Di.., 30.10.'01 Informatik 12_M (Kr)

Übungen mit sortierten Reihen

Dauer: 2 Schulstunden (Rest als Hausaufgabe) – Kasten nach Antwort 1a abtippen und zusätzlich Methoden zu 1 b1, 2 a und 2 b in gleiche Datei eintippen sowie weitere Fragen schriftlich beantworten!

1. Gegeben ist nebenstehendes Programm mit den Reihungen *auf* und *ab*, in denen schon zufällig ausgewählte Zahlen in aufsteigend sortierter bzw. absteigend sortierter Reihenfolge stehen.

a) Was erscheint bei der Ausführung von *starte* auf dem Bildschirm? Muss vorher noch eine andere Methode ausgeführt werden? Und welchem Zweck dient der unterstrichene Teil in *public void zeige (int[] reihe)* ?

b) Auch die Zahlen von *ab* sollen aufsteigend sortiert werden (sodass nachher *ab* = {20, 33, .., 89} ist):

b1) Schreiben Sie eine Methode *dreheUm*. Schreiben Sie dazu auch die Methode *tausche*, die von *dreheUm* benutzt wird! Achtung: *dreheUm* soll bei beliebiger (gerader oder ungerader) Anzahl von Elementen in der Reihung funktionieren!

b2) Vergleichen Sie den Aufwand von b1) (in Abhängigkeit von der Anzahl der Element) mit dem Aufwand eines der bekannten einfachen Sortierverfahren, mit dem man die Elemente in *ab* ebenfalls hätte ordnen können.

```java
import stiftUndCo.*;

public class UebungenMitSortiertenReihen
{
  Bildschirm schirm = new Bildschirm();
  BuntStift  stift  = new BuntStift();
  int[] ab  = {89, 83, 76, 75, 72, 68, 59, 53, 46, 33, 20};
  int[] auf = {17, 28, 31, 35, 36, 47, 51, 53, 64, 69};

  public void zeige (int[] reihe)
  {
    stift.bewegeBis (10, stift.vPosition()+12);
    for (int i=0; i < reihe.length; i++)
    {
      stift.schreibe (" "+reihe[i]);
    }
  }

  public void starte ()
  {
    zeige (auf);
    zeige (ab);
  }
```

2. Suchen in einer sortierten Reihung:

a) Version a: Um eine Zahl in einer sortierten Reihung zu finden (z.B. 51 in *auf* aus Aufg. 1), beginnt man mit der 0-ten Stelle der Reihung, untersucht dann die 1., die 2., usw. und endet, wenn die gesuchte Zahl gefunden wurde oder wenn klar ist, dass die Zahl nicht mehr in der Reihe stehen kann (Wird z.B. 33 in *auf* gesucht, braucht man nicht weiter als bis zur 35 gehen. Dahinter darf 33 wegen der Sortierung nicht mehr stehen!). Schreiben Sie eine passende Methode *findeA (int zahl)* ins Programm von Aufgabe 1, die entweder die Fundstelle (bei *zahl*=51 z.B. „Gefunden an Stelle 6") oder „Nicht gefunden“ auf den Bildschirm schreibt.

b) Version b: (Halbierungsmethode) Hier beginnt man zunächst mit *sprungweite = reihe.length / 2;* und *index = 0;*. Dann wird *index = index+sprungweite*[1] gesetzt und *reihe[index]* kontrolliert. Findet sich hier die gesuchte Zahl noch nicht, wird die Sprungweite halbiert und der neue Index entweder durch *index = index - sprungweite* oder *index = index + sprungweite* bestimmt, je nach dem, ob links oder rechts vom zuletzt kontrollierten Element gesucht werden muss. Jedenfalls wird an der so berechneten neuen Stelle wieder *reihe[index] == zahl ?* überprüft usw. Schreiben Sie die passende Java-Methode *findeB*, die sonst wie *findeA* reagiert.

c) Vergleichen Sie c1) den Aufwand im schlimmsten Fall (‘worst case’) bzw. c2) den mittleren Aufwand von *findeA* und *findeB* bei der Suche nach einer Zahl in einer Reihung von 10, 100 oder 1000 Elementen! Macht es einen wesentlichen Unterschied, ob die gesuchte Zahl tatsächlich in der Reihung vorhanden ist oder nicht? Vergleichen Sie auch mit der Suche in einer unsortierten Reihung (wie?)

---

[1]) Das ‘=’-Zeichen ist das Zuweisungszeichen aus Java: der rechts genannte oder berechnete Wert kommt in die Variable links vom (vor dem) „=“-Zeichen.