Universität Tübingen
Fachbereich Informatik
Sebastian Otte

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Praktische Informatik 2

Übungsblatt 11

(Unbewertet - keine Abgabe)

Bei Rückfragen wenden Sie sich bitte an info2-support@cs.uni-tuebingen.de.

Lernziele

Dieses Übungsblatt soll folgende Lerninhalte vertiefen:

- 11.1: Vertiefung Ihres Wissens über **Such- und Sortieralgorithmen**.
- 11.2: Verinnerlichung der Funktionsweise der behandelten **Sortierverfahren**.

Aufgaben

11.1 Wissenstest Suchen und Sortieren

Geben Sie an, ob die nachfolgenden Aussagen wahr oder falsch sind und begründen Sie Ihre Antwort.

a) Sei L eine Liste von Spielkarten, deren Elemente aus ihrem Wert und ihrer Farbe bestehen:

$$L = [(7, \spadesuit), (3, \heartsuit), (5, \spadesuit), (2, \heartsuit), (5, \heartsuit)]$$

Sortiert man L zweimal nacheinander mit Mergesort, zunächst nach der Farbe (sei $\heartsuit < \spadesuit$) und danach nach dem Wert, so ist garantiert, dass $(5, \heartsuit)$ vor $(5, \spadesuit)$ liegt.

b) Quicksort ist unabhängig von der Eingabe schneller als Bubblesort.

c) Bei Insertionsort ist das Ermitteln der richtigen Position eines Elements in der Ausgabe vergleichsweise einfach, die nötigen Verschiebungen können jedoch aufwändig sein.

d) Ist die Sortiertheit eines Arrays bekannt, so kann ein Element schneller gefunden werden, als das Array linear zu durchsuchen.

e) Sortierverfahren, die nach dem Divide-and-conquer-Prinzip arbeiten, haben in der Regel eine Laufzeitkomplexität von $O(\log n)$.

11.2 Sortierverfahren

Gegeben sei die nachfolgende Zahlensequenz:

```
4, 12, 2, 11, 9, 0, 3, 10, 8, 5, 1, 7
```

Sortieren Sie diese Sequenz jeweils mittels der unten aufgeführten Sortierverfahren per Hand und skizzieren Sie dabei fundamentale Zwischenschritte (stellen Sie beispielsweise bei Bubblesort den aktuellen Zwischenstand nur nach jedem äußeren Schleifendurchlauf dar).

a) Selectionsort

b) Insertionsort

c) Bubblesort

d) Mergesort

e) Quicksort

Hinweis: Zur besseren Verinnerlichung kann es auch hilfreich sein, dass Sie alle Algorithmen nochmal von Hand nachprogrammieren.

11.3 Weitere Aufgaben zur Klausurvorbereitung

Wichtiger Hinweis: Die nachfolgenden Aufgaben decken nur einen Teil der in der Klausur abgeprüften Inhalte ab und sollen Ihnen lediglich exemplarisch dazu dienen, das erwartete Niveau und die Art der Fragestellungen einzuschätzen. Es besteht dennoch keine verbindliche Garantie, dass die Klausur Aufgaben in exakt der angegebenen Form enthalten wird.

a) Geben Sie jeweils an, ob die nachfolgenden Code-Fragmente zulässig sind oder nicht. Begründen Sie eine negative Antwort.

i)
```
float a = 1;
float b = 2;
boolean c = a > b;
```

ii)
```
Integer i = 42;
Float f = i;
```

iii)
```
int i = 12;
Object o = i;
```

iv)
```
int a = 12;
float f = a * 1.2;
```

b) Berechnen Sie nachfolgende Aufgaben in der 2er-Komplement-Darstellung unter Verwendung von 8-Bit langen Zahlen. Wandeln Sie das Ergebnis jeweils wieder in eine Dezimalzahl um.

i) 78 - 42

ii) -12 + 50

c) Implementieren Sie eine Methode `boolean isSymmetric(double[][] A)`, welche überprüft, ob die angegebene Matrix symmetrisch ist ($\mathbf{A} = \mathbf{A}^\top$, d.h $a_{ij} = a_{ji}$). Sie dürfen hier trotz des Datentyps `double` exakt auf Gleichheit prüfen. Sie dürfen zudem annehmen, dass `A` immer quadratisch ist und alle Zeilen gleich lang und nicht `null` sind.

d) Gegeben sei die nachfolgende Klasse eines Listen-Knotens:

```java
public class Node {
    public int value;
    public Node next;

    public Node(final int value, final Node next) {
        this.value = value;
        this.next = next;
    }
}
```

i) Implementieren Sie eine Methode `boolean isSorted(Node node)`, die `true` zurückgeben soll, wenn die übergebene Liste absteigend sortiert ist und `false`, wenn nicht.

ii) Schreiben Sie eine Methode `Node joinPairwise(Node node)`. Die Methode soll immer paarweise zwei benachbarte Knoten einer Liste zusammenführen. D.h. aus zwei Knoten soll jeweils ein neuer Knoten entstehen, dessen Element (Variable `value`) die Summe der beiden ursprünglichen Elemente enthalten soll. Ist die Länge der Liste ungerade, soll das letzte Element einfach erhalten bleiben.

Beispiel:

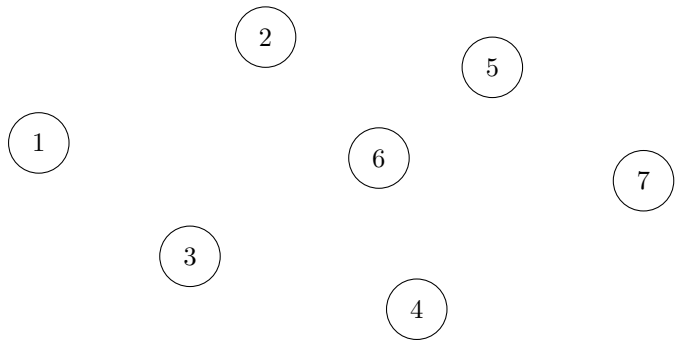
```
input:    5,  3,  1,  2,  4,  7,  0,  2,  1
           \ /     \ /     \ /     \ /    |
output:    8,      3,     11,      2,     1
```

e) Gegeben sei die nachfolgende Adjazenzmatrix **A**:

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

i) Handelt es sich um einen gerichteten oder ungerichteten Graphen? Begründen Sie Ihre Antwort.

ii) Vervollständigen Sie den unten abgebildeten Graphen durch Einzeichnen der fehlenden Kanten zwischen den Knoten, sodass der Graph der Adjazenzmatrix **A** entspricht.



f) Gegeben sei die nachfolgende generische Klasse:

```
public class Pair<T, U> {
    private T first;
    private U second;

    public T getFirst() { return this.first; }
    public U getSecond() { return this.second; }
}
```

i) Schreiben Sie einen spezialisierten Konstruktor, der beide Instanzvariablen mit gegebenen Werten initialisiert. Sie können den Konstruktor direkt aufschreiben und müssen das Klassengerüst nicht erneut wiederholen.

ii) Erstellen Sie nun eine Instanz der generischen Klasse `Pair` mit `String` und `Integer` als konkrete Typen für `first` und `second`.

iii) Schreiben Sie ein Interface `PairConverter`. Dieses soll eine Methode `convert` vorsehen, welche eine Instanz von `Pair` erwartet und wieder eine Instanz von `Pair` zurückliefert. Dabei sollen alle vier Typparameter frei wählbar sein, d.h. es soll beispielsweise möglich sein, ein `Pair` aus `String` und `Integer` in ein `Pair` aus `Long` und `Double` umzuwandeln. Sie können sich aussuchen, ob die Methode `convert` oder das gesamte Interface generisch ist.

g) Gegeben sei der nachfolgende Algorithmus:

```
public static int foo(int n) {
    int m = 0;
    for (int i = 0; i < (n/2); i++) {
        for (int j = i; j >= 0; j--) {
            int k = 1;
            while (k < n) {
                k = k * 2;
                m++;
            }
        }
    }
    return m;
}
```

Sie sollen die Laufzeit des Algorithmus (obere Schranke) in Abhängigkeit von n mittels der O-Notation abschätzen. **Hinweis:** Verwenden Sie die Regel $O(f(n)) \cdot O(g(n)) = O(f(n) \cdot g(n))$.