Do., 27.9.2012 Informatik Q1 M (Kr)

1. Klausur Q1.1 (12/I)

Dauer: 4·45 min = 180 min (9:15 bis 12:15 Uhr) Name: _____www.r-krell.de____

Hilfsmittel: Taschenrechner

* *Achte auf sorgfältige Darstellung mit vollständigem, nachvollziehbarem Lösungsweg!* *
* *Kommentiere deine Programme!* *

1 Verwandlungen

Früher hat man Zahlwörter wie Gros (144), Schock (60) und Dutzend (12) benutzt. So könnten z.B. 88 = Schock + Dutzend + Dutzend + 4 oder 289 = Gros + Gros +1 dargestellt werden. Die Methode *aufgabe1* liefert bei Übergabe von 88 oder 289 als *zahl* allerdings nicht immer das erhoffte Ergebnis:

a) Vollziehe die gegeben Methode *aufgabe1* nach und notiere die für 88 und 289 erzeugten Ausgaben.

b) Ändere die Methode so ab, dass sie immer die erhoffte Ausgabe mit möglichst großen Zahlwörtern liefert (Bitte Methode vollständig neu aufschreiben)!

```
public void aufgabe1 (int zahl)
{
  while (zahl >= 12)
  {
     if (zahl >= 144)
     {
       System.out.print ("Gros + "); zahl = zahl-144;
     }
     if (zahl >=  60)
     {
       System.out.print ("Schock + "); zahl = zahl-60;
     }
     if (zahl >=  12)
     {
       System.out.print ("Dutzend + "); zahl = zahl-12;
     }
  }
  System.out.println (""+zahl);
}
```

7.0	1.2	5.8	4.2	7.0	7.6	3.6	...

Für die Aufgaben 2, 3 und 5 steht die gezeigte Reihung *reihe* mit 7 eingetragenen Kommazahlen zur Verfügung, die in der Klasse *Klausur* definiert ist:

```
public class Klausur
{
  double [ ]  reihe = new double [500];
  int anzahl = 0;

  public Klausur ( )    // Konstruktor
  {
    reihe[0] = 7.0;   reihe[1] = 1.2;   reihe[2] = 5.8;   reihe[3] = 4.2;
    reihe[4] = 7.0;   reihe[5] = 7.6;   reihe[6] = 3.6;   anzahl = 7;
  }
 ...
```

2 Sortierverfahren

a) Sortiere die sieben Zahlen in der *reihe* (s.o.) von Hand nach der Bubblesort-Methode. Mache alle Vertauschungen nachvollziehbar und schreibe den Inhalt der Reihung nach jedem Durchgang auf!

b) Sortiere die *reihe* von Hand mit Quicksort, wobei immer das Element ganz links im zu sortieren Teilbereich als Pivot-Element dienen soll (d.h. ergänze im Struktogramm *int pivot* =

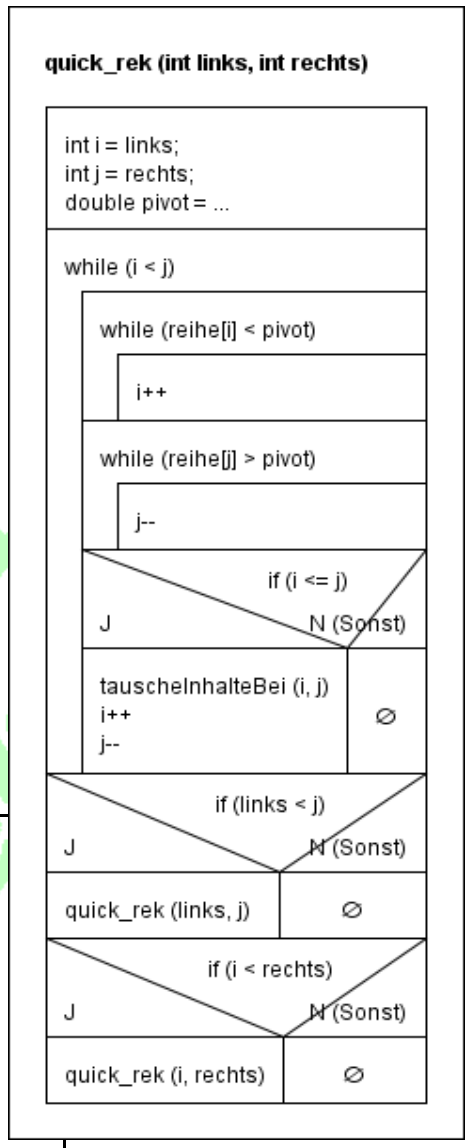
reihe[links]). Mache alle Vertauschungen und den Beginn jeder neuen Rekursionsstufe deutlich!

c) Beschreibe Quicksort als Beispiel eines Verfahrens mit „Teile-und-herrsche“-Strategie.

d) Erläutere, wie viele Vergleiche von Feldinhalten (hier: unter Beteiligung einer Kommazahl) und wie viele Vertauschungen im besten Fall in Abhängigkeit von der *anzahl = n* für Bubblesort und Quicksort gemacht werden müssen. Gib auch die O-Notation des Aufwands an.

e) Wie d), nur für den schlechtesten Fall.

f) Im Struktogramm soll nun nicht ein Feldelement, sondern der Mittelwert *int pivot = (reihe[links] +reihe[rechts)/2.0;* als Pivotelement genommen werden. Funktioniert das Verfahren trotzdem? Vollziehe wieder von Hand nach und entscheide!



3 Wieder soll die *reihe* 7.0 1.2 5.8 4.2 7.0 7.6 3.6 sortiert werden. Weil nur Kommazahlen zwischen 0.0 und 9.9 vorkommen, wird folgender Algorithmus vorgeschlagen:

```
public void aufgabe3( )
{
 int [ ]  kontrolle = new  int [100];
 for (int j = 0; j < 100; j++)      // zur Vorbereitung alles auf Null
 {
  kontrolle [j] = 0;
 }
 for (int i = 0; i < anzahl; i++)  // am 10-fachen der Reihenwerte
 {                                 // die Kontrolle erhöhen
  kontrolle [ (int)(10.0*reihe[i]) ]++;
 }
 int i=0;
 for (int j = 0; j < 100; j++)     // Kontrolle auslesen, reihe neu füllen
 {
   while (kontrolle [j] > 0)
   {
     reihe[i] = j/10.0;
     kontrolle[j]--;
     i++;
   }
 }
}
```

a) Begründe, dass für jede *anzahl = n* der Aufwand der Methode *aufgabe3* (nämlich die Anzahl der Schleifendurchläufe bzw. das Schreiben oder Lesen in/von Komponenten von *kontrolle*) durch 100 + *n* + 100 gegeben ist und damit von der Ordnung O(*n*) linear mit *n* ist!

b) Erläutere die Idee des Verfahrens (gib dabei an, in welchen Komponenten von *kontrolle* hier nach der zweiten Schleife von Null verschiedene Werte stehen -- welche/warum?) und entscheide begründet, ob immer richtig sortiert wird.

c) Überlege und begründe, warum wir nicht für sämtliche Sortieraufgaben immer nur das vorstehende Verfahren einsetzten, wo der Aufwand mit O(n) doch offensichtlich besser ist als der

Aufwand von Quicksort oder Bubblesort.

d) Gelegentlich kann es passieren, dass nach dem Sortieren aus einer 6.7 eine 6.6, eine 6.6999999997, eine 6.7000000002 oder eine 6.8 geworden ist. Erkläre dieses Phänomen unter Hinweis auf ein ähnliches Problem. Abhilfe?

4 Suchverfahren

Jetzt soll die Reihung *reihe* absteigend sortiert sein: 7.6 - 7.0 - 7.0 - 5.8 - 4.2 - 3.6 - 1.2
In dieser (absteigend) sortierten Reihe wird gesucht – z.B. nach der Stelle mit dem Wert 3.6 (Stelle 5) oder nach der Stelle mit dem nicht vorhandenen Wert 6.5 (Stelle -1).

a) Schreibe eine sequenzielle Suche in Java (die die Sortierung nicht ausnutzt) und gib zusätzlich an, wie viele Elemente von *anzahl = n* Elementen im besten bzw. im schlimmsten Fall mit dem gesuchten Wert verglichen werden müssen.

b) Beschreibe die binäre Suche (Halbierungsverfahren) für eine absteigend sortierte *reihe* in Worten und gib auch hier den Vergleichs-Aufwand für den besten und den schlechtesten Fall an.

c) Notiere die binäre Suche für die absteigende Sortierung [wie in b) beschrieben] in Java – egal, ob iterativ oder rekursiv.

5 Weitere Arbeit mit Reihungen

a) Schreibe eine Java-Methode, die zurück gibt, an welcher Stelle die größte Zahl im belegten Teil der *reihe* steht (bei der unsortierten reihe wie vor Aufgabe 2 müsste 5 zurückgegeben werden, weil die größte Kommazahl 7.6 an der Stelle 5 steht)

a1) iterativ

a2) rekursiv (mit zusätzlicher Methode zum Start)