Universität Konstanz
Fachbereich Informatik & Informationswissenschaft
Prof. Dr. Ulrik Brandes / Dr. Andreas Karrenbauer / Dr. Jürgen Lerner / Martin Mader

Algorithmen und Datenstrukturen
WS 2010/2011

2. Übungsblatt

Ausgabe: 27. Oktober 2010 **Abgabe:** 03. November 2010, 10 Uhr
Die Bearbeitung in Zweiergruppen ist ausdrücklich erwünscht.

Aufgabe 1: SelectionSort und QuickSort – Beispiel 4 Punkte

Führen Sie die Algorithmen SelectionSort und QuickSort auf der folgenden Eingabe aus

7	3	9	8	5	11	1

Geben Sie bei SelectionSort den Zustand des Arrays M nach jeder Ausführung der ersten **for** Schleife (d. h., nach jeder Vertauschung) an. (1 Punkt)

Geben Sie bei QuickSort den Zustand des Arrays M jeweils vor den rekursiven Aufrufen an. Markieren Sie welche Elemente schon einmal als Pivot-Element gedient haben und auf welche Teilarrays die nächsten rekursiven Aufrufe angewandt werden. (3 Punkte)

Aufgabe 2: QuickSort – *Advocatus Diaboli* 6 Punkte

Sehen Sie sich den Pseudocode für QuickSort an. Stellen Sie sich vor Sie hätten es mit einem *Advocatus Diaboli* (Anwalt des Teufels) zu tun, der Ihnen bei jeder elementaren Operation (Vergleich, Speicherzuweisung, rekursiver Aufruf, etc) weismachen will, dass diese Operation unnötig sei oder sogar zu falschen Ergebnissen führe.

(a) Widerlegen Sie ihn indem Sie kurz erläutern, was in den jeweiligen Schritten gemacht wird, warum dies korrekt ist und warum es nicht ohne geht.

(b) Oder hat der Anwalt doch recht und der Pseudocode für QuickSort könnte vereinfacht werden?

[Bitte wenden]

```
quicksort(M, l, r) begin
    if l < r then
        i ← l + 1;      j ← r
        p ← M[l]
        while i ≤ j do
            while i ≤ j and M[i] ≤ p do
                i ← i + 1

            while i ≤ j and M[j] > p do
                j ← j − 1

            if i < j then
                vertausche M[i] und M[j]


        if l < j then
            vertausche M[l] und M[j]
            quicksort(M, l, j − 1)

        if j < r then
            quicksort(M, j + 1, r)
end
```

Antwortbeispiel: "Der Vergleich in Zeile 2 (**if** $l < r$) testet ob das Teilarray noch aus zwei oder mehr Elementen besteht. Bei weniger als zwei Elementen ist das Teilarray trivialerweise schon sortiert und es ist nichts mehr zu tun. Allerdings ist diese Abfrage unnötig; würde der Algorithmus mit $l \geq r$ aufgerufen, so wären die Bedingungen der **while**-Schleife und der beiden letzten **if**-Abfragen falsch und der Algorithmus würde terminieren.

[Bitte wenden]

Aufgabe 3: Quicksort-Pivotstrategie **10 Punkte**

Eine verbesserte Pivotstrategie für Quicksort heißt *Median-von-3*: Aus dem zu sortierenden Feld $M[1, \ldots, n]$ werden drei Elemente bestimmt und das mittlere davon als Pivot p verwendet (wir nehmen an, dass das Feld n unterschiedliche Werte enthält und $n \geq 3$ gilt).

(a) Erläutern Sie den Vorteil dieser Methode im Vergleich zum herkömmlichen ($p \leftarrow M[1]$) und randomisierten Quicksort ($p \leftarrow M[\text{zufällig aus } \{1, \ldots, n\}]$). Für welche Eingaben sind diese drei Verfahren besser oder schlechter geeignet? (3 Punkte)

(b) Implementieren Sie alle drei Varianten (herkömmlich, randomisiert, Median-von-3) in Java. Implementieren Sie das Interface `material.u02.IQuickSort` durch eine *abstrakte* Klasse `u02.<name1_name2>.QuickSort` und drei *abgeleitete konkrete* Klassen

- `u02.<name1_name2>.FirstElementQuickSort`,
- `u02.<name1_name2>.RandomizedQuickSort` und
- `u02.<name1_name2>.MedianOf3QuickSort`

je nach Pivotstrategie. (5 Punkte)

Hinweis: In der Praxis wird *Median-von-3* so realisiert, dass als Pivot der Median des ersten, letzten und mittleren Elements des Feldes verwendet wird.

(c) Realisieren Sie eine Ordnung auf Paaren von ganzen Zahlen. Erstellen Sie dazu eine Klasse `u02.<name1_name2>.IntegerPair`, die das `Comparable<IntegerPair>`-Interface implementiert.

Seien $p_1 = (i_1, i_2)$ und $p_2 = (j_1, j_2)$ zwei Paare. Es gilt $p_1 = p_2$ genau dann, wenn $i_1 = j_1$ und $i_2 = j_2$. Ferner gilt

$$p_1 < p_2 \Leftrightarrow \begin{cases} i_1 < j_1 \\ i_1 = j_1 \text{ und } i_2 < j_2 \end{cases}$$

und $p_1 > p_2$ sonst.

Stellen Sie einen Konstruktor `public IntegerPair(Integer firstInt, Integer secondInt)` bereit. Überschreiben Sie außerdem die `toString()`-Methode dieser Klasse, so dass ein Paar (i_1, i_2) genau in dieser Schreibweise (mit Klammern, Komma als Trennzeichen, ohne Leerzeichen) ausgegeben wird. (2 Punkte)