Aufgabe 1: Arbeiten mit komplexen Zahlen

Die folgende Klasse `IComplex` stellt komplexe Zahlen mit ganzzahligen Komponenten dar.

```
\public class IComplex implements Comparable<IComplex> {
    private Integer re;
    private Integer im;

    public String toString() {
       return ("(" + re + "," + im + ")");
    }

    public Double betrag() {
       return Math.sqrt(re*re + im*im);
    }

    public int compareTo(IComplex x) {
       -- fehlende Implementierung
    }

    public static void main(String[] args) {
       IComplex[] v = { IComplex(1,2), IComplex(2,1), IComplex(2,3), IComplex(1,1) };
       -- fehlendes Eintragen aller Elemente aus v in ein TreeSet<IComplex>
       -- und anschließende sortierte Ausgabe aller Eingträge.
    }

    public static int countBetraege(TreeSet<IComplex> ts) {
       -- fehlende Implementierung
    }
}
```

a) Implementieren Sie die Vergleichsmethode `int compareTo(IComplex x)`, so dass zwei Objekte der Klasse `IComplex` primär nach ihrem Betrag verglichen werden. Bei Gleichheit der Beträge sollen die `re`-Komponente verglichen werden und sind auch diese gleich, soll nach `im`-Komponenten unterschieden werden.

b) Programmieren Sie die fehlenden Anweisungen des Hauptprogramms, so dass alle Elemente des Arrays `v` in ein `TreeSet<IComplex>` eingetragen und dann sortiert auf `System.out` ausgegeben werden.

c) Programmieren Sie die fehlende Methode `int countBetraege(TreeSet<IComplex> ts)`, die zählt, wieviele verschiedene Beträge die Einträge von `ts` besitzen.

d) Erklären Sie, warum es sinnvoll ist, dass die Methode
`int countBetraege(TreeSet<IComplex> ts)`
mit dem Attribut **static** deklariert ist.

Aufgabe 2: Wortvergleich

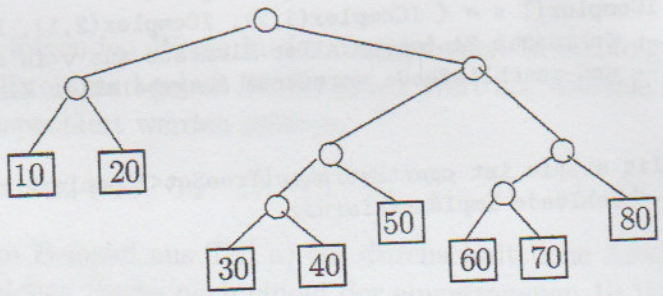
a) Schreiben Sie eine Methode `boolean test(String sa,String sb)`, die für zwei Strings `sa` und `sb` prüft, ob in beiden Strings genau die gleichen Zeichen vorkommen.

Hinweis: Tragen Sie dazu die Zeichen aus `sa` in eine Menge `a` und die Zeichen aus `sb` in eine Menge `b` ein und testen Sie dann mit der Methode `equals` des Interface `Collection` ob beide Mengen gleich sind.

b) Welche größenordnungsmäßige Laufzeit $T_{\text{test}}(n)$ hat Ihre Implementierung aus a), wenn die beiden Strings die gleiche Länge n haben?

c) Benutzt man in a) statt der Methode `equals` den Operator ==, erhält man nicht das gewünschte Ergebnis. Woran liegt das?

Aufgabe 3: Balancierte Bäume



a) Geben Sie für den angegebenen Baum eines Wörterbuchs für alle inneren Knoten die Balancewerte an.

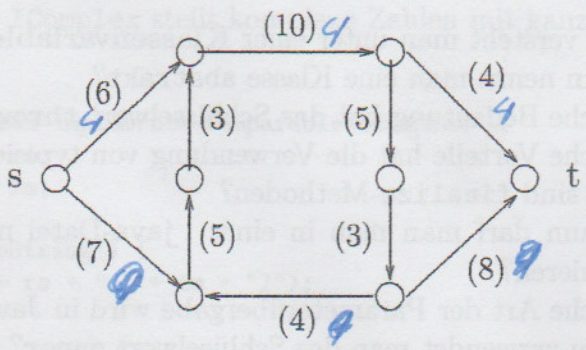
b) Geben Sie den α-Wert für den Baum an.

c) Rebalancieren Sie den Baum am Knoten mit der schlechtesten Balance, um den α-Wert zu verbessern und geben Sie an, welche Rotationsart Sie dabei verwenden.

d) Welche größenordnungsmäßige Laufzeit braucht man, um n Daten in einen anfangs leeren Baum einzutragen?

e) Welche größenordnungsmäßige Laufzeit braucht eine Doppelrotation?

Aufgabe 4: Flussproblem



a) Skizzieren Sie für das angegebene Flussproblem einen maximalen Fluss von s nach t. (Es ist egal, wie Sie diesen Fluss finden)

b) Konstruieren Sie mit dem Fluss aus a) den Hilfsgraphen mit dem mögliche Flussverbesserungen bzw. der minimale Schnitt konstruiert werden kann.

c) Skizzieren Sie den minimalen Schnitt für Flüsse von s nach t.

Aufgabe 5: Obere und untere Schranken

a) Geben Sie für die angegebenen Funktionen f, g und h jeweils möglichst gute größenordnungsmäßige obere Schranken (O-Notation) und untere Schranken (Ω-Notation) an.

$$
\begin{aligned}
f(n) &= 5n^2 + 2\log_2 n + 1 \\
g(n) &= |\sin(3n/5)| \cdot \log_2 n \\
h(n) &= \begin{cases} \log_2(4n+2) & \text{falls } n \leq 3 \\ \sqrt{3n}+1 & \text{falls } n \geq 4 \text{ und } n \text{ ungerade} \\ n/8 & \text{falls } n \geq 4 \text{ und } n \text{ gerade} \end{cases}
\end{aligned}
$$

b) Geben Sie für die größenordnungsmäßige Laufzeit der folgenden beiden Methoden möglichst gute obere und untere Schranken an.

```
boolean test (int n) {
  for (int i=2; i<n; ++i) {
    if (n%i == 0)
      return false;
  }
  return true;
}
```

```
int compute(int n) {
  if (n <= 1)
    return 1;
  else
    if (n%2 == 1)
      return 3*compute(n/2);
    else
      return 2*compute(n/2);
}
```

Aufgabe 6: Allgemeines

a) Was versteht man unter einer **Klassenvariable**?
b) Wann nennt man eine Klasse `abstrakt`?
c) Welche Bedeutung hat das Schlüsselwort `throws` im Kopf einer Methode?
d) Welche Vorteile hat die Verwendung von typisierten Klassen?
e) Was sind `finalize`-Methoden?
f) Warum darf man man in einer `.java`-Datei nicht mehrere öffentliche Klassen definieren?
g) Welche Art der Parameterübergabe wird in Java für Objekte verwendet?
h) Wozu verwendet man das Schlüselwort `super`?

Aufgabe 7: Hashing mit offener Adressierung

a) Zur Speicherung von Zahlen in einer Hashtabelle der Größe 11 soll die Hashfunktion $h_0(x)$ benutzt werden. Im Fall von Kollisionen sollen nacheinander $h_1, h_2, \ldots$ versucht werden.

$$h_i(x) = (x \bmod 11 + i \cdot (1 + (x \bmod 6))) \bmod 11$$

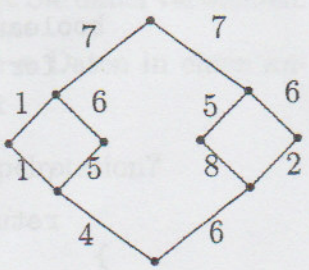
Tragen Sie die folgenden Werte in der angegebenen Reihenfolge in eine anfangs leere Hashtabelle ein und geben Sie für jeden Wert an, wieviele Hashadressen für den Eintrag ausprobiert werden müssen.

12 13 14 17 18 20 22 41 44 70

b) Wie hoch ist im Beispiel aus Teil a) die durchschnittliche Anzahl von Zugriffen bei der erfolgreichen Suche nach einem der eingetragenen 10 Werten?
c) Wie hoch ist im Beispiel aus Teil a) die Anzahl von Zugriffen bei der (erfolglosen) Suche nach den Werten 10 und 15?

Aufgabe 8: Kruskal-Algorithmus

a) Bestimmen Sie für den gegebenen Graphen mit dem Kruskal-Algorithmus einen minimalen aufspannenden Baum und geben Sie sein Gewicht an.



b) Skizzieren Sie die *Union-Find-Datenstruktur* die bei Ihrer Lösung aus a) entsteht.
c) Wieviele verschiedene minimale aufspannende Bäume gibt es für diesen Beispielgraphen? Begründen Sie Ihre Antwort.