Institut für Informatik
Heinrich-Heine-Universität Düsseldorf
Prof. Dr. Alice McHardy
Robin Küppers
Sebastian Konietzny
Aaron Weimann

WS 2013/2014
03.12.2013
Übungsblatt 6

Einführung in die Informatik I
Übungen zur Vorlesung
Musterlösung

Die Lösungen der zu bearbeitenden Aufgaben müssen spätestens bis **Dienstag**, den **10.12.2013, 11:59 Uhr** über https://ilias.uni-duesseldorf.de abgegeben werden. Bitte achten Sie immer auf einen vollständigen und nachvollziehbaren Lösungsweg, da ansonsten Punkte abgezogen werden können! Vergessen Sie bitte nicht auf Ihrer Abgabe Ihren Namen und Ihre Matrikelnummer zu vermerken (bei Gruppenabgaben müssen sämtliche Mitglieder der Gruppe aufgeführt werden).

6.1 Binäre Suche (4 Punkte)

Sei a ein aufsteigend sortiertes Array mit ganzzahligen Werten:

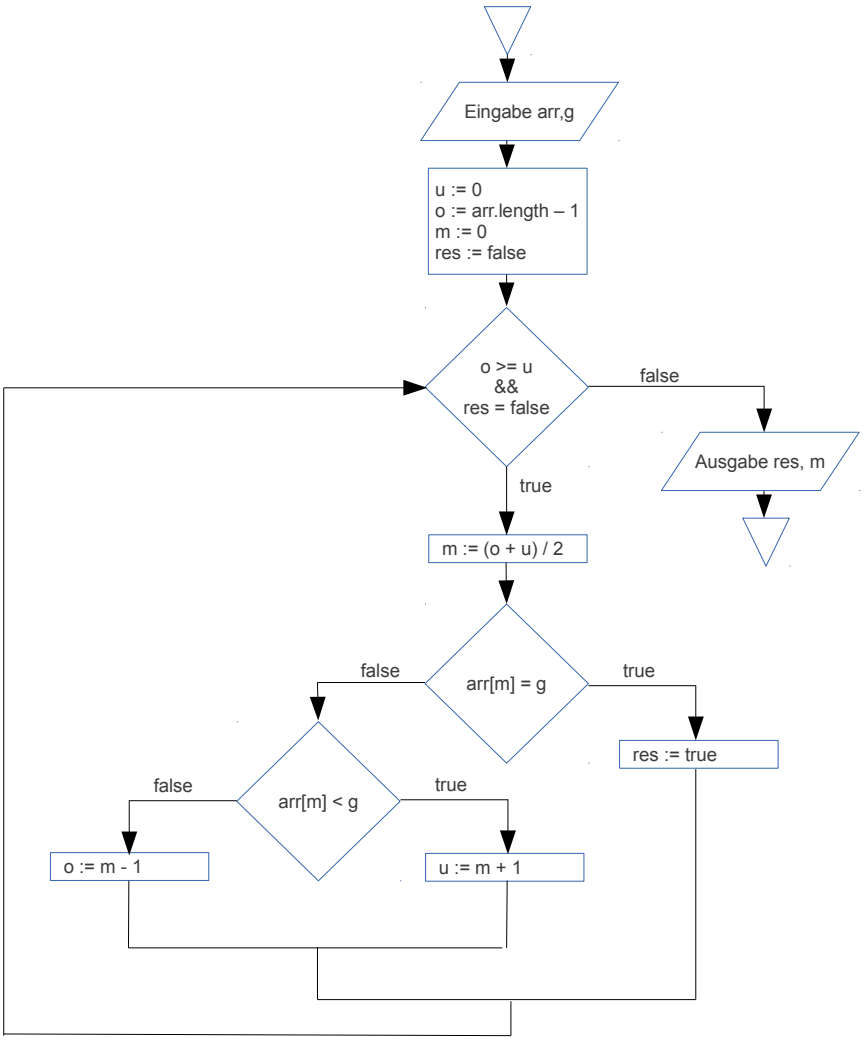
```
a = [ 1, 2, 3, 4, 5, 6, 7, 8, 9 ]
```

1. Suchen Sie in `a` nach dem Wert 4. Verwenden Sie dafür den binären Suchalgorithmus, den Sie in der Vorlesung kennengelernt haben. Geben Sie für jeden Schritt das verbleibende Teilarray an, das noch durchsucht werden muss, sowie das zu vergleichende Element (Mitte), außerdem die Indexpositionen m, o und u für das mittlere Element und die obere und untere Schranke. (2 Punkte)

2. Geben Sie für den binären Suchalgorithmus ein Flussdiagramm an. (2 Punkte)

Musterlösung

1.
```
Schritt 1: [1, 2, 3, 4, 5, 6, 7, 8, 9] (u=0, o=8, m=4, res=false)
Schritt 2: [1, 2, 3, 4 ]               (u=0, o=3, m=1, res=false)
Schritt 3:      [ 3, 4 ]               (u=2, o=3, m=2, res=false)
Schritt 4:         [ 4 ]               (u=3, o=3, m=3, res=true)
```



Anmerkung: Anstatt der von den Folien bekannten Schreibweise für Variablenzuweisungen (*var := wert*) dürfen die Studenten natürlich auch die Java-Syntax verwenden. Aber dann muss klar zwischen dem Gleichheits- (==) und Zuweisungsoperator (=) unterschieden werden.

6.2 Sortieralgorithmen (15 Punkte)

Sei a ein Array mit ganzzahligen Werten:

```
a = [ 42, 23, 16, 15, 8, 4 ]
```

Sortieren Sie `a` aufsteigend. Verwenden Sie die folgenden Sortieralgorithmen aus der Vorlesung:

1. Insertion-Sort (3 Punkte)
2. Selection-Sort (3 Punkte)
3. Bubble-Sort (3 Punkte)
4. Merge-Sort (3 Punkte)
5. Quick-Sort (3 Punkte)

Geben Sie immer sämtliche Array-Zwischenergebnisse an!

Musterlösung

Insertion Sort:

`Schritt 1:` [23, 42, 16, 15, 8, 4]
`Schritt 2:` [16, 23, 42, 15, 8, 4]
`Schritt 3:` [15, 16, 23, 42, 8, 4]
`Schritt 4:` [8, 15, 16, 23, 42, 4]
`Schritt 5:` [4, 8, 15, 16, 23, 42]

Selection Sort:

`Schritt 1:` [4, 23, 16, 15, 8, 42]
`Schritt 2:` [4, 8, 16, 15, 23, 42]
`Schritt 3:` [4, 8, 15, 16, 23, 42]
`Schritt 4:` [4, 8, 15, 16, 23, 42]
`Schritt 5:` [4, 8, 15, 16, 23, 42]

Bubble Sort:

`Schritt 1:` [23, 16, 15, 8, 4, 42]
`Schritt 2:` [16, 15, 8, 4, 23, 42]
`Schritt 3:` [15, 8, 4, 16, 23, 42]
`Schritt 4:` [8, 4, 15, 16, 23, 42]
`Schritt 5:` [4, 8, 15, 16, 23, 42]
`Schritt 6:` [4, 8, 15, 16, 23, 42]

Merge Sort:

`split:` [42, 23, 16, 15, 8, 4]
`split:` [42, 23, 16, 15, 8, 4]
`split:` [42, 23, 16, 15, 8, 4]
`split:` [42, 23, 16, 15, 8, 4]
`split:` [42, 23, 16, 15, 8, 4]
`merge:` [**23**, 42, 16, 15, 8, 4]
`split:` [23, 42, 16, 15, 8, 4]
`merge:` [16, **23**, 42, 15, 8, 4]
`split:` [16, 23, 42, 15, 8, 4]
`split:` [16, 23, 42, 15, 8, 4]
`split:` [16, 23, 42, 15, 8, 4]
`split:` [16, 23, 42, 15, 8, 4]
`merge:` [16, 23, 42, **8**, 15, 4]
`split:` [16, 23, 42, 8, 15, 4]
`merge:` [16, 23, 42, 4, **8**, 15]
`merge:` [4, 8, **15**, 16, 23, 42]

Quick Sort:

`qsort:` [42, 23, **16**, 15, 8, 4]
`part.:` [4, 15, 8, **16**, 42, 23]
`qsort:` [4, **15**, 8, 16, 42, 23]
`part.:` [4, 8, **15**, 16, 42, 23]
`qsort:` [**4**, 8, 15, 16, 42, 23]
`part.:` [**4**, 8, 15, 16, 42, 23]
`qsort:` [4, 8, 15, 16, **42**, 23]
`part.:` [4, 8, 15, 16, 23, **42**]

6.3 Vermischtes (6 Punkte)

1. Funktioniert die binäre Suche auch für absteigend sortierte Arrays? Müsste der Algorithmus dafür grundlegend verändert werden? Begründen Sie. (1 Punkt)

2. Wieso wird bei Insertion-Sort keine `swap`-Operation (Dreieckstausch) benötigt, obwohl Elemente getauscht werden? Wäre es sinnvoll die `swap`-Operation zu verwenden? (2 Punkte)

3. Geben Sie eine `Java`-Methode

   ```
   public static void rotate(int[] arr, int n)
   ```

   an, die alle Elemente von `arr` um `n` Positionen nach links verschiebt. Elemente, die durch die Verschiebung aus dem Array fallen würden, sollen hinten wieder angehangen werden. Wie viel Schritte braucht ihr Algorithmus um die richtige Lösung zu generieren in Abhängigkeit von der Eingabegröße n. Verwenden sie dabei keine Hilfsarrays oder andere Datenstrukturen. (Sie dürfen aber Hilfsvariablen primitiver Datentypen verwenden.) Zählen Sie nur die Anzahl der Zuweisungen. (3 Punkte)

Musterlösung

1. Die binäre Suche kann für absteigende Arrays adaptiert werden, indem die Bedingung `(arr[m] < g)` durch `(arr[m] > g)` ersetzt wird.

2. Bei Insertion-Sort werden ganze Teilarrays um eine Position nach links verschoben. Dadurch, dass der erste Wert des Teilarrays in einer Hilfsvariable gesichert wird, können die nachfolgenden Elemente einfach um eine Position nach links verschoben werden (wobei die alten Werte überschrieben werden). Der gesicherte Wert wird an die letzte Position geschoben.
   Anmerkung: Die Verwendung des Dreieckstausch wäre sogar ineffizienter, da für jede Linksverschiebung beim Einfügen eines Elementes (1 Zuweisung) ein vollständiger Dreieckstausch (3 Zuweisungen) durchgeführt werden müsste.

3. ```
   public static void rotate(int arr[], int n) {
       while(n-- > 0) {
           int tmp = arr[0];
           for (int i = 1; i < arr.length; i++) {
               arr[i - 1] = arr[i];
           }
           arr[arr.length - 1] = tmp;
       }
   }
   ```

   Der Algorithmus hat eine Laufzeit die mindestens proportional zu n. Die Laufzeit der naiven Implementierung ist quadratisch in der Eingabegröße n.
   ```