Prof. Dr. Ir. Joost-Pieter Katoen

Christian Dehnert, Friedrich Gretz, Benjamin Kaminski, Thomas Ströder

Allgemeine Hinweise:

- Die **Hausaufgaben** sollen in Gruppen von je **2-3 Studierenden** aus der **gleichen Kleingruppenübung (Tutorium)** bearbeitet werden. **Namen und Matrikelnummern** der Studierenden sind auf jedes Blatt der Abgabe zu schreiben. **Heften bzw. tackern Sie die Blätter!**
- Die **Nummer der Übungsgruppe** muss **links oben** auf das **erste Blatt** der Abgabe geschrieben werden. Notieren Sie die Gruppennummer gut sichtbar, damit wir besser sortieren können.
- Die Lösungen müssen **bis Mittwoch, den 29.04.2015 um 12:15 Uhr** in den entsprechenden Übungskasten eingeworfen werden. Sie finden die Kästen am Eingang Halifaxstr. des Informatikzentrums (Ahornstr. 55). Alternativ können Sie die Lösungen auch vor der Abgabefrist direkt bei Ihrer Tutorin/Ihrem Tutor abgeben.
- Aufgaben, die mit einem * markiert sind, sind Sonderaufgaben mit erhöhtem Schwierigkeitsgrad. Sie tragen nicht zur Summe der erreichbaren Punkte bei, die für die Klausurzulassung relevant ist, jedoch werden Ihnen die in solchen Aufgaben erreichten Punkte ganz normal gutgeschrieben.

Tutoraufgabe 1 (Asymptotische Komplexität):

Ordnen Sie die folgenden Funktionen nach ihrer asymptotischer Komplexität in aufsteigender Reihenfolge:

$$n!, \quad n^2, \quad n \log n, \quad n^n, \quad \sqrt{n}, \quad n, \quad n^3, \quad 2^n, \quad 2^{\log n}, \quad \log n, \quad n\sqrt{n} \ .$$

Tutoraufgabe 2 (Ungleichungen):

Zeigen Sie die folgenden Aussagen für beliebige $n \in \mathbb{N}^{>0}$:

a) $\sum_{i=1}^{n}(4 \cdot i + 3) \ \leq \ 4(n^2 + n)$ **b)** $\sum_{i=1}^{n} \log_2 i \ \leq \ \log_2 n^n$ **c)** $\log_2 n \ \leq \ 3 \cdot \log_4 n$

Tutoraufgabe 3 (O-Notation):

Zeigen oder widerlegen Sie die folgenden Aussagen:

a) $g(n) = 2n^3 + 142n^2 + 462 \in \Theta(n^3)$
b) $2^{n+1} \in \Theta(2^n)$
c) $\log n \in \mathcal{O}(\sqrt{n})$
d) $max(f(n),\ g(n)) \in \Theta(f(n) + g(n))$
e) $g(n) + f(n) \in \mathcal{O}(g(f(n)))$

Tutoraufgabe 4 (Programmanalyse):

Gegeben sei der folgende Algorithmus zur Suche von Duplikaten in einem Array:

```
bool duplicate(int E[], int n) {
  for(int i = 0; i < n; i++)
    for(int j = i+1; j < n; j++)
      if(E[i] == E[j])
        return true;

  return false;
}
```

Er erhält ein Array E mit n Einträgen, für das er überprüft, ob zwei Einträge des Arrays denselben Wert besitzen.

Für die Average-Case Analyse gehen wir von folgenden Voraussetzungen aus:

- Mit einer Wahrscheinlichkeit von 0.3 gibt es im Array ein Duplikat.
- Es gibt immer maximal ein Duplikat.
- Wenn ein Wert doppelt enthalten ist, so steht dieser Wert an der ersten Position im Array.
- Das zweite Auftauchen des Wertes ist an jeder (anderen) Position gleich wahrscheinlich.

Bestimmen Sie in Abhängigkeit von n ...

a) die exakte Anzahl der Vergleiche von Einträgen des Arrays im Best-Case.

b) die exakte Anzahl der Vergleiche von Einträgen des Arrays im Worst-Case.

c) die exakte Anzahl der Vergleiche von Einträgen des Arrays im Average-Case.

Aufgabe 5 (Asymptotische Komplexität): (5 · 3 Punkte)

Begründen oder widerlegen Sie die folgenden Aussagen:

a) $n^2 \in O(n^2 - 20n - 250)$.

b) $3n^3 + 12n^2 - 34n + 100 \in \mathcal{O}(n^3)$.

c) Aus $f(n) \in \Omega(g(n))$ und $f(n) \in \mathcal{O}(h(n))$ folgt $g \in \Theta(h(n))$.

d) Aus $f(n) \in \Theta(n)$ folgt $2^{f(n)} \in \mathcal{O}(2^n)$.

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

Aufgabe 6 (Beweis): (3+6 Punkte)

Wir wollen beweisen, dass $n!$ und n^n nicht dieselbe Komplexität haben, also $n! \notin \Theta(n^n)$. Führen sie dazu folgende Schritte aus.

a) Zeigen Sie zunächst, dass $n! \in o(n^n)$.
Hinweis: es gilt allgemein $f(n) \in o(g(n))$, wenn $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.

b) Zeigen Sie nun, dass für eine beliebige Funktion g gilt: $o(g(n)) \cap \Theta(g(n)) = \emptyset$.
Hinweis: es bietet sich ein Widerspruchsbeweis an.
Folgern Sie schließlich die oben gewünschte Aussage.

Aufgabe 7 (Programmanalyse): (2+2+3 Punkte)

Gegeben sei ein Algorithmus zur Suche eines Modus (einer am häufigsten vorkommenden Zahl) in einem Array:

```
int findModus(int [] E) {
  int i = 0;
  int [] counter = {0,0,0,0}; // Zaehlt Vorkommen der Zahlen 0 bis 3
  boolean flag = false;
  int haelfte = (int)Math.ceil(E.length/2.0); // Bei ungerader Laenge
                                              // des Arrays runden wir auf
  while(!flag && i < E.length) {
    int number = E[i];
    counter[number]++;
    if(counter[number] >= haelfte) {
      flag = true;
    }
    i++;
  }

  return getMaxCount(counter); // Extrahiert einen Modus aus dem Zaehler in
                               // konstanter Zeit
}
```

Er erhält ein Array E von Zahlen **zwischen 0 und 3** und findet heraus welche dieser vier Zahlen am häufigsten in E vorkommt. Bei Betrachtung der Laufzeit vernachlässigen wir die Initialisierung der Variablen sowie den Aufruf von `getMaxCount(counter)`, da diese Operationen unabhängig von der Eingabe E immer konstante Laufzeit haben. Wir interessieren uns lediglich für die **Anzahl der Schleifendurchläufe**, von denen jeder einzelne Durchlauf eine Zeiteinheit kostet. Sei n die Länge des Arrays E. Bestimmen Sie in Abhängigkeit von n...

a) die Anzahl der Schleifendurchläufe im Best-case.

b) die Anzahl der Schleifendurchläufe im Worst-case.

c) die Anzahl der Schleifendurchläufe im Average-case. Hierzu nehmen wir an, dass in E eine 1 vorkommt und alle anderen Einträge gleich 3 sind. Die 1 kann jedoch an jeder der n Positionen gleich wahrscheinlich auftreten.