Humboldt-Universität zu Berlin
Institut für Informatik
Prof. Dr. Ulf Leser
E. Angriman, F. Nelles, M. Sänger, Dr. P. Schäfer

Berlin, den 10.04.2019

Übungen zur Vorlesung
Algorithmen und Datenstrukturen

Übungsblatt 1

Abgabe: Mittwoch, den 24.04.2019, bis 11:10 Uhr vor der Vorlesung im Hörsaal. Die Übungsblätter sind in Gruppen von 2 Personen zu bearbeiten. 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 den **Übungstermin** (z.B. Mo 13 Uhr bei Mario Sänger), an dem Sie Ihre korrigierten Blätter zurückerhalten möchten.
Beachten Sie die Informationen auf der Übungswebseite (https://hu.berlin/algodat19).

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.
- Die Menge der natürlichen Zahlen $\mathbb{N}$ enthält die Zahl 0.
- Zur Bestimmung der Anzahl der Schritte eines in Pseudocode vorgestellten Algorithmus, reicht es auf diesem Übungsblatt aus, pro Zeile zu zählen, wie oft diese beim Aufruf des Algorithmus mit einer Eingabe der Größe n (maximal) durchgeführt wird. Addieren Sie anschließend alle Werte auf.

Aufgabe 1 (Pseudocodeanalyse I) — 1+2+3+(1+1+2) = 10 Punkte

Betrachten Sie den folgenden Algorithmus **N**, dem als Eingabe ein Array A aus $|A| = n$ natürlichen Zahlen übergeben wird und der als Ausgabe die Zahl x liefert.

Algorithmus N(A)
Input: Array A der Länge $|A| = n$ von natürlichen Zahlen
Output: Zahl $x \in \mathbb{N}$

```
1:  x := 0;
2:  for i := 1 to n do
3:      if A[i] > x then
4:          x := A[i];
5:      end if
6:      if x > n then
7:          x := n;
8:      end if
9:  end for
10: return x;
```

a) Was gibt der Algorithmus **N** bei Eingabe $[1, 7, 3, 8]$ aus?

b) Was berechnet Algorithmus **N** allgemein bei Eingabe eines Arrays A der Größe n?

c) Bestimmen Sie die minimale und maximale Anzahl ausgeführter Zuweisungen für die Variable x in Algorithmus **N** in Abhängigkeit von n. Betrachten Sie dafür die Zeilen 1,4 und 7. Geben Sie je ein Beispiel für eine Best-Case bzw. Worst-Case Array-Instanz an.

d) Wie lautet die Antwort auf a) - c), falls Sie nach Zeile 7 und vor Zeile 8 den Ausdruck "**return** x;" einfügen? Begründen Sie Ihre Antwort. Markieren Sie Ihre Teillösungen zu dieser Aufgabe mit a1) - c1).

Aufgabe 2 (Pseudocodeanalyse II) **1+2+4+4+2 = 13 Punkte**

Betrachten Sie den folgenden Algorithmus **M**:

Algorithmus M(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 **M** bei Eingabe des Arrays $[2, 0, 1, 1, 2]$ aus?

2. Was berechnet der Algorithmus **M** allgemein bei Eingabe eines Arrays A natürlicher Zahlen?

3. 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 und geben Sie je ein Beispiel für eine Best-Case bzw. Worst-Case Array-Instanz an.

4. Entwerfen Sie einen Algorithmus, der bei gleicher Eingabe immer die gleiche Ausgabe wie der Algorithmus **M** erzeugt und nur $c_1 \cdot n + c_2$ Schritte durchführt für zwei Konstanten $c_1, c_2 \in \mathbb{N}$. Notieren Sie Ihren Algorithmus als Pseudocode. Ihr Algorithmus kann beliebig viel zusätzlichen Speicherplatz verwenden.

5. Geben Sie nun die Anzahl der Schritte Ihres Algorithmus an und erklären Sie Ihre Analyse Schritt für Schritt.

Aufgabe 3 (Algorithmenentwurf) **(4+2+2)+(5+2) = 15 Punkte**

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

   a) Entwerfen Sie einen möglichst effizienten Algorithmus in Pseudocode, 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.

   Hinweis: Die Punkte werden gestaffelt nach Effizienz der Lösung vergeben. Für einen korrekten Linearzeit-Algorithmus gibt es 4 Punkte. Ihr Algorithmus kann beliebig viel zusätzlichen Speicherplatz verwenden.

   b) Geben Sie die Anzahl der Schritte Ihres in Pseudocode vorgestellten Algorithmus in Abhängigkeit von n an. Erklären Sie Ihre Analyse Schritt für Schritt.

   c) Begründen Sie informell, warum jeder Algorithmus, der dieses Problem löst, mindestens n Schritte durchführen muss.

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 definiert

   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.

   Notieren Sie diesen Algorithmus als Pseudocode.

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

   Hinweis: Der Algorithmus kann beliebig viel zusätzlichen Speicherplatz verwenden.

   b) Geben Sie die Anzahl der Schritte Ihres vorgestellten Algorithmus in Abhängigkeit von n an. Erklären Sie Ihre Analyse Schritt für Schritt.

Aufgabe 4 (Die böse Stiefmutter) **6 + 6 = 12 Punkte**

1. Um Schneewittchen zu vergiften, hat die böse Stiefmutter einen Apfel so vergiftet, dass man es ihm nicht ansieht. Leider hat ihr Putzmann diesen vergifteten Apfel zurück in die Apfelkiste gelegt. Nun liegen in der Kiste n Äpfel, die allesamt identisch aussehen, von denen aber einer giftig ist. Glücklicherweise weiß die böse Stiefmutter, dass der giftige Apfel wegen des Gifts 101 Gramm wiegt und damit etwas schwerer als die anderen Äpfel ist, die jeweils 100 Gramm wiegen. Außerdem besitzt sie eine Waage mit zwei Waagschalen, mit der sich überprüfen lässt, ob zwei beliebig große Mengen von Äpfeln gleich viel wiegen. Entwerfen Sie einen Algorithmus zur Bestimmung des giftigen Apfels mit maximal $\lceil \log_2(n) \rceil$ Wiegeoperationen. Notieren Sie Ihren Algorithmus in Pseudocode und begründen Sie, warum er mit $\lceil \log_2(n) \rceil$ Wiegeoperationen auskommt. Überlegen Sie, ob und wenn ja wie man mit noch weniger Wiegeoperationen auskommt.

2. Schneewittchen hat dann den Apfel gegessen und ist in einen tiefen Schlaf gefallen. Um auch noch die sieben Zwerge zu vergiften, hat die böse Stiefmutter eine ganze Kiste mit 16 vergifteten Äpfeln gefüllt. Leider sorgt ihr Putzmann schon wieder für Probleme: Er hat die Kiste mit den vergifteten Äpfeln zwischen einigen anderen Apfelkisten in den Keller gestellt. Nun stehen dort insgesamt zehn Kisten, welche jeweils 16 identisch aussehende Äpfel enthalten, wobei genau eine Kiste giftige Äpfel zu je 101 Gramm enthält, während die anderen Kisten normale Äpfel zu je 100 Gramm enthalten. Da die böse Stiefmutter ihre Waage gerade verliehen hat, fragt sie ihren Spiegel um Rat. Dieser ist allerdings leider nicht so allwissend, wie er im Märchen beschrieben wird. Er ist aber immerhin dazu in der Lage, das exakte Gewicht einer ihm präsentierten Menge von Äpfeln zu nennen. Beschreiben Sie, wie die böse Stiefmutter die Kiste mit den giftigen Äpfeln bestimmen kann, ohne den allwissenden Spiegel mehr als einmal zu verwenden (es ist kein Pseudocode nötig).