Prof. Dr. Ir. Joost-Pieter Katoen

Sebastian Junges, Benjamin Kaminski, David Korzeniewski, Tim Quatmann

Übung 2

– Musterlösung –

Hinweise:

- Die Lösungen müssen bis **Donnerstag, den 26. April um 16:00 Uhr** in den entsprechenden Übungskasten eingeworfen werden. Sie finden die Kästen am Eingang Halifaxstr. des Informatikzentrums (Ahornstr. 55).
- Die Übungsblätter sollen vorzugsweise in Gruppen von je 3 Studierenden aus der gleichen Kleingruppenübung bearbeitet werden. Ab dem nächsten Übungsblatt **müssen** die Übungsblätter in Dreiergruppen abgegeben werden.
- Drucken Sie ggf. digital angefertigte Lösungen aus. Abgaben z.B. per Email sind nicht zulässig.
- Namen und Matrikelnummer sowie die **Nummer der Übungsgruppe** sind auf jedes Blatt der Abgabe zu schreiben. Abgaben, die aus mehreren Blättern bestehen **müssen geheftet bzw. getackert** werden! Die **Gruppennummer muss sich auf der ersten Seite oben links** befinden.
- **Bei Nichtbeachten der obigen Hinweise müssen Sie mit erheblichen Punktabzügen rechnen!**

Aufgabe 1 (Funktionen sortieren): (6 + 14 = 20 Punkte)

Wir definieren die Quasiordnung $\sqsubseteq$ auf Funktionen als

$$f \sqsubseteq g \qquad \text{genau dann wenn} \qquad f \in \mathcal{O}(g) \ .$$

a) Beweisen Sie, dass $\sqsubseteq$ eine Quasiordnung ist, das heißt dass $\sqsubseteq$ reflexiv und transitiv ist.

b) Sortieren Sie die Funktionen

$$n!, \quad 2^{9000}, \quad 4, \quad 0, \quad \log(n), \quad \sum_{i=0}^{n} \frac{14i^2}{1+i}, \quad \frac{n^3}{2},$$
$$n^n, \quad n^3, \quad n^2, \quad n \cdot \log(n), \quad 2^n, \quad n^2 \cdot \log(n), \quad n \cdot \sqrt{n}$$

in aufsteigender Reihenfolge bezüglich der Quasiordnung $\sqsubseteq$. Schreiben Sie also

$$f \sqsubseteq g \sqsubseteq h \sqsubseteq \ldots \qquad \text{falls} \qquad f \in \mathcal{O}(g) \quad \text{und} \quad g \in \mathcal{O}(h) \quad \text{und} \quad \ldots \ .$$

Lösung:

a) $\sqsubseteq$ ist reflexiv:

$$
\begin{aligned}
& f \sqsubseteq f \\
\iff\ & f \in \mathcal{O}(f) && \text{(per Definition von } \sqsubseteq\text{)} \\
\iff\ & \exists c > 0\ \exists n_0\ \forall n \geq n_0 \colon 0 \leq f(n) \leq c \cdot f(n) && \text{(per Definition von } \mathcal{O}\text{)} \\
\impliedby\ & \exists n_0\ \forall n \geq n_0 \colon 0 \leq f(n) \leq f(n) && \text{(wähle oben } c = 1\text{)} \\
\impliedby\ & \forall n \colon 0 \leq f(n) \leq f(n) && \text{(wähle oben } n_0 = 0\text{)} \\
\iff\ & \forall n \colon 0 \leq f(n) \\
\impliedby\ & f \colon \mathbb{N} \longrightarrow \mathbb{R}_{\geq 0}
\end{aligned}
$$

$\sqsubseteq$ ist transitiv:

$$
\begin{aligned}
& f \sqsubseteq g \quad \text{und} \quad g \sqsubseteq h \\
\iff\ & f \in \mathcal{O}(g) \quad \text{und} \quad g \in \mathcal{O}(h) && \text{(per Definition von } \sqsubseteq\text{)} \\
\iff\ & \exists c > 0\ \exists n_0\ \forall n \geq n_0 \colon 0 \leq f(n) \leq c \cdot g(n) \quad \text{und} \quad \exists c' > 0\ \exists n_0'\ \forall n \geq n_0' \colon 0 \leq g(n) \leq c' \cdot h(n) \\
& && \text{(per Definition von } \mathcal{O}\text{)} \\
\iff\ & \exists c > 0\ \exists n_0\ \forall n \geq n_0 \colon 0 \leq \frac{1}{c} \cdot f(n) \leq g(n) \quad \text{und} \quad \exists c' > 0\ \exists n_0'\ \forall n \geq n_0' \colon 0 \leq g(n) \leq c' \cdot h(n) \\
\implies\ & \exists c, c' > 0\ \exists n_0, n_0'\ \forall n \geq \max\{n_0,\ n_0'\} \colon 0 \leq \frac{1}{c} \cdot f(n) \leq g(n) \leq c' \cdot h(n) \\
\implies\ & \exists c, c' > 0\ \exists n_0, n_0'\ \forall n \geq \max\{n_0,\ n_0'\} \colon 0 \leq \frac{1}{c} \cdot f(n) \leq c' \cdot h(n) \\
\iff\ & \exists c, c' > 0\ \exists n_0, n_0'\ \forall n \geq \max\{n_0,\ n_0'\} \colon 0 \leq f(n) \leq c \cdot c' \cdot h(n) \\
\iff\ & \exists c, c' > 0\ \exists n_0\ \forall n \geq n_0 \colon 0 \leq f(n) \leq c \cdot c' \cdot h(n) \\
\iff\ & \exists c'' > 0\ \exists n_0\ \forall n \geq n_0 \colon 0 \leq f(n) \leq c'' \cdot h(n) \\
\iff\ & f \in \mathcal{O}(h) && \text{(per Definition von } \mathcal{O}\text{)} \\
\iff\ & f \sqsubseteq h && \text{(per Definition von } \sqsubseteq\text{)}
\end{aligned}
$$

b)

$$
0 \ \sqsubseteq\ 2^{9000} \ \sqsubseteq\ 4 \ \sqsubseteq\ \log(n) \ \sqsubseteq\ n \cdot \log(n) \ \sqsubseteq\ n \cdot \sqrt{n} \ \sqsubseteq\ \sum_{i=0}^{n} \frac{14 i^2}{1+i}
$$

$$
\sqsubseteq\ n^2 \ \sqsubseteq\ n^2 \cdot \log(n) \ \sqsubseteq\ \frac{n^3}{2} \ \sqsubseteq\ n^3 \ \sqsubseteq\ 2^n \ \sqsubseteq\ n! \ \sqsubseteq\ n^n
$$

Aufgabe 2 ($\mathcal{O}$-Notation konkret): (5 · 5 = 25 Punkte)

Beweisen oder widerlegen Sie folgenden Aussagen:

a) $\frac{1}{4} n^3 - 7n + 17 \in \mathcal{O}(n^3)$

b) $n^4 \in \mathcal{O}(2n^4 + 3n^2 + 42)$

c) $\log(n) \in \mathcal{O}(n)$

d) $\forall \epsilon > 0 \colon \log(n) \in \mathcal{O}(n^\epsilon)$

e) $a^n \in \Theta(b^n)$, für zwei beliebige Konstanten $a, b > 1$

Lösung:

a) Für alle $n \geq 3$ gilt

$$\frac{1}{4}n^3 - 7n + 17 \leq n^3 \ ,$$

denn

$$\begin{aligned} & \frac{1}{4}n^3 - 7n + 17 \leq n^3 \\ \iff \quad & n^3 - 28n + 68 \leq 4n^3 \\ \iff \quad & 68 - 28n \leq 3n^3 \ , \end{aligned}$$

und die linke Seite der Ungleichung ist monoton fallend in n und für $n \geq 3$ negativ, wohingegen die rechte Seite der Ungleichung monoton wachsend in n und immer positiv ist. Somit existiert ein c (nämlich 1) und ein n_0 (nämlich 3), sodass für alle $n \geq n_0$ gilt:

$$\frac{1}{4}n^3 - 7n + 17 \leq c \cdot n^3$$

Da außerdem $\frac{1}{4}n^3 - 7n + 17$ immer positiv ist, gilt $\frac{1}{4}n^3 - 7n + 17 \in \mathcal{O}(n^3)$.

b) Es gilt für alle $n \in \mathbb{N}$

$$n^4 \leq 2n^4 + 3n^2 + 42$$

und damit auch $n^4 \in \mathcal{O}(2n^4 + 3n^2 + 42)$. Wähle dazu die Konstanten $c = 1$ und $n_0 = 1$.

c) Wir betrachten den folgenden Limes:

$$\lim_{n\to\infty} \frac{\log(n)}{n} \overset{\text{L'Hospital}}{=} \lim_{n\to\infty} \frac{\frac{1}{n}}{1} = \lim_{n\to\infty} \frac{1}{n} = 0$$

Da der Limes 0 ist, haben wir laut (alternativer) Definition von $\mathcal{O}$ gezeigt, dass $\log(n) \in \mathcal{O}(n)$.

d) Wir betrachten den folgenden Limes:

$$\lim_{n\to\infty} \frac{\log(n)}{n^\epsilon} \overset{\text{L'Hospital}}{=} \lim_{n\to\infty} \frac{\frac{1}{n}}{\epsilon \cdot n^{\epsilon-1}} = \lim_{n\to\infty} \frac{1}{\epsilon \cdot n^\epsilon} = 0$$

Da der Limes 0 ist, haben wir laut (alternativer) Definition von $\mathcal{O}$ gezeigt, dass $\log(n) \in \mathcal{O}(n^\epsilon)$.

e) Diese Aussage gilt nicht. Wir wählen beispielsweise $a = 4$ und $b = 2$. Dann ist aber $4^n \notin \mathcal{O}(2^n)$, denn

$$\lim_{n\to\infty} \frac{4^n}{2^n} = \lim_{n\to\infty} \left(\frac{4}{2}\right)^n = \lim_{n\to\infty} 2^n = \infty$$

zeigt, dass der Limes ∞ ist und damit ist laut (alternativer) Definition von $\mathcal{O}$ gezeigt, dass $4^n \notin \mathcal{O}(2^n)$.

Aufgabe 3 ($\mathcal{O}$-Notation abstrakt): (8 + 8 = 16 Punkte)

a) Zeigen oder widerlegen Sie:

$$o(g(n)) \cap \Theta(g(n)) = \emptyset$$

b) Zeigen oder widerlegen Sie:

$$f(n) \in \Omega(g(n)) \quad \text{und} \quad f(n) \in \mathcal{O}(h(n)) \qquad \text{impliziert} \qquad g \in \Theta(h(n))$$

Lösung:

a) Die Aussage gilt. Wir zeigen dies mit einem Widerspruchsbeweis.
Angenommen es gilt $o(g(n)) \cap \Theta(g(n)) \neq \emptyset$. Dann gibt es eine Funktion f mit

$$\begin{aligned} & f \in o(g(n)) \cap \Theta(g(n)) \\ \implies\ & f \in o(g(n)) \text{ und } f \in \Theta(g(n)) \\ \implies\ & f \in o(g(n)) \text{ und } f \in \Omega(g(n)) \end{aligned}$$

Nach der Definition der Klasse Ω folgt, dass es ein $c > 0$ und ein $n_0 \in \mathbb{N}$ gibt mit

$$\forall n \geq n_0 \colon c \cdot g(n) \leq f(n). \tag{1}$$

Außerdem folgt aus der Definition der Klasse o, dass insbesondere für das oben gewählte c ein $n_0' \in \mathbb{N}$ existiert mit

$$\forall n \geq n_0' \colon 0 \leq f(n) < c \cdot g(n). \tag{2}$$

Aus den Aussagen (1) und (2) folgt für $n_0^{\max} := \max(n_0, n_0')$

$$\begin{aligned} & \forall n \geq n_0^{\max} \colon c \cdot g(n) \leq f(n) < c \cdot g(n) \\ \implies\ & c \cdot g(n_0^{\max}) \leq f(n_0^{\max}) < c \cdot g(n_0^{\max}) \\ \implies\ & c \cdot g(n_0^{\max}) < c \cdot g(n_0^{\max}) \\ \implies\ & c < c \end{aligned}$$

Dies führt zum Widerspruch ↯.

b) Die Aussage gilt nicht. Wir zeigen, dass ein Gegenbeispiel existiert. Wähle

$$f(n) = n, \qquad g(n) = 1 \qquad \text{und} \qquad h(n) = n^2.$$

Es gilt $n \in \Omega(1)$, da

$$\liminf_{n\to\infty} \frac{n}{1} = \liminf_{n\to\infty} n = \infty > 0.$$

Außerdem gilt $n \in \mathcal{O}(n^2)$, da

$$\limsup_{n\to\infty} \frac{n}{n^2} = \limsup_{n\to\infty} \frac{1}{n} = 0 \geq 0.$$

Es gilt jedoch nicht $1 \in \Theta(n^2)$, da $1 \notin \Omega(n^2)$:

$$\liminf_{n\to\infty} \frac{1}{n^2} = 0 \not> 0.$$

Aufgabe 4 (Laufzeitanalyse): (5 + 5 + 15 + 10 + 4 = 39 Punkte)

Gegeben sei ein Algorithmus, der für ein Array von Booleans überprüft, ob alle Einträge wahr sind:

```
int allTrue(bool [] E) {
    if (E.length < 1) {
        return -1;
    }
    int m = E.length;
    int i = 0;
    while (i < E.length) {
        if (E[i] == true) {
            m = m - 1;
            if (m == 0) {
                return 1;
            }
        }
        i = i + 1;
    }
    return 0;
}
```

Bei Betrachtung der Laufzeit wird angenommen, dass Vergleiche (z.B. x < y oder b == 0) jeweils eine Zeiteinheit benötigen. Die Laufzeit aller weiteren Operationen wird vernachlässigt.
Sei n die Länge des Arrays E.

a) Bestimmen Sie in Abhängigkeit von n die Best-case Laufzeit $B(n)$.

b) Bestimmen Sie in Abhängigkeit von n die Worst-case Laufzeit $W(n)$.

c) Bestimmen Sie in Abhängigkeit von n die Average-case Laufzeit $A(n)$. Hierzu nehmen wir eine uniforme Verteilung der möglichen Eingaben D_n an, d.h. jede beliebige Eingabe $E \in D_n$ tritt mit Wahrscheinlichkeit $1/|D_n|$ auf.

Hinweise:

- Ihre Lösung darf Summenzeichen $\sum$ enthalten.

d) Geben Sie einen äquivalenten Algorithmus an, dessen Average-case Laufzeit ab einer gewissen Eingabelänge kleiner ist. Hierzu nehmen wir wieder eine uniforme Verteilung der möglichen Eingaben an. Begründen Sie Ihre Antwort kurz. Sie müssen nicht die Average-case Laufzeit ihres Algorithmus berechnen.

Hinweise:

- Wir nennen zwei Algorithmen äquivalent, wenn sie mit der gleichen Eingabe die gleiche Ausgabe produzieren.

e) Gibt es einen äquivalenten Algorithmus, dessen Worst-case Laufzeit in $o(n)$ liegt? Begründen Sie Ihre Antwort.

Lösung:

Zur besseren Erklärung fügen wir Zeilennummern im obigen Programm ein.

```
1  int allTrue(bool [] E) {
2    if (E.length < 1) {
3      return -1;
4    }
5    int m = E.length;
6    int i = 0;
7    while (i < E.length) {
8      if (E[i] == true) {
9        m = m - 1;
10       if (m == 0) {
11         return 1;
12       }
13     }
14     i = i + 1;
15   }
16   return 0;
17 }
```

a) Wir unterscheiden zwei Fälle:

- Falls $n < 1$, so wird nur der Vergleich in Zeile 2 einmal ausgeführt. Daher gilt $B(n) = B(0) = 1$.
- Falls $n \geq 1$, stellen wir fest, dass der return Befehl im Schleifenrumpf (Zeile 11) frühestens nach n Iterationen ausgeführt wird. Daher wird die Schleife immer genau n mal durchlaufen. Im besten Fall sind demnach alle Einträge von E falsch, da so der Vergleich in Zeile 10 nie ausgeführt wird. Dann führen wir 1 mal den Vergleich in Zeile 2, $n+1$ mal den Vergleich in Zeile 7, und n mal den Vergleich in Zeile 8 aus. Es ergibt sich die Best-case Laufzeit

$$B(n) = 1 + (n+1) + n = 2n + 2.$$

b) Wir unterscheiden wieder zwei Fälle:

- Falls $n < 1$, so wird nur der Vergleich in Zeile 2 einmal ausgeführt. Daher gilt $W(n) = W(0) = 1$.
- Falls $n \geq 1$, stellen wir wieder fest, dass die Schleife genau n mal durchlaufen wird. Im schlimmsten Fall sind alle Einträge von E wahr. Dann wird auch der Vergleich in Zeile 10 bei jedem Schleifendurchlauf ausgeführt. Nach genau n Iterationen wird der return Befehl in Zeile 11 ausgeführt. Insgesamt führen wir also den Vergleich in Zeile 2 einmal aus und die Vergleiche in den Zeilen 7, 8 und 10 jeweils n mal. Es ergibt sich die Worst-case Laufzeit

$$W(n) = 1 + n + n + n = 3n + 1.$$

Hinweise:

- Die Worst-case Laufzeit wird auch erreicht, wenn die ersten $n-1$ Einträge wahr sind und der $n$te Eintrag falsch, da dann die Schleifenbedingung insgesamt $n+1$ mal ausgeführt wird.

c) Wir unterscheiden wieder zwei Fälle:

- Falls $n < 1$, so wird nur der Vergleich in Zeile 2 einmal ausgeführt. Daher gilt $A(n) = A(0) = 1$.
- Falls $n \geq 1$, wird die Schleife wieder genau n mal durchlaufen. Aus der Kombinatorik wissen wir, dass es

$$\binom{n}{k} = \frac{n!}{(n-k)! \cdot k!}$$

viele Eingaben mit genau $k \in \{0, \cdots, n\}$ wahren Einträgen gibt[1]. Aufgrund der Gleichverteilung der Eingaben ist die Wahrscheinlichkeit, dass bei einer Eingabe der Länge n genau $k \in \{0, \cdots n\}$ Einträge wahr sind:

$$\frac{\binom{n}{k}}{|D_n|} = \frac{\binom{n}{k}}{2^n}$$

Für den Fall $k = n$ beträgt die Laufzeit (siehe Aufgabenteil b))

$$3n + 1.$$

Falls $k < n$ führen wir 1 mal den Vergleich in Zeile 2, $n + 1$ mal den Vergleich in Zeile 7, n mal den Vergleich in Zeile 8 und k mal den Vergleich in Zeile 10 aus. Insgesamt beträgt die Laufzeit

$$1 + (n + 1) + n + k = 2n + k + 2.$$

Rechnen wir die unterschiedlichen Fälle für k zusammen, ergibt sich

$$A(n) = \underbrace{\frac{1}{2^n} \cdot (3n + 1)}_{n \text{ wahre Einträge}} + \sum_{k=0}^{n-1} \underbrace{\left(\frac{\binom{n}{k}}{2^n} \cdot (2n + k + 2) \right)}_{k \text{ wahre Einträge}}$$

d)

```
int allTrue2(bool [] E) {
    if(E.length < 1) {
        return -1
    }
    int i = 0;
    while(i < E.length) {
        if (E[i] == false) {
            return 0
        }
        i = i + 1;
    }
    return 1
}
```

Der Algorithmus terminiert, sobald ein falscher Eintrag gefunden wurde. Für große n ist die Wahrscheinlichkeit, dass alle Einträge wahr sind sehr klein. Das heißt, dass der alternative Algorithmus mit hoher Wahrscheinlichkeit weniger Schleifendurchläufe benötigt. Somit hat er eine niedrige Laufzeit im Average case.

e) Nein, da sich der Algorithmus im Zweifelsfalle immer alle n Einträge ansehen muss.

[1] "Wähle von den n Einträgen k Einträge aus (ohne Zurücklegen und ohne Beachtung der Reihenfolge), die auf wahr gesetzt werden"