

# Aufgaben zur Veranstaltung Algorithmen und Datenstrukturen, SS 2023

H. Pflug, J. Dietel

FH Aachen, Campus Jülich; IT Center, RWTH Aachen

## Klausurvorbereitung, Blatt 3

26./27.06.2023

### Aufgabe 1

Gegeben sei das folgende Programmfragment:

```
public static void unknown(int[] numbers) {
    boolean swapped = true;
    for(int i = numbers.length - 1; i > 0 && swapped; i--) {
        swapped = false;
        for (int j = 0; j < i; j++) {
            if (numbers[j] > numbers[j+1]) {
                int temp = numbers[j];
                numbers[j] = numbers[j+1];
                numbers[j+1] = temp;
                swapped = true;
            }
        }
    }
}
```

- a) Wenden Sie den obigen Algorithmus auf das folgende Feld an. Geben Sie den Inhalt des Feldes jeweils nach Durchlaufen der äußeren Schleife an.

	22	108	74	25	42	3	112	15
i=7								
i=6								
i=5								
i=4								
i=3								
i=2								
i=1								

- b) Was leistet der Algorithmus?  
c) Wie ist die Laufzeitkomplexität im Best-Case und im Worst-Case. Geben Sie jeweils eine möglichst kleine obere Schranke mit Hilfe der O-Notation an.

## Aufgabe 2

Schreiben Sie eine Klasse *HashSet* für eine Menge von Strings mit folgenden Eigenschaften:

- *HashSet* implementiert eine Hashtabelle mit Teillisten. Verwenden Sie als Teillisten Objekte der Klasse *ArrayList*.
- Die Elemente sind Strings, in denen ausschließlich Großbuchstaben vorkommen. Sie müssen die Strings nicht auf Korrektheit überprüfen.
- *HashSet* enthält die Methoden  
`public void add(String s)`  
zum Hinzufügen eines Strings in die Hashtabelle und  
`public boolean contains(String s)`  
zur Überprüfung, ob ein String in der Hashtabelle vorhanden ist.
- Die Hashfunktion ist die Position des 1. Buchstabens des Strings im Alphabet. Beispiel: „HAL-LO“ ergibt 8. Hinweis: A hat den ASCII-Wert 65.
- Wird versucht, einen String einzufügen, der schon vorhanden ist, wird nichts getan.

## Aufgabe 3

Ein Binärbaum (kein Suchbaum) für Integer-Zahlen soll implementiert werden.

- a) Schreiben Sie die Klasse *Node* für einen Knoten des Binärbaums.
- b) Für die Klasse *BinaryTree* sollen Sie die nötigen Attribute sowie diesen Konstruktor und diese Methode schreiben:

```
public BinaryTree(int x)
//Erzeugt einen neuen Baum mit dem einzigen Element x

public int insertRandom(int x)
//Fuegt das Element x an einer zufaelligen Stelle in den Baum ein.
//Der Algorithmus startet an der Wurzel und geht mit 50% Wahr-
//scheinlichkei nach links oder rechts. Dies wird so lange fortgesetzt,
//bis eine freie Stelle erreicht wird. Die Methode gibt zurueck, in
//welcher Ebene des Baums das Element eingefuegt wurde.
```

## Aufgabe 4

Fügen Sie der Klasse *BinaryTree* aus Aufgabe 3 eine Methode

```
public int getMaxLevel()
```

hinzu. Die Methode soll zurückgeben, in welchem Level sich der Knoten mit dem größten Wert befindet.

Hinweis: Die Wurzel hat den Level 0. Der Binärbaum ist kein binärer Suchbaum.

## Aufgabe 5

Sortieren Sie das folgende Feld mit Quick-Sort in aufsteigender Reihenfolge. Dabei sollte als Pivot-Element immer das am weitesten rechts liegende Element gewählt werden. Teilfelder von 2 Elementen brauchen Sie nicht mehr mit Quick-Sort zu sortieren - hier dürfen Sie die beiden Elemente gegebenenfalls einfach tauschen. Bitte markieren Sie in jedem Schritt das Pivot-Element, die Vertauschungen und die Grenzen.

14	18	12	15	5	19	6	4	2	10

## Aufgabe 6

Ein Heap (größtes Element in der Wurzel) ist in eine `ArrayList` eingebettet.

Schreiben Sie eine Funktion

```
public static void insertElement(ArrayList<Integer> heap, int newElement)
```

welche das neue Element an der richtigen Stelle im Heap einfügt.

Hinweis: Kommentieren Sie die Einbettung in das Array sowie die Methode zur Bewegung der Elemente im Heap.

## Aufgabe 7

Schreiben Sie eine Methode

```
public static int findElement(int[] array, int val)
```

die im Feld `array` das Element mit dem Wert `val` sucht und seine Position zurückgibt. Falls das Element nicht vorkommt, wird -1 zurückgegeben.

Benutzen Sie dabei die binäre Suche. Gehen Sie davon aus, dass das Feld aufsteigend sortiert ist.

Tipp: Benutzen Sie einen rekursiven Algorithmus und schreiben Sie dazu die Methode

```
private static int findElement(int[] array, int val, int first, int last)
```

die `val` im Teilfeld zwischen `first` und `last` (einschließlich) sucht.

## Aufgabe 8

Ihre Aufgabe ist es, einen Fahrkartenautomaten zu simulieren. Dabei gibt es die folgenden Randbedingungen:

- Abstufung der Fahrpreise in 10 Cent-Beträgen
- Akzeptiert werden Münzen ab 10 Cent und Scheine bis 10 Euro.
- Nur Münzen (ab 10 Cent) werden als Wechselgeld ausgegeben (2€, 1€, 50ct, 20ct und 10ct).
- Der Bestand an Wechselgeld ist endlich und wird mit simuliert.
- Die Ausgabe soll mit möglichst wenigen Geldstücken erfolgen.

- a) Gehen Sie zunächst von einem Automaten aus, der ausreichend Wechselgeld enthält und zeichnen Sie die ersten drei Ebenen des entsprechenden Suchbaums für die Rückgabe von 2,50€. Welcher Zweig des Suchbaums wird mit einem Greedy-Algorithmus durchlaufen?
- b) Betrachten Sie nun einen Automaten der keine 10ct Stücke mehr enthält und 60ct als Wechselgeld zurückgeben soll. Zeichnen Sie den kompletten Suchbaum und benennen Sie, welches Problem mit dem Greedy-Algorithmus auftritt. Beschreiben Sie anschließend kurz, mit welchen Entwurfsprinzipien dieses Problem behoben werden kann.

- c) Was ändert sich bei der Suche, wenn der Automat nun außerdem noch 5ct und 2ct Stücke als Wechselgeld verwenden kann und aktuell folgendes Wechselgeld zur Verfügung hat:

- 1x            50ct
- 3x            20ct
- 50x           2ct

Es soll 0,60€ als Wechselgeld zurückgegeben werden.

Zeichnen Sie den Suchbaum und geben Sie unter den gefundenen Lösungen jeweils die Anzahl der benötigten Münzen an.

- d) Mit welchem Entwurfsprinzip lässt sich die Suche in Aufgabenteil c) verbessern? Beschreiben Sie das Vorgehen und kennzeichnen Sie die Optimierungen im Suchbaum.