$.

$4^{\log(n)} \le n^3$, da $4^{\log(n)} = \left(2 \cdot 2\right)^{\log(n)} = \left(2^2\right)^{\log_2(n)} = 2^{2 \cdot \log(n)} = 2^{\log(n) \cdot 2} = n^2$ und $n^2 \le n^3$ für $n \ge 2$.

$n^3 \le \left(\log(n)\right)^{\log(n)}$, da $n^3 = \left(2^{\log(n)}\right)^3 = 8^{\log(n)}$. Für $n > 2^8$ ist somit die Basis bei der Funktion $\left(\log(n)\right)^{\log(n)}$ größer als bei der Funktion $8^{\log(n)}$, sodass der Funktionswert ebenfalls größer sein muss.

$\left(\log(n)\right)^{\log(n)} < \left(\frac{3}{2}\right)^n$, da bei der Funktion $\left(\frac{3}{2}\right)^n$ der Exponent linear wächst, während bei der Funktion $\left(\log(n)\right)^{\log(n)}$ sowohl die Basis als auch der Exponent nur sehr langsam wachsen.

Aufgabe 2

Die Laufzeit $T(n)$ von Algorithmus A sei gegeben durch die Rekursionsgleichung $T(n) = 7T\left(\frac{n}{2}\right) + n^2$. Die Laufzeit $S(n)$ von Algorithmus B sei gegeben durch die Rekursionsgleichung $S(n) = aS\left(\frac{n}{4}\right) + n^2$. Bestimmen Sie die asymptotische Laufzeit beider Algorithmen. Bestimmen Sie außerdem den größten Wert von a für den B asymptotisch schneller als A ist. Begründen Sie Ihre Antworten.

$$\begin{aligned}
T(n) &= 7 \cdot T\left(\frac{n}{2}\right) + n^2 \\
&= 7 \cdot \left(7 \cdot T\left(\frac{n}{4}\right) + \left(\frac{n}{2}\right)^2\right) + n^2 \\
&= 7 \cdot \left(7 \cdot \left(7 \cdot T\left(\frac{n}{8}\right) + \left(\frac{n}{4}\right)^2\right) + \left(\frac{n}{2}\right)^2\right) + n^2 \\
&= 7^3 \cdot T\left(\frac{n}{8}\right) + 7^2 \cdot \left(\frac{n}{4}\right)^2 + 7\left(\frac{n}{2}\right)^2 + n^2
\end{aligned}$$

$$= 7^l \cdot T(1) + 7^{l-1} \cdot \left(\frac{n}{2^{l-1}}\right)^2 + \ldots + 7^3 \cdot \left(\frac{n}{8}\right)^2 + 7^2 \cdot \left(\frac{n}{4}\right)^2 + 7\left(\frac{n}{2}\right)^2 + n^2$$

$$= 7^l \cdot T(1) + \sum_{i=0}^{l-1} 7^i \cdot \left(\frac{n}{2^i}\right)^2$$

Bei $T(1)$ handelt es sich um den Rekursionsanker, bei dem keine weitere Rekursion mehr aufgerufen wird. Der Rekursionsanker wird in konstanter Zeit ausgeführt, so dass folgt:

$$= 7^l \cdot c + \sum_{i=0}^{l-1} 7^i \cdot \left(\frac{n}{2^i}\right)^2$$

Da $n = 2^l$ folgt $l = \log(n)$:

$$= 7^{\log(n)} \cdot c + n^2 \cdot \sum_{i=0}^{\log(n)-1} \frac{7^i}{2^{i+1}}$$

$$= 2^{\log(7)^{l\,\log(n)}} \cdot c + n^2 \cdot \sum_{i=0}^{\log(n)-1} \left(\frac{7}{4}\right)^i$$

$$= 2^{\log(n)\cdot\log(7)} \cdot c + n^2 \cdot \frac{\left(\frac{7}{4}\right)^{\log(n)} - 1}{\left(\frac{7}{4}\right) - 1}$$

$$= n^{\log(7)} \cdot c + \frac{4 \cdot n^2}{3}\left(\left(\frac{7}{4}\right)^{\log(n)} - 1\right)$$

$$= n^{\log(7)} \cdot c + \frac{4 \cdot n^2}{3} \cdot \frac{7^{\log(n)}}{4^{\log(n)}} - \frac{4 \cdot n^2}{3}$$

$$= n^{\log(7)} \cdot c + \frac{4 \cdot n^2}{3} \cdot \frac{7^{\log(n)}}{n^2} - \frac{4 \cdot n^2}{3}$$

$$= n^{\log(7)} \cdot c + \frac{4}{3} \cdot n^{\log(7)} - \frac{4 \cdot n^2}{3}$$

$$= \left(c + \frac{4}{3}\right) \cdot n^{\log(7)} - \frac{4}{3} n^2$$

$\log(7) \approx 2{,}81 \approx 3$, so dass der Algorithmus eine kubische Laufzeit hat $\left(O\left(n^3\right)\right)$.

$$S(n) = a \cdot S\left(\frac{n}{4}\right) + n^2$$

$$= a \cdot \left(a \cdot S\left(\frac{n}{16}\right) + \left(\frac{n}{4}\right)^2\right) + n^2$$

$$= a \cdot \left(a \cdot \left(a \cdot S\left(\frac{n}{64}\right) + \left(\frac{n}{16}\right)^2\right) + \left(\frac{n}{4}\right)^2\right) + n^2$$

$$= a^3 \cdot S\left(\frac{n}{64}\right) + a^2 \cdot \left(\frac{n}{16}\right)^2 + a\left(\frac{n}{4}\right)^2 + n^2$$

$$= a^l \cdot S(1) + a^{l-1} \cdot S\left(\frac{n}{4^{l-1}}\right)^2 + \ldots + a^3 \cdot \left(\frac{n}{64}\right)^2 + a^2 \cdot \left(\frac{n}{16}\right)^2 + a\left(\frac{n}{4}\right)^2 + n^2$$

Bei $S(1)$ handelt es sich um den Rekursionsanker, bei dem keine weitere Rekursion mehr aufgerufen wird. Der Rekursionsanker wird in konstanter Zeit ausgeführt, so dass folgt:

$$= a^l \cdot c + \sum_{i=0}^{l-1} a^i \cdot \left(\frac{n}{4^i}\right)^2$$

Da $n = 4^l$ folgt $l = \log_4(n)$:

$$= a^{\log_4(n)} \cdot c + \sum_{i=0}^{\log_4(n)-1} a^i \cdot \left(\frac{n}{4^i}\right)^2$$

$$= a^{\log_4(a)^{\log_4 n}} \cdot c + \sum_{i=0}^{\log_4(n)-1} a^i \cdot \frac{n^2}{4^{2i}}$$

$$= n^{\log_4(a)} \cdot c + n^2 \sum_{i=0}^{\log_4(n)-1} \frac{a^i}{4^{2i}}$$

$$= n^{\log_4(a)} \cdot c + n^2 \frac{\left(\frac{a}{16}\right)^{\log_4(n)} - 1}{\frac{a}{16} - 1}$$

$$= n^{\log_4(a)} \cdot c + n^2 \frac{\frac{a^{\log_4(n)}}{16^{\log_4(n)}} - 1}{\frac{a}{16} - 1}$$

$$= n^{\log_4(a)} \cdot c + \frac{\cancel{n^2}\frac{a^{\log_4(n)}}{\cancel{n^2}} - n^2}{\frac{a}{16} - 1}$$

$$= n^{\log_4(a)} \cdot c + \frac{a^{\log_4(n)} - n^2}{\frac{a}{16} - 1}$$

$$= n^{\log_4(a)} \cdot c + \frac{16}{a-16} \cdot \left(n^{\log_4(a)} - n^2\right)$$

$$= n^{\log_4(a)} \cdot c + \frac{16}{a-16} n^{\log_4(a)} - \frac{16}{a-16} n^2$$

$$= n^{\log_4(a)} \cdot \left(c + \frac{16}{a-16}\right) - \frac{16}{a-16} n^2$$

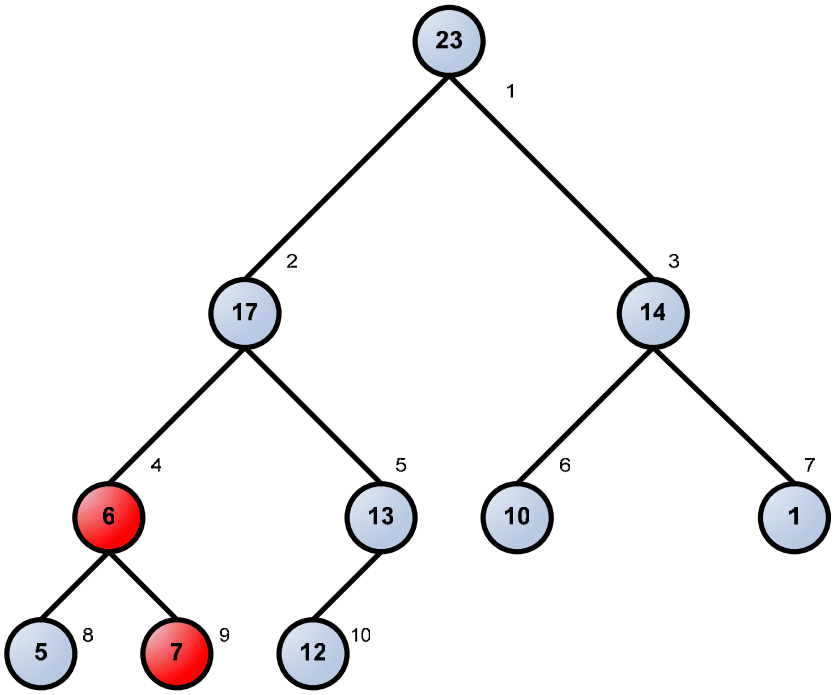
Die Einordnung gemäß der O-Notation hängt beim vorliegenden Algorithmus somit von der Wahl der Variable a ab.

Da $4^3 = 64$ folgt somit, dass der Algorithmus B für $a < 64$ asymptotisch schneller als der Algorithmus A ist.

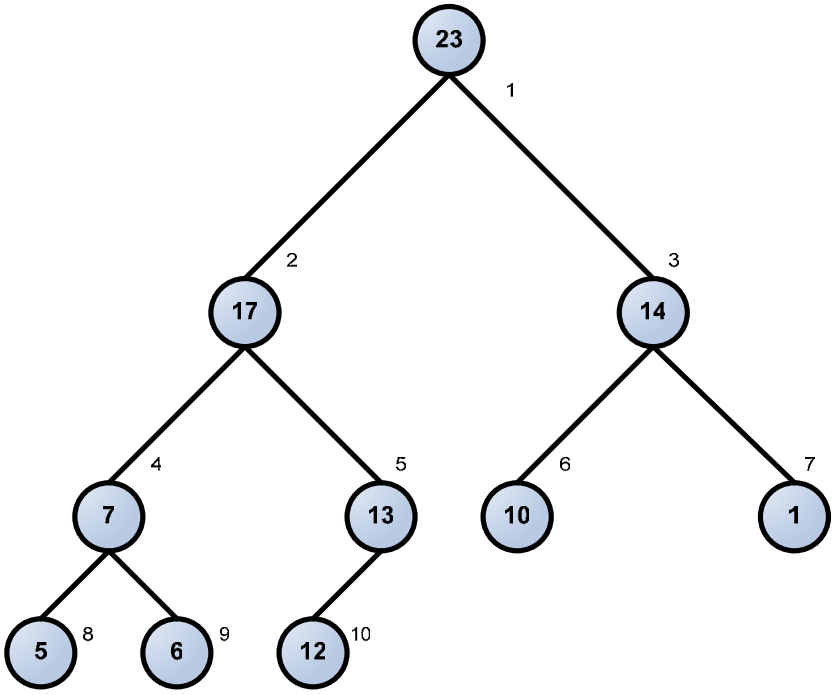
Aufgabe 3

Ist das Array $[23,17,14,6,13,10,1,5,7,12]$ ein max-Heap? Sind Sie der Meinung, dass das Array kein max-Heap ist, ordnen Sie die Elemente so um, dass ein max-Heap entsteht.

Bei der Visualisierung des Arrays in eine Baumstruktur fällt sofort auf, dass die Knoten mit den Indizes 4 bzw. 9 falsch sortiert sind, da der Wert des Kindknoten (Index 9; Wert 7) größer als der Wert des Elternknoten (Index 4; Wert 6) ist, was laut Definition nicht sein darf.



Durch Vertauschen der beiden fehlerhaften Knoten entsteht ein korrekter max-Heap, welcher dem Array $[23,17,14,7,13,10,1,5,6,12]$ entspricht.



Aufgabe 4
Schreiben Sie ausgehend vom Algorithmus MAX-HEAPIFY einen Algorithmus MIN-HEAPIFY, der den entsprechenden Effekt bei min-Heaps hat, d.h. wird MIN-HEAPIFY mit Array A und Index i aufgerufen, so ist nach Beendigung des Algorithmus MIN-HEAPIFY der Teilbaum mit der Wurzel i ein min-Heap.
Analysieren Sie die Laufzeit Ihres Algorithmus.

MIN-HEAPIFY (A, i)

```
 1    l←Left(i)
 2    r←Right(i)
 3    if l ≤ heap-size(A) AND A[l] ≤ A[i]
 4         then smallest ← l
 5         else smallest ← i
 6    if r ≤ heap-size(A) AND A[r] < A[smallest]
 7         then smallest ← r
 8    if smallest ≠ i
 9         then  A[i] ↔ A[smallest]
10              Min-Heapify(A,smallest)
```

Die Zeilen eins bis neun benötigen jeweils eine konstante Zeit c. Ist n die Größe des Baums mit der Wurzel i, so besitzt jeder der beiden Teilbäume mit Wurzel `A[Left(i)]` bzw. `A[Right(i)]` höchstens $\frac{2n}{3}$ Knoten. Dieser Sachverhalt wurde während der Vorlesung während der Besprechung des MAX-HEAPIFY-Algorithmus besprochen und bewiesen (vgl. Foliensatz 06, Folie 15). Somit benötigt der rekursive Aufruf in Zeile 10 höchstens $T\left(\frac{2n}{3}\right)$.

Es ergibt sich somit die Rekursionsgleichung $T(n) = T\left(\frac{2n}{3}\right) + c$ (wobei für $n = 1$ gilt: $T(1) = c$).

$$T(n) = T\left(\frac{2n}{3}\right) + c$$

$$= T\left(\frac{2}{3}\left(\frac{2n}{3}\right)\right) + c + c = T\left(\frac{4n}{9}\right) + c + c$$

$$= T\left(\frac{8n}{27}\right) + c + c + c = \ldots$$

$$\Rightarrow T\left(\left(\frac{2}{3}\right)^k \cdot n\right) + k \cdot c$$

Für welches k gilt $\left(\frac{2}{3}\right)^k \cdot n \leq 1$?

$$k = \log_{\frac{3}{2}}(n) = \log_{\frac{2}{3}}(2) \cdot \log(n)$$

$$T(n) = \left(\log_{\frac{2}{3}}(2) \cdot \log(n) + 1\right) \cdot c = O(\log(n))$$

Somit hat der Algorithmus, genau wie der verwandte MAX-HEAPIFY-Algorithmus, eine logarithmische Laufzeit: $O(\log(n))$.