

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:

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

Induktionsanfang: $n_0 = 16$

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

Induktionsvoraussetzung:

$$\forall m < n : T(m) \leq c * m \log m \quad (3)$$

Induktionsschritt:

$$T(n) = 2 * T(\frac{n}{4}) + T(\frac{n}{2}) + n = \quad (4)$$

Teil b)

$$T(n) \in \Omega(3^{\frac{n}{3}})$$

Aufgabe 2

Teil a)

$$\begin{aligned}T(n) &= 8 * T\left(\frac{n}{2}\right) + 2^n \\b &= 8 \\c &= 2 \\&\rightarrow n^E = n^3 \\f(n) &= 2^n \in \Omega(n^{3+\epsilon})\end{aligned}\tag{5}$$

3. Fall! Prüfe Randbedingung:

$$\begin{aligned}8 * f\left(\frac{n}{2}\right) &\leq d * f(n) \\&\Leftrightarrow 8 * 2^{\frac{n}{2}} \leq d * 2^n \\&\Leftrightarrow 8 * 2^{-\frac{n}{2}} \leq d \\&\Leftrightarrow \frac{8}{2^{\frac{n}{2}}} \leq d\end{aligned}\tag{6}$$

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)

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

2. Fall!

Damit gilt $T(n) \in \Theta(n^3 \log n)$.

Teil c)

$$\begin{aligned}T(n) &= 27 * T\left(\frac{n}{3}\right) + n(n^2 \log n + n) \\b &= 27 \\c &= 3 \\&\rightarrow n^E = n^3 \\f(n) &= n(n^2 \log n + n) = n^3 \log n + n^2 \in \Omega(n^{3+\epsilon})\end{aligned}\tag{8}$$

Teil d)

Teil e)

Teil d)

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

Beweis:

```
1 T: [3, 1, 2, 4, 5, 6]
2 S: [3, 1, 2, 4, 5, 6] [0, 0, -1]
3 S: [1, 2, 4, 5, 6] [1, 0, 3]
4 S: [2, 4, 5, 6] [2, 0, 3]
5 S: [4, 5, 6] [3, 0, 3]
6 S: [5, 6] [4, 3, 4]
7 S: [6] [5, 4, 5]
8 S: [] [6, 5, 6]
9 Swap: i=5
10 T: [1, 2, 4, 5, 3]
11 S: [1, 2, 4, 5, 3] [0, 0, -1]
12 S: [2, 4, 5, 3] [1, 0, 1]
13 S: [4, 5, 3] [2, 1, 2]
14 S: [5, 3] [3, 2, 4]
15 S: [3] [4, 3, 5]
16 S: [] [5, 3, 5]
17 Swap: i=3
18 T: [2, 4, 1, 3]
19 S: [2, 4, 1, 3] [0, 0, -1]
20 S: [4, 1, 3] [1, 0, 2]
21 S: [1, 3] [2, 1, 4]
22 S: [3] [3, 1, 4]
23 S: [] [4, 1, 4]
24 Swap: i=1
25 T: [2, 1, 3]
26 S: [2, 1, 3] [0, 0, -1]
27 S: [1, 3] [1, 0, 2]
28 S: [3] [2, 0, 2]
29 S: [] [3, 2, 3]
30 Swap: i=2
31 T: [1, 2]
32 S: [1, 2] [0, 0, -1]
33 S: [2] [1, 0, 1]
34 S: [] [2, 1, 2]
35 Swap: i=1
36 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 entspricht 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

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.