Grundlagen Java

Aufgabe 1.1 (5 Punkte)
Füllen Sie in der unten gegebenen Methode ein quadratisches Array mit der übergebenen Kantenlänge zeilenweise mit aufsteigenden natürlichen Zahlen (jede Zeile mit den Zahlen von 1 bis `kantenlaenge - 1`). Benutzen Sie zwei geschachtelte Schleifen.

```
public class ArrayFuellen {

  public static void fuelleArray(int kantenlaenge) {









  }

}
```

Aufgabe 1.2 (6 Punkte)
Betrachten Sie die Java-Klasse `Sichtbar` und geben Sie die Werte der Variablen `a` und `b` zu den Zeitpunkten `1 - 6` an. Falls eine oder mehrere der Variablen zu einem Zeitpunkt nicht definiert sind, so setzen Sie an der entsprechenden Stelle ein Minus (`-`) in die untenstehende Tabelle.

```
public class Sichtbar {
    public static void main(String[] args) {
      int[] a = {1, 2, 3};

      /* Zeitpunkt 1 */

      methode1(a);

      /* Zeitpunkt 4 */

      a[0] = methode2(a);

      /* Zeitpunkt 6 */
    }

    public static void methode1(int[] a) {
      int[] b = new int[2];

      /* Zeitpunkt 2 */

      b[0] = a[0];
      b[1] = ++a[1];

      /* Zeitpunkt 3 */
    }

    public static int methode2(int[] a) {
      int[] b = a;

      /* Zeitpunkt 5 */

      b[1] = 7;
      return 11;
    }
}
```

Zeitpunkt	1	2	3	4	5	6
`int[] a`						
`int[] b`						

Aufgabe 1.3 (5 Punkte)
Die *Stirling-Zahl zweiter Art* $S_{n,k}$ ist eine Zahl aus der Kombinatorik. Sie gibt die Anzahl der Möglichkeiten an, eine n-elementige Menge in k nichtleere disjunkte Teilmengen aufzuteilen.

Berechnen lässt sich die Stirling-Zahl über die rekursive Formel

$$S_{n,k} = S_{n-1,k-1} + n \cdot S_{n-1,k}$$

mit $S_{n,n} = 1$ und $S_{n,0} = S_{0,k} = 0$ für alle $n, k > 0$.

Implementieren Sie diese **rekursive** Formel. Auf Fehlerbehandlung können Sie verzichten.

Laufzeiten, Verifikation, Terminierung

Aufgabe 2.1 (6 Punkte)
Geben Sie zu den folgenden drei Methoden in O-Notation an, in welchen Laufzeitklassen sie in Abhängigkeit des Parameters $n > 0$ liegen. Geben Sie hierbei immer die *kleinste* Laufzeitklasse an.

```
public static int a(int n) {
  int b = 1;
  while ((n /= 2) > 0) {
    b += 1;
  }
  return b;
}
```

Laufzeitklasse (2 Punkte):

```
public static int b(int n) {
  int c = 0;
  for(int i = 0; i < n; i++) {
    c += a(n);
  }
  return c;
}
```

Laufzeitklasse (2 Punkte):

```
public static int c(int n) {
  return a( b(n) );
}
```

Laufzeitklasse (2 Punkte):

Aufgabe 2.2 (4 Punkte)
Ein Algorithmus löse ein Problem der Größe n, indem er es in $n/2$ Schritten auf die Größe $n/2$ aufteilt (jeweils ganzzahlig geteilt). Ein Problem der Größe 1 kann er in einem Schritt lösen. Für die Anzahl der benötigten Schritte ergibt sich also:

$$f : \mathbb{N} \to \mathbb{N}$$
$$f(n) = \begin{cases} 1 & \text{für n} = 1 \\ \frac{n}{2} + f(\frac{n}{2}) & \text{sonst} \end{cases}$$

Zeigen Sie mit Hilfe einer vollständigen Induktion, dass der zugrundeliegende Algorithmus der Laufzeitklasse $O(n)$ zuzuordnen ist.

Suchen und Sortieren

Aufgabe 3.1 (4 Punkte)
Füllen Sie die fehlenden Einträge der Tabelle sinnvoll mit Laufzeiten bzw. Namen der Sortieralgorithmen:

Algorithmus	*Best Case*	*Average Case*	*Worst Case*
	$O(n)$		$O(n^2)$
		$O(n \cdot log(n))$	$O(n^2)$
HeapSort			$O(n \cdot log(n))$

Aufgabe 3.2 (6 Punkte)
Sortieren Sie die Zahlenfolge

`6 3 8 5 9 2`

nach der Methode des in der Vorlesung behandelten **QuickSort**. Markieren Sie jeweils das Pivot-Element. Schreiben Sie die gerade betrachtete Teilfolge in eine neue Zeile, wenn Sie zwei Elemente getauscht haben oder einen neuen rekursiven Aufruf starten. Notieren Sie neben der jeweiligen Zeile, welche Elemente Sie getauscht haben.

6	3	8	5	9	2	getauschte Elemente:

Aufgabe 3.3 (6 Punkte)
Sortieren Sie die Zahlenfolge

```
10 4 7 2 6 9
```

nach der Methode des in der Vorlesung behandelten **HeapSort**. Zeichnen Sie dazu zunächst den initialen binären Baum und diesen danach jeweils nach dem Ende eines *Sift*-Vorganges bzw. dem Ende der `sift`-Methode.

Aufgabe 3.4 (5 Punkte)
Schreiben Sie eine *rekursive* Methode, die die Anzahl der Blätter im übergebenen Baum `baum` zählt und zurückgibt.

```
public static int zaehleBlaetter (Baum baum){




}
```

Objektorientierung

Aufgabe 4.1 (7 Punkte)
Schreiben Sie in dieser Aufgabe zum Thema "Vererbung" zwei kurze Java-Klassen `Person` und `Student`.
Die Klasse `Person` soll über ein privates Datenfeld `name` vom Typ `String` verfügen und einen Konstruktor, mit dem dieses Datenfeld bei Instantiierung einer neuen Person gesetzt werden kann. Außerdem soll die Klasse über eine öffentliche Methode `werBinIch()` verfügen, die den Typ der Klasse, also den Text `Person`, auf die Konsole schreibt.
Die zweite Klasse `Student` soll von der Klasse `Person` erben. Die Klasse soll ebenfalls über einen Konstruktor das "geerbte" Datenfeld `name` setzen können. Außerdem soll die Methode `werBinIch()` *überschrieben* werden, so dass nun der Text `Student` auf die Konsole geschrieben wird.
Hinweis: Sie können annehmen, dass die `AlgoTools` bereits importiert sind.

Aufgabe 4.2 (5 Punkte)
Unter der Annahme, dass Sie die Klassen wie in Aufgabe 4.1 beschrieben implementiert haben - welche dieser Java-Ausdrücke werden vom Compiler zugelassen, beziehungsweise nicht zugelassen und warum?

a) `Object o = new Student("Max");` (1 Punkt)

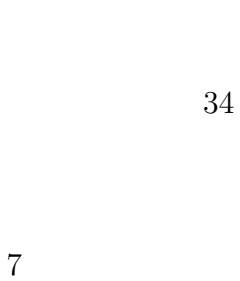
b) `Student s = new Person("Moritz");` (2 Punkte)

c) Was passiert, wenn Sie versuchen folgenden Java-Code auszuführen und warum? (2 Punkte)

```
Person p = new Student("Mueller");
p.werBinIch();
```

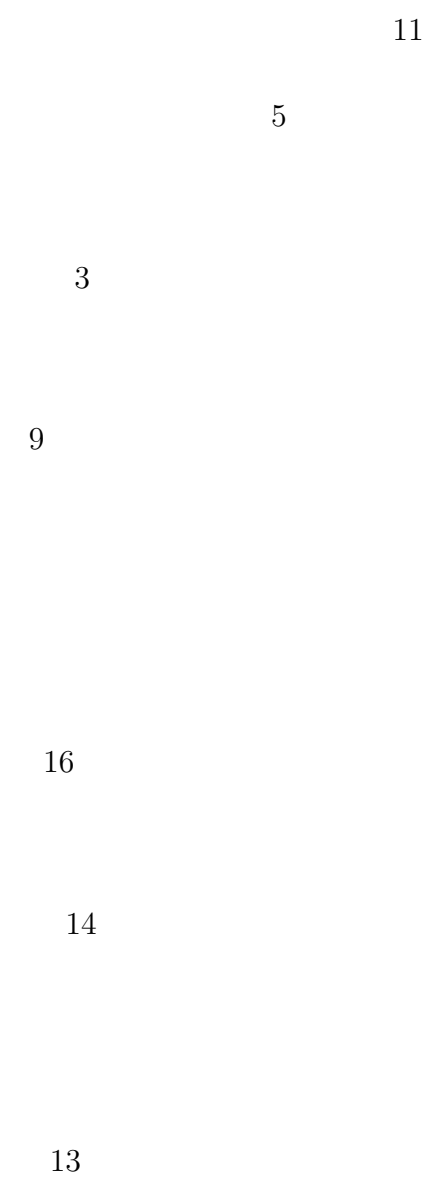
Abstrakte Datentypen

Aufgabe 5.1 (5 Punkte)
Gegeben sei der folgende Baum:



3

Aufgabe 5.2 (6 Punkte)
Gegeben sei folgender Suchbaum:



e in den obigen Baum den Schlüssel `15` ein. (2 Punkte)

c) Löschen Sie aus dem Baum den Schlüssel `11` und zeichnen Sie den resultierenden Baum erneut. (2 Punkte)

Aufgabe 5.3 (8 Punkte)
Für einen String s sei folgende Hashfunktion gegeben: $h(s) = (AnzahlVokale(s) + AnzahlZeichen(s)) \bmod 5$
Beispiele:

$$h(Nicolas) = (3 + 7) \bmod 5 = 10 \bmod 5 = 0$$
$$h(Jana) = (2 + 4) \bmod 5 = 6 \bmod 5 = 1$$

Berechnen Sie die Hashwerte der folgenden Elemente: (2 Punkte)

String	Hashwert
Sebastian	
Julian	
Andrea	
Maike	
Patrick	

Fügen Sie die folgenden Strings in gegebener Reihenfolge nach dem Verfahren des geschlossenen Hashings mit linearem Sondieren in die Hashtabelle ein. (3 Punkte)

`Sebastian, Julian, Andrea, Maike, Patrick`

$h(s)$	Element s
0	
1	
2	
3	
4	

Fügen Sie nun die folgenden Elemente in gegebener Reihenfolge nach dem Verfahren des geschlossenen Hashings mit quadratischem Sondieren in die Hashtabelle ein. (3 Punkte)

`Julian, Andrea, Patrick, Maike`

$h(s)$	Element s
0	
1	
2	
3	
4	

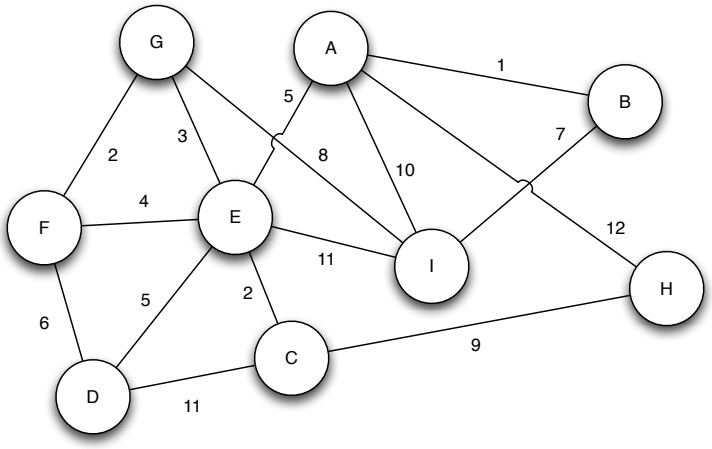
Aufgabe 5.4 (12 Punkte)
Implementieren Sie den abstrakten Datentyp *Liste* mit Hilfe des ADTs *Keller*. Schreiben Sie dazu eine Klasse `KellerListe`, die das Interface *Liste* implementiert.

Eine `KellerListe` speichert alle Listenelemente, die **vor** dem aktuellen Element liegen, in einem Keller `davor` und alle übrigen Listenelemente in einem zweiten Keller `dahinter`. Das aktuelle Element der `KellerListe` ist das oberste Element des Kellers `dahinter`. Die Elemente, die in der `KellerListe` gespeichert werden, sind vom Typ `Object`.

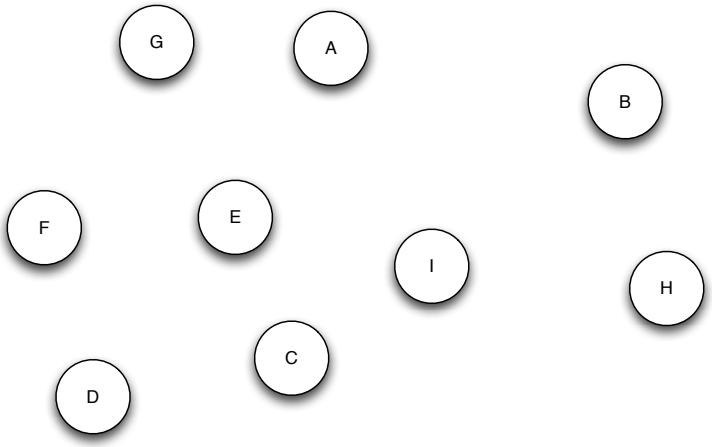
Auf Fehlerbehandlung können Sie verzichten.

Graphenalgorithmen

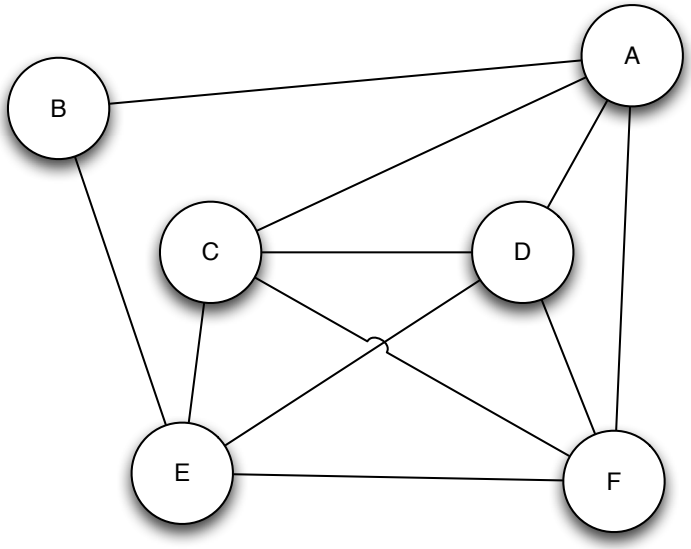
Aufgabe 6.1 (5 Punkte)
Gegeben sei der folgende ungerichtete, gewichtete Graph $G = (V, E)$:



Berechnen Sie mit Hilfe des *Algorithmus von Kruskal* einen minimalen Spannbaum von G und zeichnen Sie die zugehörigen Kanten in die unten dargestellte Vorlage ein.



Aufgabe 6.2 (5 Punkte)
Gegeben sei der folgende ungerichtete Graph $G = (V, E)$:



Identifizieren Sie in diesem Graph eine *Eulertour* beginnend bei Knoten A und geben Sie die von Ihnen gefundene Reihenfolge, in der die Kanten (beziehungsweise Knoten) abgelaufen werden sollen, an.