Grundlagen Java

Aufgabe 1.1 (5 Punkte)
Füllen Sie in der unten gegebenen Methode ein quadratisches Array in der übergebenen Kantenlänge mit aufsteigenden natürlichen Zahlen. Beginnen Sie bei 1 und tragen Sie Zeile für Zeile so viele Zahlen ein, wie in das Array passen. Benutzen Sie genau eine Schleife. Eine Lösung mit zwei geschachtelten Schleifen gibt maximal 3 Punkte. Tipp: Nutzen Sie ganzzahliges Teilen und Modulo-Rechnung.

```
public class ArrayFuellen {

  public static void fuelleArray(int kantenlaenge) {









  }

}
```

Aufgabe 1.2 (5 Punkte)
a) Kodieren Sie die Dezimalzahl -3.25 als *Float* in der binären Exponent-Mantisse-Darstellung, wie in der Vorlesung eingeführt. (3 Punkte)

b) Welchen Wahrheitswert hat folgender Ausdruck? (2 Punkte)

```
(true && false) || ((true == (false || true)) && !(true && false))
```

Aufgabe 1.3 (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 = {0,1};
      /* Zeitpunkt 1 */
      methode1(a);
      /* Zeitpunkt 4 */
      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] + 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`						

Laufzeiten, Verifikation, Terminierung

Aufgabe 2.1 (2 Punkte)
Geben Sie auf die folgenden Theoriefragen eine kurze und prägnante Antwort:

1. Nennen Sie eine aus der Vorlesung bekannte Methode zur Verifikation der partiellen Korrektheit eines Algorithmus'.

2. Zu welchen Zeitpunkten muss bei Ablauf eines Algorithmus' die *Schleifeninvariante* gültig sein?

Aufgabe 2.2 (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;
  int i = 0;
  while (++i < n) {
    b = b + 2 * i + 1;
  }
  return b;
}
```

Laufzeitklasse (2 Punkte):

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

Laufzeitklasse (2 Punkte):

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

Laufzeitklasse (2 Punkte):

Aufgabe 2.3 (4 Punkte)
Geben Sie in O-Notation zu der folgenden rekursiven Methode, unter der Annahme, dass sie initial mit $pos = 0$ aufgerufen wird, in Abhängigkeit der Eingabegröße n an, in welcher Laufzeitklasse sie liegt. Geben Sie hierbei die *kleinste* Laufzeitklasse an. Was ist in diesem Fall die Eingabegröße n?

```
public static int add(int[] ary, int pos) {

  if(pos == ary.length) return 0;

  return ary[pos] + add(ary, pos+1);

}
```

Aufgabe 2.4 (4 Punkte)
Entscheiden Sie, ob der durch die Methode `fraglich(int)` gegebene Algorithmus bei Aufruf mit einer beliebigen ganzen Zahl *terminiert*. Begründen Sie Ihre Entscheidung.

```
public static int fraglich(int n) {

  if( n % 3 == 0 ) return 1;

  else return fraglich(n-3);

}
```

Suchen und Sortieren

Aufgabe 3.1 (2 Punkte)
Geben Sie die asymptotische Laufzeit der folgenden Algorithmen im *Best Case* an:

1. Selectionsort

2. Bubblesort

Aufgabe 3.2 (5 Punkte)
Schreiben Sie eine Methode `count(int pos, int[] numbers, int what)`, welche *rekursiv* die Häufigkeit des Vorkommens der Zahl `what` in einem Array `numbers` berechnet und zurückliefert (wenn die Methode initial mit `pos`= 0 aufgerufen wird). Auf Fehlerbehandlung können Sie verzichten.

```
public static int count(int pos, int[] numbers, int what) {

}
```

Aufgabe 3.3 (6 Punkte)
Sortieren Sie die Zahlenfolge

```
6 3 8 5 9 2
```

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 (2 Punkte)
Wie groß ist die asymptotische Laufzeit des oben betrachteten *HeapSort*-Algorithmus' in den jeweiligen *Cases*?

Best Case	*Average Case*	*Worst Case*

Aufgabe 3.5 (1 Punkt)
Nach welchem Prinzip arbeiten Quicksort und Mergesort?

Objektorientierung

Aufgabe 4.1 (5 Punkte)
Gegeben sei die folgende Java-Klasse sowie das nachfolgende Programm. Geben Sie nachfolgend an, was das Programm an Stelle 1 und Stelle 2 ausgibt. Die `AlgoTools` seien bereits importiert.

```
public class Space {
  public String name;
  public Space left;
  public Space right;

  public Space(String name) {
    this.name = name;
  }

  public Space(String name, Space a, Space b) {
    this.name = name;
    left = a;
    right = b;
  }

  public void method(Space a) {
    a = left;
  }
}


public class TestSpace {
  public static void main(String[] args) {
    Space s1 = new Space("Space 1");
    Space s2 = new Space("Space 2");
    Space s3 = new Space("Space 3", s1, s2);
    Space s4 = new Space("Space 4", s3, s1);
    s1.left = s4.right;
    s3.right.left = s4.left.right;
    s3.left = s4.left.right.left;
    IO.println(s4.left.right.left.name); //Stelle 1

    s4.left.method(s1);
    IO.println(s1.name); //Stelle 2
  }
}
```

Stelle 1 (3 Punkte):

Stelle 2 (2 Punkte):

Aufgabe 4.2 (3 Punkte)
Nehmen Sie an, es gäbe die Klasse `Person` und ihre Unterklassen `Student` und `Professor`. Jede Klasse verfüge über eine Methode `toString()`, die den Typ des jeweiligen Objekts zurück liefert. Geben Sie zu den folgenden Java-Schnipseln an, wie die Ausgabe lautet oder ob die Kompilierung beziehungsweise Ausführung zu einem Fehler führt:

```
Person p = new Professor();
IO.println(p);
```

```
Student p = new Professor();
IO.println(p);
```

```
Professor p = (Professor) (new Person());
IO.println(p);
```

Aufgabe 4.3 (3 Punkte)
Gegeben seien die folgenden beiden Klassen. Geben Sie nachfolgend für die kurzen Ausdrücke an, was diese in einer separaten Testklasse ausgeführt zurück liefern oder ob die Kompilierung beziehungsweise Ausführung zu einem Fehler führt.

```
public class A {
  private int var1;

  public A(int var1) {
    this.var1 = var1;
  }
}
```

```
public class B extends A {
  public var2;

  public B(int var1) {
    super(var1);
    var2 = 42;
  }
}
```

```
A a = new A(23);
IO.println(a.var1);
```

```
B b = new B(5);
IO.println(b.var1);
```

```
A a = new A(23);
IO.println(a.var2);
```

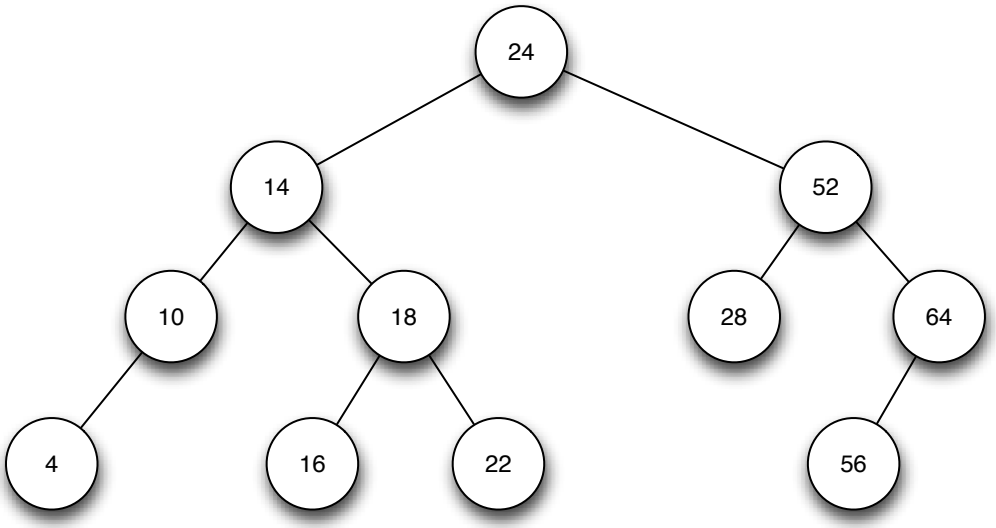
Abstrakte Datentypen

Aufgabe 5.1 (4 Punkte)
Gegeben sei die folgende Postorder-Traversierung eines binären Suchbaums:

```
2 5 8 11 9 7 14 28 32 26 12
```

Zeichnen Sie den zugehörigen binären Suchbaum.

Aufgabe 5.2 (3 Punkte)
Löschen Sie aus dem folgenden Suchbaum mit dem in der Vorlesung vorgestellten Verfahren den Knoten 24. Zeichnen Sie den resultierenden Suchbaum.



Aufgabe 5.3 (6 Punkte)
Erweitern Sie die Klasse `VerweisListe` zu einer `ShiftListe`. Eine `ShiftListe` ergänzt die Funktionalität der `VerweisListe` um eine Methode `public void shift(int n)`, welche die ersten n Elemente in der `ShiftListe` entfernt und unter Berücksichtigung der Reihenfolge an das Ende der Liste anhängt. Auf Fehlerbehandlung können Sie verzichten.

Aufgabe 5.4 (8 Punkte)
Implementieren Sie den abstrakten Datentyp *Keller* und das gleichnamige Interface mit Hilfe eines Arrays als interne Datenstruktur.
Ein `ArrayKeller` speichert alle Elemente in einem `Array`, so dass das unterste Element des Kellers das erste Element im Array ist. Beachten Sie, dass der ADT *Keller* nach dem Prinzip *Last in, first out* arbeitet. Achten Sie auf geeignete Fehlerbehandlung in Situationen, in denen das Array leer oder voll sein könnte. Benutzen Sie *RuntimeException*s.

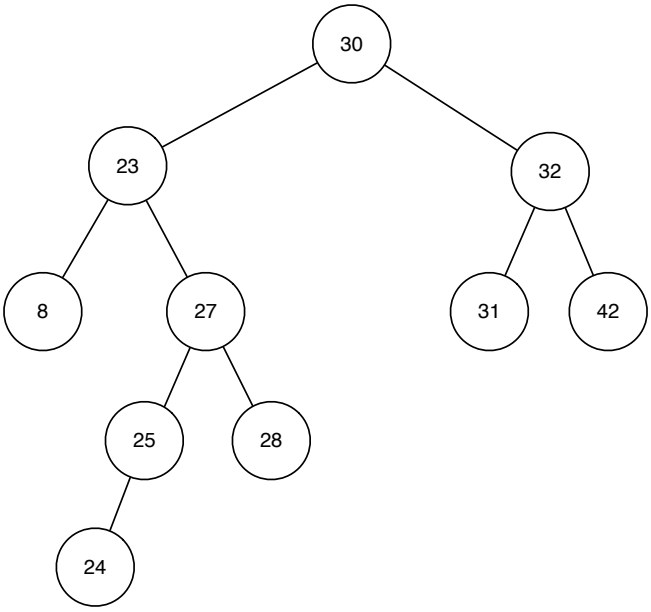
Aufgabe 5.5 (3 Punkte)
Die folgenden Aufgaben beziehen sich auf die Interfaces `Menge, Schlange` sowie die Klasse `SuchBaum` aus der Vorlesung.

a) Wie lautet der Methodenkopf der Methode *insert* aus dem Interface `Menge`?

b) Angenommen es sind n Elemente in einem `SuchBaum`. Wie lange dauert es höchstens, ein Element in einem solchen Baum zu suchen?

c) Welche Methoden sind im Interface `Schlange` deklariert?

Aufgabe 5.6 (3 Punkte)
Gegeben sei folgender, durch das Einfügen des Knoten 24 aus der Balance geratener AVL-Baum:



Geben Sie an *welche* Art von Rotation durchzuführen ist, um den Baum erneut zu balancieren, führen Sie die Rotation durch und zeichnen Sie den resultierenden Baum.

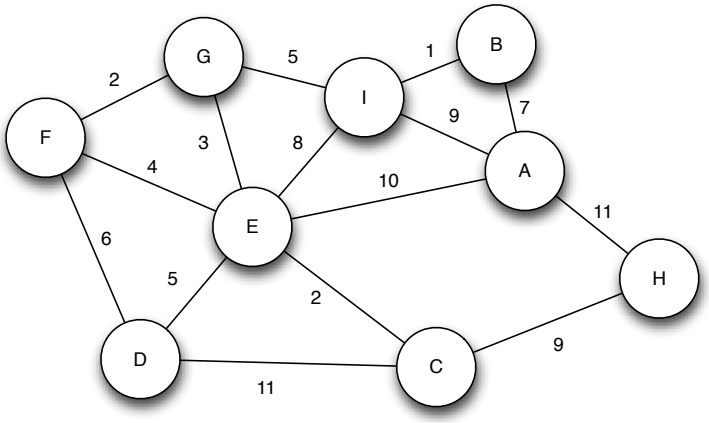
Graphenalgorithmen

Aufgabe 6.1 (6 Punkte)
Schreiben Sie eine *rekursive* Methode `sucheKnoten(Vertex start, String such)`, die mit einem *Backtracking*-Verfahren prüft, ob in einem *kreisfreien Graphen* vom angegeben Knoten `start` aus ein Knoten mit dem Namen `such` erreicht werden kann. In diesem Fall soll die Methode `true` liefern, sonst `false`.

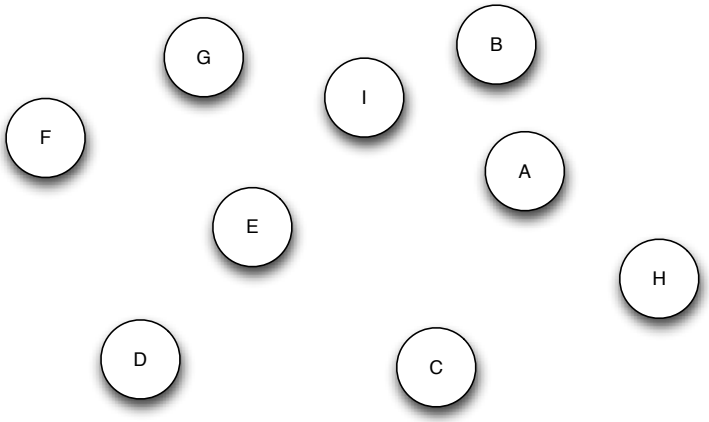
Hinweise: Die Klasse `Vertex` hat die öffentlichen Datenfelder `String name` und `List<Edge> edges`, über die Sie beispielsweise mit einer *for-each*-Schleife iterieren können. Die Klasse `Edge` hat das öffentliche Datenfeld `Vertex dest`.

```
public static boolean sucheKnoten(Vertex start, String such) {




}
```

Aufgabe 6.2 (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 in die unten dargestellte Vorlage die zugehörigen Kanten ein.



Aufgabe 6.3 (3 Punkte)
Geben Sie auf die folgenden Theoriefragen eine kurze und prägnante Antwort:

1. Aus welchen fünf Komponenten besteht ein endlicher Automat?

2. Wie kann man zeigen, dass der Fluss in einem Graphen maximal ist?

3. Wonach sucht man beim Problem des *Chinese Postman*?