Humboldt-Universität zu Berlin
Institut für Informatik
Prof. Dr. Ulf Leser
M. Bux, B. Grußien, L. Heimberg, P. Schäfer

Berlin, den 24.04.2017

Übungen zur Vorlesung
Algorithmen und Datenstrukturen

Übungsblatt 1

Abgabe: Montag den 8.5.2017 bis 11:10 Uhr vor der Vorlesung im Hörsaal oder bis 10:45 Uhr im Briefkasten neben Raum 3.321, RUD25. Die Übungsblätter sind in Gruppen von 2 Personen zu bearbeiten. Jedes Übungsblatt muss bearbeitet werden. (Sie müssen mindestens ein Blatt für wenigstens eine Aufgabe jedes Übungsblattes abgeben.) Die Lösungen sind auf nach Aufgaben getrennten Blättern abzugeben. Heften Sie bitte die zu einer Aufgabe gehörenden Blätter vor der Abgabe zusammen. Vermerken Sie auf allen Abgaben Ihre Namen, Ihre **CMS-Benutzernamen**, Ihre Abgabegruppe (z.B. AG123) aus Moodle, und Ihren Übungstermin (z.B. Di 13 Uhr bei Marc Bux), zu dem Sie Ihre korrigierten Blätter zurückerhalten werden.
Beachten Sie auch die aktuellen Hinweise auf der Übungswebsite unter:
`https://hu.berlin/algodat17`

Konventionen:

- Für ein Array A ist $|A|$ die Länge von A, also die Anzahl der Elemente in A. Die Indizierung aller Arrays auf diesem Blatt beginnt bei 1 (und endet also bei $|A|$). Bitte beginnen Sie die Indizierung der Arrays in Ihren Lösungen auch bei 1.
- Mit der Aufforderung "Analysieren Sie die Laufzeit" ist hier gemeint, dass Sie eine möglichst gute obere Schranke der Zeitkomplexität angeben sollen und diese begründen sollen.
- Mit log wird der Logarithmus $\log_2$ zur Basis 2 bezeichnet.
- Die Menge der natürlichen Zahlen $\mathbb{N}$ enthält die Zahl 0.

Aufgabe 1 (Funktionen ordnen) — 6 · 2 = 12 Punkte

Beweisen oder widerlegen Sie für die Teilaufgaben (a) bis (f) die folgenden Aussagen:

1. $f \in \mathcal{O}(g)$,
2. $f \in \Omega(g)$.

Hinweis: Benutzen Sie entweder die Definitionen direkt, oder betrachten Sie den Grenzwert des Quotienten der Funktionen und wenden Sie falls nötig den Satz von l'Hôpital an.

	$f(n)$	$g(n)$
(a)	$0.1n$	999^{88^7}
(b)	$3n^2$	$n^2 + 1000$
(c)	2^n	$n2^n$
(d)	2^n	2^{2n}
(e)	n	$\log(n)$
(f)	$\sqrt{n}$	$(\log(n))^2$

Aufgabe 2 (Eigenschaften der $\mathcal{O}$-Notation) **(2+2+2+2)+4 = 12 Punkte**

Beweisen Sie die folgenden Aussagen.

1. Seien f und g Funktionen, die von $\mathbb{N}$ nach $\mathbb{R}^+$ abbilden. Sei $f_0 \in \mathcal{O}(f)$ und $g_0 \in \mathcal{O}(g)$. Dann gilt:
   a) $f_0 \cdot g_0 \in \mathcal{O}(f \cdot g)$.
   b) $f_0 + g_0 \in \mathcal{O}(\max(f, g))$.
   c) $f \notin o(f)$.
   d) $f \in \Theta(f)$.
2. Sei $p\colon \mathbb{N} \to \mathbb{R}$ mit $p(n) = a_d n^d + a_{d-1} n^{d-1} + \cdots + a_1 n + a_0$ ein Polynom in n vom Grad d mit $a_i \in \mathbb{R}$ für $0 \leq i \leq d$ und $a_d > 0$. Dann gilt $|p(n)| \in \mathcal{O}(n^d)$.

Aufgabe 3 (Pseudocodeanalyse) **1+2+3+2+4 = 12 Punkte**

Betrachten Sie den folgenden Algorithmus **N**:

Algorithmus N(A)

Input: Array A der Länge $|A| = n$ von natürlichen Zahlen
Output: ein Boolescher Wert

```
1:  x := 0;
2:  for i := 1 to n − 1 do
3:      x := x + A[i];
4:      y := 0;
5:      for j := i + 1 to n do
6:          y := y + A[j];
7:      end for
8:      if x = y then
9:          return true;
10:     end if
11: end for
12: return false;
```

1. Was gibt der Algorithmus **N** bei Eingabe des Arrays $[2, 0, 1, 1, 2]$ aus?
2. Was berechnet der Algorithmus **N** allgemein bei Eingabe eines Arrays A natürlicher Zahlen?
3. Analysieren Sie die Laufzeit des Algorithmus in Abhängigkeit von n. Erklären Sie Ihre Analyse Schritt für Schritt.
4. Wie lautet die minimale und maximale Anzahl ausgeführter Zuweisungen für die Variable y in Abhängigkeit von n. Betrachten Sie dazu die Zeilen 4 und 6. Begründen Sie Ihre Antwort.
5. Entwerfen Sie einen bezüglich der Laufzeit effizienteren Algorithmus, der bei gleicher Eingabe immer die gleiche Ausgabe wie der Algorithmus **N** erzeugt. Notieren Sie Ihren Algorithmus als Pseudocode. Analysieren Sie die Laufzeit Ihres Algorithmus und erklären Sie Ihre Analyse Schritt für Schritt.

   Hinweis: Volle Punktzahl gibt es nur für einen korrekten Algorithmus mit linearer Laufzeit.

Aufgabe 4 (Algorithmenentwurf) **(4+2+2)+(4+2) = 14 Punkte**

1. Seien A und B Arrays der Länge n, die nur Zahlen aus der Menge $\{1, \ldots, n\}$ enthalten. In den Arrays kann also jede der Zahlen $1, \ldots, n$ beliebig oft vorkommen, und es müssen nicht alle der Zahlen $1, \ldots, n$ vorkommen.

   a) Entwerfen Sie einen möglichst effizienten Algorithmus, der bei Eingabe von A und B ausgibt, ob A und B jede Zahl aus $\{1, \ldots, n\}$ mit der selben Häufigkeit enthalten, d.h. also, ob B eine Permutation von A ist. Die Reihenfolge der Vorkommen der Zahlen ist dabei unwichtig.

   Beispiel: Die Eingabe $A = [1, 4, 3, 4]$ und $B = [4, 1, 4, 3]$ soll zum Ergebnis **true**, die Eingabe $A = [1, 2, 3, 2]$ und $B = [3, 2, 3, 1]$ dagegen zum Ergebnis **false** führen.

   Notieren Sie diesen Algorithmus als Pseudocode.

   Hinweis: Der Algorithmus kann beliebig viel zusätzlichen Speicherplatz verwenden. Volle Punktzahl gibt es jedoch nur für einen korrekten Algorithmus mit linearer Laufzeit.

   b) Analysieren Sie die Laufzeit Ihres in Pseudocode vorgestellten Algorithmus. Erklären Sie Ihre Analyse Schritt für Schritt.

   c) Begründen Sie informell, warum das Problem nicht mit einer sublinearen Anzahl von Elementzugriffen gelöst werden kann (d.h. warum eine Lösung nicht in $o(n)$ liegen kann).

2. Seien A und B Arrays der Länge n, die jeweils jede der Zahlen aus der Menge $\{1, \ldots, n\}$ genau einmal enthalten.

   Für eine Zahl $i \in \{1, \ldots, n\}$ sei i_A die Position, an der die Zahl i im Array A steht. Analog sei i_B defininiert.

   a) Entwerfen Sie einen möglichst effizienten Algorithmus, der bei Eingabe von A und B ein Array C der Länge n ausgibt, für das folgendes gilt:

   - Falls $i_A < i_B$, steht in C an Position i eine -1.
   - Falls $i_A = i_B$, steht in C an Position i eine 0.
   - Falls $i_A > i_B$, steht in C an Position i eine 1.

   Beispiel: Die Eingabe $A = [2, 4, 3, 1]$ und $B = [2, 3, 1, 4]$ soll zum Ergebnis $[1, 0, 1, -1]$ führen.

   Notieren Sie diesen Algorithmus als Pseudocode.

   Hinweis: Volle Punktzahl gibt es jedoch nur für einen korrekten Algorithmus mit linearer Laufzeit.

   b) Analysieren Sie die Laufzeit Ihres vorgestellten Algorithmus. Erklären Sie Ihre Analyse Schritt für Schritt.