Grundlagen Java

Aufgabe 1.1 (5 Punkte)
Füllen Sie in der unten gegebenen Methode das übergebene Array mit aufsteigenden natürlichen Zahlen. Beginnen Sie bei 1 und tragen so viele Zahlen ein, wie in das Array passen.

```
public class ArrayFuellen {

  public static void fuelleArray(int[] zahlen) {









  }

}
```

Aufgabe 1.2 (1 Punkt)
Wie lautet die Signatur der Methode aus Aufgabenteil 1.1?

Aufgabe 1.3 (3 Punkte)
Kodieren Sie die Dezimalzahlen 5 und -7 im *4-Bit Zweierkomplement* und berechnen Sie anschließend $5 - 7$ schriftlich im *4-Bit Zweierkomplement*.

Aufgabe 1.4 (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 int a=5;

    public static void main(String[] args) {

      /* Zeitpunkt 1 */

      int[] b = {1,2,3};

      /* Zeitpunkt 2 */

      a = methode1(b);

      /* Zeitpunkt 6 */

    }

    public static int methode1(int[] b) {

      /* Zeitpunkt 3 */

      b[0] = 0;
      int a = b[a-4];

      /* Zeitpunkt 4 */

      b = new int[2];

      /* Zeitpunkt 5 */

      return 7;
    }
}
```

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

Laufzeiten, Verifikation, Terminierung

Aufgabe 2.1 (3 Punkte)
Geben Sie auf die folgenden Theoriefragen eine kurze und prägnante Antwort:

1. Was ist zu zeigen, um die totale Korrektheit eines Algorithmus' zu beweisen?

2. Ein Algorithmus löse ein Problem in konstanter Laufzeit, wenn das Problem die Größe 1 oder 0 hat. Ansonsten teilt er das Problem in *linearer Laufzeit* in *zwei halb so große* Probleme auf und löst diese rekursiv. In welcher Laufzeitklasse liegt der Algorithmus? (2 Punkte)

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 j = 0;
  for(int i = 0; (i / n) < n; i++)
    j++;
  return j;
}
```

Laufzeitklasse (2 Punkte):

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

Laufzeitklasse (2 Punkte):

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

Laufzeitklasse (2 Punkte):

Aufgabe 2.3 (4 Punkte)
Beweisen Sie, dass der durch die Methode `ggt(int,int)` gegebene Algorithmus (Berechnung des größten gemeinsamen Teilers) bei Aufruf mit zwei ganzen Zahlen *terminiert*. Vervollständigen Sie den unten begonnenen Beweis.

```
public static int ggt(int i, int j) {

  int z = 0;
  i = (i < 0) ? -i : i;   //i = Betrag von i
  j = (j < 0) ? -j : j;   //j = Betrag von j

  while (i > 0) {
    z = j % i;
    j = i;
    i = z;
  }

  return j;
}
```

Zu zeigen ist, dass `i` in `while(i > 0)` einen Wert $<= 0$ erreicht.

Fall 1: $i = j$:
Wenn $i = j$, dann wird z zu $j\%i = 0$, darauf wird i zu z, also zu 0.

Fall 2: $i > j$:
Wenn $i > j$, dann wird z zu $j\%i = j$, darauf wird i zu z, also zu j und j wird zu i. Also tauschen die Variablen ihre Werte und es gilt Fall 3.

Fall 3: $i < j$:
(*Hinweis:* Betrachten Sie die Größe des alten i im Vergleich zu $j\%i$, also dem neuen i.)

Suchen und Sortieren

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

1. Binäre Suche

2. Lineare Suche

Aufgabe 3.2 (8 Punkte)
Schreiben Sie eine Methode `binSearch (int left, int right, int[] numbers, int what)`, die in einem sortierten Array `numbers` mit *binärer Suche* rekursiv in den Index-Grenzen `left`, `right` das Element `what` sucht. Die Methode liefert `true`, wenn das Element enthalten ist und `false` sonst. Auf eine Fehlerbehandlung können Sie verzichten.

```
public static boolean binSearch (int left, int right, int[] numbers, int what)
{

}
```

Aufgabe 3.3 (6 Punkte)
Sortieren Sie die Zahlenfolge

`5 6 9 7 2 1`

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.

5	6	9	7	2	1	getauschte Elemente:

Aufgabe 3.4 (3 Punkte)
Wie groß ist die asymptotische Laufzeit des oben betrachteten *QuickSort*-Algorithmus in den jeweiligen *Cases*?

Best Case	*Average Case*	*Worst Case*

Aufgabe 3.5 (1 Punkt)
Wie groß ist die asymptotische Laufzeit eines auf Vergleichen basierenden Sortieralgorithmus mindestens?

Objektorientierung

Aufgabe 4.1 (8 Punkte)
Gegeben seien die folgenden Java-Klassen (die `AlgoTools` seien bereits importiert). Beantworten Sie hierzu die Fragen auf der folgenden Seite.

```
public class Alpha {

  private int a;

  public Alpha(int a){
    this.a = a;               //Stelle 1
  }

  public void meldung(){
    IO.println("Alpha: " + a);
  }
}
```

```
public class Beta extends Alpha {

  public Beta(int a){
    super(a);                 // Stelle 2
  }

  public void meldung(){
    IO.println("Beta");
  }
}
```

```
public class Test{

  public static void main(String[] argv){
    Alpha a1 = new Alpha(1);
    Alpha a2 = new Beta(2);         // Stelle 3

    a1.meldung();                   // Stelle 4
    a2.meldung();                   // Stelle 5
  }
}
```

Die folgenden Teilaufgaben beziehen sich auf die drei Klassen `Alpha`, `Beta` und `Test`. Beantworten Sie jede Frage mit **maximal drei** prägnanten Sätzen.

a) Erläutern Sie Funktion und Notwendigkeit des Schlüsselwortes `this` an Stelle 1. (2 Punkte)

b) Erläutern Sie die Funktion des Schlüsselwortes `super` an Stelle 2. (2 Punkte)

c) Erläutern Sie, warum die Zuweisung an Stelle 3 vom Compiler akzeptiert wird. (2 Punkte)

d) Welche Konsolenausgabe liefert der Aufruf an Stelle 5. Erläutern Sie, warum sich dieser vom Aufruf an Stelle 4 unterscheidet. (2 Punkte)

Abstrakte Datentypen

Aufgabe 5.1 (6 Punkte)
Erweitern Sie die Klasse `VerweisListe` zu einer `AnzahlListe`. Eine `AnzahlListe` ergänzt die Funktionalität der `VerweisListe` um eine Methode `public int anzahl()`, welche die Anzahl der Elemente in der `AnzahlListe` zurückliefert. Implementieren Sie die Methode *seiteneffektfrei*, das heißt sowohl die Reihenfolge als auch das aktuelle Element müssen erhalten bleiben.

Aufgabe 5.2 (8 Punkte)
Implementieren Sie den abstrakten Datentyp *Keller* mit Hilfe des ADTs *Schlange*. Ergänzen Sie dazu die leeren Methoden in der Klasse `SchlangenKeller`.
Ein `SchlangenKeller` speichert alle Elemente in einer `ArraySchlange`, so dass das erste Element der Schlange das oberste Element des Kellers ist. Beachten Sie, das der ADT *Keller* nach dem Prizip *Last in, first out* arbeitet. Auf Fehlerbehandlung können Sie verzichten.

```
public class SchlangenKeller implements Keller {

  private Schlange schlange;

  private int max;

  public SchlangenKeller(int max) {
    schlange = new ArraySchlange(max);
    this.max = max;
  }


  public boolean empty() {









  }


 public void push(Object element) {









  }
```

```
  public Object top() {









  }

  public void pop() {









  }
}
```

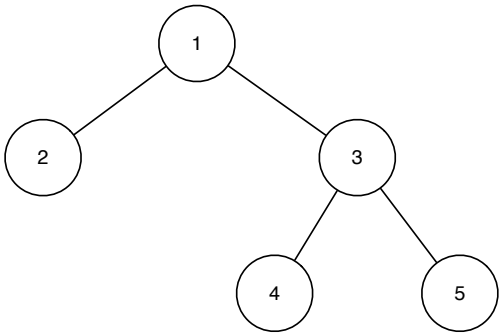
Aufgabe 5.3 (3 Punkte)
Die folgenden Aufgaben beziehen sich auf das Interface `Baum` sowie die Klassen `VerweisBaum` und `AVLBaum` aus der Vorlesung.

a) Wieviele Rotationen sind höchstens nötig, um einen AVLBaum nach Einfügen eines Elementes wieder in Balance zu bringen?

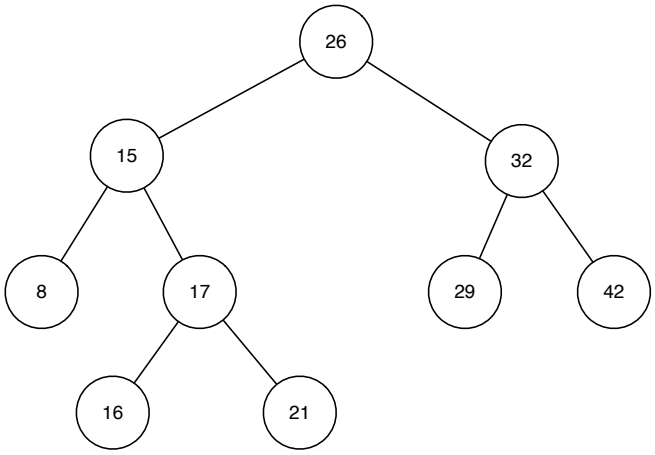
b) In welcher Komplexitätsklasse liegt das Suchen eines Eintrags im AVLBaum?

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

Aufgabe 5.4 (5 Punkte)
Erzeugen Sie per Java einen `VerweisBaum` folgender Gestalt:



Aufgabe 5.5 (7 Punkte)
Gegeben sei folgender AVL-Baum:



a) Fügen Sie in den Baum den Schlüssel 20 ein, indem Sie den Knoten direkt in den obigen AVL-Baum einzeichnen. Versehen Sie den kompletten Baum mit Balancen. (1 Punkt)

Führen Sie eine eventuell notwendige Rotation aus und zeichnen Sie den reorganisierten Baum erneut. Vermerken Sie, welche Rotation Sie verwendet haben. (3 Punkte)

b) Fügen Sie in den neu gezeichneten Baum in Aufgabenteil a) den Schlüssel `7` ein und versehen Sie den kompletten Baum mit Balancen. (1 Punkt)

Führen Sie die notwendige Rotation aus und zeichnen Sie den reorganisierten Baum unten nochmals. Vermerken Sie die Art der Rotation, die Sie verwendet haben. (2 Punkte)

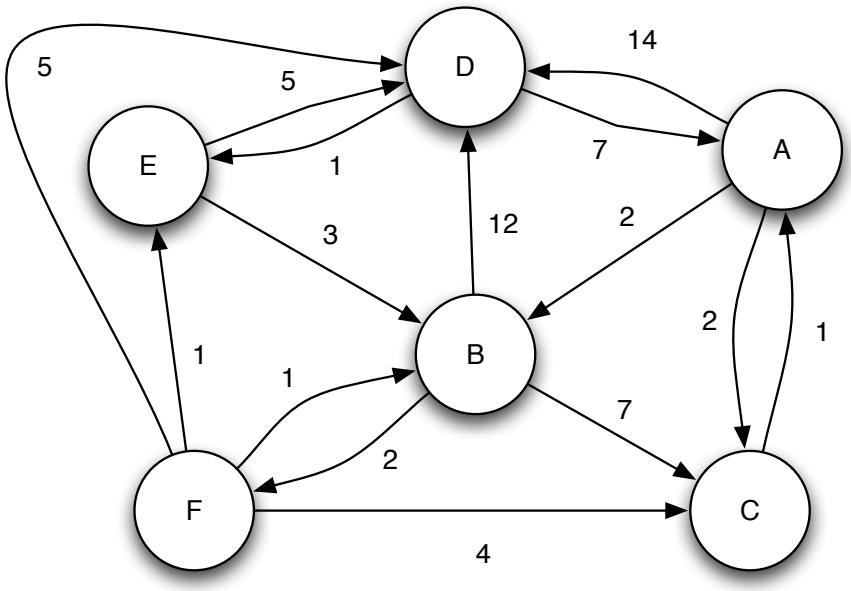
Graphenalgorithmen

Aufgabe 6.1 (2 Punkte)
Geben Sie auf die folgenden Theoriefragen eine kurze und prägnante Antwort:

1. Welche Eigenschaft muss ein Graph haben, damit er topologisch sortierbar ist?

2. Wann ist ein zusammenhängender, ungerichteter Graph *eulersch*?

Aufgabe 6.2 (8 Punkte)
Gegeben sei der folgende gerichtete, gewichtete Graph $G = (V, E)$:

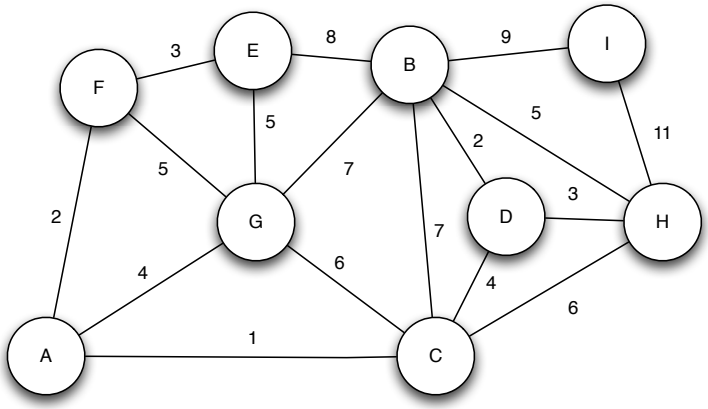


a) Stellen Sie G mit Hilfe einer *Adjazenz-Matrix* dar. Benutzen Sie die folgende Vorlage (2 Punkte):

	A	B	C	D	E	F
A						
B						
C						
D						
E						
F						

b) Berechnen Sie mit dem *Algorithmus von Dijkstra* kürzeste Wege von Knoten A zu allen anderen Knoten in G. Notieren Sie deutlich die errechneten Gesamtkosten für den jeweils günstigsten Weg zu einem Knoten und machen Sie beispielsweise durch Markierung der Kanten klar, welcher Weg zu den gefundenen kürzesten Wegen gehört. (Hinweis: Sollten Sie dabei einen Fehler machen, weisen Sie deutlich darauf hin und wählen für Ihre Korrektur eine Darstellung der Form `H: I -> J -> K -> H`.) (6 Punkte)

Aufgabe 6.3 (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.