Datenstrukturen und Algorithmen: Übungsblatt #5

Abgabe am 17. Mai 2018

Finn Hess (378104), Jan Knichel (377779), Paul Orschau (381085)

13.5.2018

Aufgabe 1

Teil a)

$$T(n) = \begin{cases} 1 & 1 \leq n \leq 16 \\ 2 * T(\frac{n}{4}) + T(\frac{n}{2}) + n & n \geq 16 \end{cases} \in \mathcal{O}(n \log n)$$

Beweis

Behauptung:

$$\exists c > 0, n_0 \text{ mit } \forall n \geq n_0 : 0 \leq T(n) \leq c * n \log n \tag{1}$$

Induktionsanfang: $n_0 = 16$

$$\begin{aligned} T(16) = 2 * T(4) + T(8) + 16 = 19 \leq c * 64 = c * 16 \log 16 \\ \Leftrightarrow c \geq \frac{19}{64} \approx 0,297 \end{aligned} \tag{2}$$

Induktionsvoraussetzung:

$$\forall m < n : T(m) \leq c * m \log m \tag{3}$$

Induktionsschritt:

$$\begin{aligned} T(n) &= 2 * T(\frac{n}{4}) + T(\frac{n}{2}) + n \\ &\overset{(IV)}{\leq} 2 * (c * \frac{n}{4} \log \frac{n}{4}) + (c * \frac{n}{2} \log \frac{n}{2}) + n \\ &= (c * \frac{n}{2} \log \frac{n}{4}) + (c * \frac{n}{2} \log \frac{n}{2}) + n \\ &= c * \frac{n}{2} * (\log \frac{n}{4} + \log \frac{n}{2}) + n \\ &= c * \frac{n}{2} * (2 * \log n - 3 * \log 2) + n \\ &= c * n * \log n - c * \frac{3}{2} * n * \log 2 + n \\ &= c * n * \log n - n * \underbrace{(c * \frac{3}{2} * \log 2 - 1)}_{>0} \\ &\leq c * n * \log n \end{aligned} \tag{4}$$

□

Teil b)

$$T(n) = \begin{cases} 1 & 1 \leq n \leq 3 \\ T(n-1) + T(n-2) + T(n-3) & n > 3 \end{cases} \in \Omega(3^{\frac{n}{3}})$$

Beweis

Behauptung:

$$\exists c > 0, n_0 \text{ mit } \forall n \geq n_0 : T(n) \geq c * 3^{\frac{n}{3}} \tag{5}$$

Induktionsanfang: $n_0 = 4$

$$\begin{aligned} T(4) = T(3) + T(2) + T(1) = 1 + 1 + 1 = 3 &\geq c * 3 * 3^{\frac{1}{3}} \\ &\Leftrightarrow 1 \geq c * 3^{\frac{1}{3}} \\ &\Leftrightarrow c \leq \frac{1}{3^{\frac{1}{3}}} \approx 0,693.. \end{aligned} \tag{6}$$

Induktionsvoraussetzung:

$$\forall m < n : T(m) \geq c * 3^{\frac{n}{3}} \tag{7}$$

Induktionsschritt:

$$\begin{aligned} T(n) &= T(n-1) + T(n-2) + T(n-3) \\ &\overset{(IV)}{\geq} c * (3^{\frac{n-1}{3}} + 3^{\frac{n-2}{3}} + 3^{\frac{n-3}{3}}) \\ &= c * (3^{\frac{n}{3}-\frac{1}{3}} + 3^{\frac{n}{3}-\frac{2}{3}} + 3^{\frac{n}{3}-1}) \\ &= c * 3^{\frac{n}{3}} * \underbrace{(3^{-\frac{1}{3}} + 3^{-\frac{2}{3}} + 3^{-1})}_{\approx 1,507..} \\ &\geq c * 3^{\frac{n}{3}} \end{aligned} \tag{8}$$

□

Aufgabe 2

Teil a)

$$\begin{aligned} T(n) &= 8 * T(\frac{n}{2}) + 2^n \\ b &= 8 \\ c &= 2 \\ \rightarrow n^E &= n^3 \\ f(n) &= 2^n \in \Omega(n^{3+\epsilon}) \end{aligned} \tag{9}$$

$f(n)$ wächst exponentiell, und damit auch echt schneller als jedes Polynom.
3. Fall! Prüfe Randbedingung:

$$\begin{aligned} 8 * f(\frac{n}{2}) &\leq d * f(n) \\ \Leftrightarrow 8 * 2^{\frac{n}{2}} &\leq d * 2^n \\ \Leftrightarrow 8 * 2^{-\frac{n}{2}} &\leq d \\ \Leftrightarrow \frac{8}{2^{\frac{n}{2}}} &\leq d \end{aligned} \tag{10}$$

Man sieht leicht, dass für ein festes $d < 1$ die Bedingung für hinreichend große n erfüllt ist.
Damit gilt $T(n) \in \Theta(f(n))$. □

Teil b)

$$\begin{aligned} T(n) &= 64 * T(\frac{n}{4}) + (n^2+1)(n+7) \\ b &= 64 \\ c &= 4 \\ \rightarrow n^E &= n^3 \\ f(n) &= (n^2+1)(n+7) = n^3 + 7n^2 + n + 7 \in \Theta(n^3) \end{aligned} \tag{11}$$

Vergleiche nur die größte Potenz von n.
2. Fall!
Damit gilt $T(n) \in \Theta(n^3 \log n)$. □

Teil c)

$$\begin{aligned} T(n) &= 27 * T(\frac{n}{3}) + n(n^2 \log n + n) \\ b &= 27 \\ c &= 3 \\ \rightarrow n^E &= n^3 \\ f(n) &= n(n^2 \log n + n) = n^3 \log n + n^2 \end{aligned} \tag{12}$$

Da $f(n)$ überproportional schneller als n^E wächst, käme überhaupt nur der 3. Fall in Frage. Prüfe also $f(n) \in \Omega(n^{3+\epsilon})$ für ein beliebiges $\epsilon > 0$ durch Bestimmung des Grenzwerts

$$\begin{aligned}
\lim_{n\to\infty} \frac{f(n)}{n^{E+\epsilon}} &= \lim_{n\to\infty} \frac{n^3 \log n + n^2}{n^{3+\epsilon}} \\
&= \lim_{n\to\infty} \frac{\log n + \frac{1}{n}}{n^{\epsilon}} \\
&= \lim_{n\to\infty} \frac{\log n}{n^{\epsilon}} + \underbrace{\lim_{n\to\infty} \frac{1}{n^{1+\epsilon}}}_{=0} \\
&= \lim_{n\to\infty} \frac{\ln n}{\ln 2 * n^{\epsilon}} \\
&\overset{\text{l'Hopital}}{=} \lim_{n\to\infty} \frac{1}{n * \ln 2 * \epsilon * n^{\epsilon-1}} \\
&= \lim_{n\to\infty} \frac{1}{\ln 2 * \epsilon * n^{\epsilon}} = 0
\end{aligned} \tag{13}$$

Da der Grenzwert gleich 0 ist, gilt $f(n) \in o(n^{E+\epsilon})$ und somit $f(n) \notin \Omega(n^{E+\epsilon})$. Also ist die Rekursionsgleichung für diese Gleichung nicht anwendbar.

Teil d)

$$\begin{aligned}
T(n) &= 16 * T(\frac{n}{4}) + n(\log n + n) \\
b &= 16 \\
c &= 4 \\
\rightarrow n^E &= n^2 \\
f(n) &= n(\log n + n) = n^2 + n \log n
\end{aligned} \tag{14}$$

Da $f(n)$ überproportional schneller als n^E wächst, käme überhaupt nur der 3. Fall in Frage. Prüfe also $f(n) \in \Omega(n^{2+\epsilon})$ für ein beliebiges $\epsilon > 0$ durch Bestimmung des Grenzwerts

$$\begin{aligned}
\lim_{n\to\infty} \frac{f(n)}{n^{E+\epsilon}} &= \lim_{n\to\infty} \frac{n^2 + n \log n}{n^{2+\epsilon}} \\
&= \lim_{n\to\infty} \frac{1 + \frac{\log n}{n}}{n^{\epsilon}} \\
&= \underbrace{\lim_{n\to\infty} \frac{1}{n^{\epsilon}}}_{=0} + \lim_{n\to\infty} \frac{\frac{\log n}{n}}{n^{\epsilon}} \\
&= \lim_{n\to\infty} \frac{\log n}{n^{1+\epsilon}} \\
&\overset{\text{l'Hopital}}{=} \lim_{n\to\infty} \frac{1}{n * (\epsilon+1) * n^{\epsilon}} \\
&\overset{\text{l'Hopital}}{=} \lim_{n\to\infty} \frac{1}{(\epsilon+1) * n^{\epsilon+1}} = 0
\end{aligned} \tag{15}$$

Da der Grenzwert gleich 0 ist, gilt $f(n) \in o(n^{E+\epsilon})$ und somit $f(n) \notin \Omega(n^{E+\epsilon})$. Also ist die Rekursionsgleichung für diese Gleichung nicht anwendbar.

Aufgabe 2 wird auf nächster Seite fortgesetzt...

Teil e)

$$\begin{aligned} T(n) &= T(\frac{n}{3}) + f(n) \\ b &= 1 \\ c &= 3 \\ \rightarrow n^E &= n^0 = 1 \\ f(n) &= \begin{cases} 3n + 2^{3n} & n \in \{2^i | i \in \mathbb{N}\} \\ 3n & \text{sonst} \end{cases} \end{aligned} \tag{16}$$

Es gilt $f(n) \geq g(n) := 3n$, und da selbst $g(n) \in \Omega(\underbrace{n^{E+\frac{1}{2}}}_{\sqrt{n}})$ liegt, muss auch $f(n)$ darin liegen.

3. Fall! Prüfe Randbedingung:

$$\begin{aligned} & 1 * f(\frac{n}{3}) \leq d * f(n) \\ \Leftrightarrow & \begin{cases} n + 2^{3n} & \frac{n}{3} \in \{2^i | i \in \mathbb{N}\} \\ n & \text{sonst} \end{cases} \leq d * \begin{cases} 3n + 2^{3n} & n \in \{2^i | i \in \mathbb{N}\} \\ 3n & \text{sonst} \end{cases} \end{aligned} \tag{17}$$

Man kann nun beliebig große n konstruieren, die diese Ungleichung $\forall d < 1$ nicht erfüllen. Dazu setze man $n := 3 * 2^i$ für ein beliebig großes i.
Für dieses n gilt: $\frac{n}{3} \in \{2^i | i \in \mathbb{N}\}$ sowie $n \notin \{2^i | i \in \mathbb{N}\}$, da Vielfache von 2 nie durch 3 teilbar sind (Primfaktorzerlegung), aber jedes n durch 3 teilbar ist. Somit befinden wir uns für diese n **immer** im "oberen" Fall der linken Seite und im "unteren" Fall der rechten Seite.

$$\begin{aligned} & n + 2^{3n} \leq d * 3n \\ \Leftrightarrow & \frac{n + 2^{3n}}{3n} \leq d \\ & \Rightarrow d \geq \frac{1}{3} + \frac{2^{3n}}{3n} > \frac{2^{3n}}{3n} \overset{*}{\geq} \frac{2^{3n}}{2^{3n}} = 1 \\ & \Rightarrow d > 1 \end{aligned} \tag{18}$$

$$n < 2^n \forall n \in \mathbb{N} \tag{*}$$

Beispiel: Sei $n = 3 * 2^2 = 12$

$$\begin{aligned} & 1 * f(\frac{12}{3}) = f(4) \leq d * f(12) \\ & \Leftrightarrow \underbrace{3 * 4 + 2^{3*4}}_{f(4)} \leq d * \underbrace{3 * 12}_{f(12)} \\ & \Leftrightarrow 4108 \leq 36 * d \\ & \Leftrightarrow d \geq \frac{4108}{36} \approx 114,11.. \end{aligned} \tag{19}$$

Damit ist das Mastertheorem nicht anwendbar.

Aufgabe 3

Teil a)

```
1  T: [3, 1, 2, 4]
2  S: [3, 1, 2, 4] [0, 0, -1]
3  S: [1, 2, 4] [1, 0, 3]
4  S: [2, 4] [2, 0, 3]
5  S: [4] [3, 0, 3]
6  S: [] [4, 3, 4]
7  T: [1, 2, 3]
8  S: [1, 2, 3] [0, 0, -1]
9  S: [2, 3] [1, 0, 1]
10 S: [3] [2, 1, 2]
11 S: [] [3, 2, 3]
```

Die Rückgabe von T([3,1,2,4]) ist [4,3,2,1].

Teil b)
Bedeutung von

- `S(...)`: Sucht den Index des größten Elements der Liste.
- `a`: Wird immer inkrementiert, um zu wissen, an welcher Stelle der Liste man sich befindet.
- `b`: Speichert den Index des (aktuell) größten Elements.
- `c`: Speichert den Wert des (aktuell) größten Elements für zukünftige Vergleiche.
- `i`: Der gesuchte Index des größten Elements, um anschließend nach vorne getauscht zu werden.

Teil c)
Rekursionsgleichung für $T(n)$:

$$\begin{aligned} T(n) &= S(n) + T(n-1) \\ &\overset{(2)}{=} n + 1 + T(n-1) \\ &= n + 1 + (n-1) + 1 + T(n-2) = \dots \\ &= n + 1 + (n-1) + 1 + (n-2) + 1 + \cdots + 3 + 1 + \underbrace{2 + 1 + \underbrace{T(1)}_{=0}}_{T(2)} \\ &= \sum_{k=3}^{n+1} k = \frac{(n+1)(n+2)}{2} - 3 \in \Theta(n^2) \end{aligned} \tag{20}$$

$$S(n) = 1 + S(n-1) = 2 + S(n-2) = \cdots = n + S(0) = n + 1 \tag{21}$$

Aufgabe 3 wird auf nächster Seite fortgesetzt...

Teil d)

L = [3,1,2,4,5,6]

Beweis:

```
T: [3, 1, 2, 4, 5, 6]
S: [3, 1, 2, 4, 5, 6] [0, 0, -1]
S: [1, 2, 4, 5, 6] [1, 0, 3]
S: [2, 4, 5, 6] [2, 0, 3]
S: [4, 5, 6] [3, 0, 3]
S: [5, 6] [4, 3, 4]
S: [6] [5, 4, 5]
S: [] [6, 5, 6]
Swap: i=5
T: [1, 2, 4, 5, 3]
S: [1, 2, 4, 5, 3] [0, 0, -1]
S: [2, 4, 5, 3] [1, 0, 1]
S: [4, 5, 3] [2, 1, 2]
S: [5, 3] [3, 2, 4]
S: [3] [4, 3, 5]
S: [] [5, 3, 5]
Swap: i=3
T: [2, 4, 1, 3]
S: [2, 4, 1, 3] [0, 0, -1]
S: [4, 1, 3] [1, 0, 2]
S: [1, 3] [2, 1, 4]
S: [3] [3, 1, 4]
S: [] [4, 1, 4]
Swap: i=1
T: [2, 1, 3]
S: [2, 1, 3] [0, 0, -1]
S: [1, 3] [1, 0, 2]
S: [3] [2, 0, 2]
S: [] [3, 2, 3]
Swap: i=2
T: [1, 2]
S: [1, 2] [0, 0, -1]
S: [2] [1, 0, 1]
S: [] [2, 1, 2]
Swap: i=1
Ausgabe: [6, 5, 4, 3, 2, 1]
```

Teil e)

Nein, in jedem Schritt wächst der sortierte Bereich um 1. Wenn der sortierte Bereich $n-1$ groß ist, bleibt nur noch ein Element übrig, welches an das Ende gehört. Somit ist die Liste nach spätestens $n-1$ Schritten sortiert, also werden niemals n Swaps ausgeführt.

Aufgabe 4

Teil a)

- **9** / 3 / 5 / 1 / 10 / 4 / 6
- **3 / 9** / 5 / 1 / 10 / 4 / 6
- **3 / 5 / 9** / 1 / 10 / 4 / 6
- **1 / 3 / 5 / 9** / 10 / 4 / 6
- **1 / 3 / 5 / 9 / 10** / 4 / 6
- **1 / 3 / 4 / 5 / 9 / 10** / 6
- **1 / 3 / 4 / 5 / 6 / 9 / 10**

Teil b)
Lösung des Dutch Flag Problem:

```
1  void DutchNationalFlag(Color E[], int n) {
2    int r=0, b=n+1;
3    int u=1;
4    while (u<b) {
5      if (E[u] == rot) {
6        sawp(E[r+1], E[u]);
7        r++;
8        u++;
9      }
10     if (E[u] == weiss) {
11       u++;
12     }
13     if (E[u] == blau) {
14       swap(E[b-1], E[u]);
15       b--;
16     }
17   }
18 }
```

Der Algorithmus kann ein Array in 3 Kategorien sortieren. Die Idee bei 5 verschiedenen Farben besteht darin, zunächst in die Kategorien "Blau", "Rot" und "Alles Andere" zu sortieren, und dann den Algorithmus ein weiteres Mal auf den Bereich "Alles Andere", in dem dann nur noch 3 verschiedene Farben sind, anzuwenden. Letzteres entsprich einfach dem Dutch National Flag Problem.
Dazu muss der Algorithmus aber leicht abgewandelt werden. Er muss:

- Die dritte Farbe darüber definieren, dass es weder die erste noch die zweite Farbe ist
- Mit verschiedenen Farben, die als Eingabeparameter übergeben werden, arbeiten können
- Die Größe des dritten Bereichs zurückliefern

Aufgabe 4 wird auf nächster Seite fortgesetzt...

Hier ist die angepasste Lösung und die beschriebene Verwendung:

```
1  int DutchNationalFlag(Color E[], int n, Color first, Color second) {
2    int r=0, b=n+1;
3    int u=1;
4    while (u<b) {
5      if ((E[u] != first) and (E[u] != second)) { // Element ist "bunt"
6        sawp(E[r+1], E[u]);
7        r++;
8        u++;
9      }
10     if (E[u] == first) {  // Element hat Farbe "first"
11       u++;
12     }
13     if (E[u] == second) { // Element hat Farbe "second"
14       swap(E[b-1], E[u]);
15       b--;
16     }
17   }
18   return r; // Anzahl der Elemente im ersten ("bunten") Bereich
19 }
20
21 int main() {
22   // E ist gegeben, 5 Farben, unsortiert
23
24   int x = DutchNationalFlag(E, E.size, GELB, GRUEN); // Lauzeit in Theta(n)
25
26   // E enthaelt im Bereich 1 bis x nur BLAU, ROT und SCHWARZ unsortiert
27   // und im Bereich x+1 bis n nur GELB (links) und GRUEN (rechts)
28
29   DutchNationalFlag(E, x, ROT, SCHWARZ);  // Laufzeit in Theta(x)
30
31   // E ist sortiert
32   // BLAU, ROT, SCHWARZ, GELB, GRUEN
33   return 0;
34 }
```

Dieser Algorithmus hat immernoch eine lineare Laufzeit in $\Theta(n)$ und einen konstanten Speicherbedarf.