Prof. Dr. Ir. Dr. h. c. Joost-Pieter Katoen Stefan Dollase, Ira Fesefeldt, Tim Quatmann, Jip Spel, Tobias Winkler

Übung 2

Hinweise:

- Die Übungsblätter können in 3er Gruppen bearbeitet werden. Übung 2 soll bis zum 20.04.2020 um 8:00 Uhr online in RWTHmoodle abgegeben werden. Abgaben alleine oder zu zweit sind nicht zulässig. Die Abgabepartner müssen im gleichen Tutorium sein.
- Die Bearbeitung der Übungsblätter ist nicht zwingend notwendig um die Klausurzulassung zu erhalten. Es werden zwar Punkte vergeben, diese dienen jedoch nur zur eignen Einschätzung. Zur Klausurzulassung relevant ist nur das Bestehen der Präsenzübung, welche am 29. Mai stattfinden wird.
- Zur Vorbereitung auf Präsenzübung und Klausur empfehlen wir trotzdem alle Übungsblätter semesterbegleitend zu bearbeiten. Besonders relevant hierfür sind Aufgaben, die mit einem ★ markiert sind.
- Die Lösung des Übungsblattes wird in RWTHmoodle veröffentlicht. Solange die Tutorien nicht stattfinden können, werden außerdem Videos zu jeder Aufgabe angefertigt.
- Wir haben in RWTHmoodle ein Forum eingerichtet, welches als erste Anlaufstelle für **Fragen** dienen soll.
- Relevante Vorlesungen für dieses Übungsblatt: 1, 2

Aufgabe 1 (Ordnung O-Klassen): ★6+12+(4+4+4)=30 Punkte

Wir definieren die Quasiordnung $\sqsubseteq$ auf Funktionen als

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

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

b) Sortieren Sie die Funktionen

$$n!,\quad 10^{400},\quad 4,\quad 0,\quad \log(n),\quad \frac{n^3}{2},\quad n^n,\quad n\cdot\log(n^3),\quad n^3,$$
$$n^2,\quad n\cdot\log(n),\quad 2^n,\quad n^2\cdot\log(n),\quad \log(n!),\quad n\cdot\sqrt{n}$$

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

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

Es ist nicht notwendig die angegebene Sortierung zu begründen.

c) Beweisen oder widerlegen Sie folgende Aussagen:

(i) $\frac{1}{2}n^2 - 5n + 10 \sqsubseteq n^2$

(ii) $n^4 \sqsubseteq 3n^4 + 2n^2 + 42n$

(iii) $\log(n!) \sqsubseteq n\cdot\log(n)$

Aufgabe 2 (Asymptotische Komplexität und Grenzwerte): 6+6+6+10=28 Punkte

a) Geben Sie eine Funktion $f : \mathbb{N} \to \mathbb{R}$ an, sodass $\liminf_{n\to\infty} f(n)$ nicht existiert aber $\limsup_{n\to\infty} f(n)$ existiert. Begründen Sie Ihre Antwort.

b) Zeigen oder widerlegen Sie: Für Funktionen $f, g : \mathbb{N} \to \mathbb{R}_{\geq 0}$ gilt: $g \in \Theta(f) \implies \lim_{n\to\infty} \frac{g(n)}{f(n)} = c$ für ein $0 < c < \infty$.

c) Zeigen Sie: Für alle $c > 0$ gilt $\log(n) \in O(n^c)$. Hinweis: Regel von L'Hospital.

d) Für $n \geq 1$ gilt folgende Ungleichung (diese brauchen Sie nicht zu zeigen):

$$n \log(n) \leq \log(n!) + n$$

wobei log den natürlichen Logarithmus zur Basis e bezeichnet.

Folgern Sie, dass $\log(n!) \in \Omega(n \log(n))$.

Aufgabe 3 (Platzkomplexität): 7+7=14 Punkte

Betrachten Sie folgende Algorithmen. Beide erhalten als Parameter ein Array der Länge n, welches ausschließlich die Ziffern 0 bis 9 enthält. Beide berechnen, auf unterschiedliche Art und Weise, wie oft jede Ziffer im übergebenen Array enthalten ist.

```
public int[] count1(int[] data) {
    int[] counter = new int[10];
    for (int i = 0; i < data.length; i++) {
        counter[data[i]] += 1;
    }
    return counter;
}

public int[] count2(int[] data) {
    int[] counter = new int[10];
    for (int j = 0; j < counter.length; j++) {
        boolean[] matches = new boolean[data.length];
        for (int i = 0; i < data.length; i++) {
            matches[i] = (data[i] == j);
        }
        for (int i = 0; i < matches.length; i++) {
            if (matches[i]) {
                counter[j] += 1;
            }
        }
    }
    return counter;
}
```

a) Geben Sie für beide Algorithmen die Zeitkomplexität in Abhängigkeit von n an. Nehmen Sie dazu an, dass alle elementaren Operationen konstant viel Zeit benötigen.

b) Geben Sie für beide Algorithmen die Platzkomplexität in Abhängigkeit von n an. Nehmen Sie dazu an, dass primitive Datentypen konstant viel Platz benötigen und dass Arrays pro Arrayeintrag konstant viel Platz benötigen. Die Platzkomplexität soll angeben, wie viel Speicher zusätzlich zum übergebenen Array benötigt wird.

Aufgabe 4 (Alternative O-Klassen): 7+7+7+7=28 Punkte

Es seien $f, g : \mathbb{N} \to \mathbb{R}_{\geq 0}$. Aus der Vorlesung ist folgende Definition bekannt:

$$g \in O(f) \iff \exists c > 0, \exists n_0 \in \mathbb{N}, \forall n \geq n_0 : g(n) \leq c \cdot f(n)$$

Für ein beliebiges aber festes $c > 0$ definieren wir die Klasse $\hat{O}_c(f)$ wie folgt:

$$g \in \hat{O}_c(f) \iff \exists n_0 \in \mathbb{N}, \forall n \geq n_0 \colon g(n) \leq c \cdot f(n)$$

Außerdem definieren wir die Klasse $\tilde{O}_k(f)$ für ein beliebiges festes $k \in \mathbb{N}$ als

$$g \in \tilde{O}_k(f) \iff \exists c > 0, \forall n \geq k \colon g(n) \leq c \cdot f(n)$$

a) Zeigen Sie: $\frac{1}{2}n + 1 \in \hat{O}_1(\frac{1}{4}n^2)$

b) Zeigen Sie: $n^2 \in \tilde{O}_0(2^n)$

Zeigen oder widerlegen Sie nun die folgenden zwei Aussagen unter der zusätzlichen Voraussetzung, dass $f(n) > 0$ für alle $n \in \mathbb{N}$ gilt.

c) $g \in O(f) \iff \forall c > 0 \colon g \in \hat{O}_c(f)$

d) $g \in O(f) \iff \forall k \in \mathbb{N} \colon g \in \tilde{O}_k(f)$