


Praktikum Algorithmen und Datenstrukturen, Teil 1	Hochschule Bochum Bochum University of Applied Sciences 
Sortieren und Suchen	

Generelles:

- Kommunikation mit uns bitte über "privaten Chat" und per audio über Breakout-Room, dem Sie zugeordnet sind, nicht über den Gruppenraum
- Im privaten Chat bitte signalisieren, wenn Sie uns sprechen wollen
- Gerne mit Stichworten: „Hilfe zu ...“, „Abgabe zu ...“, „Technische Sorgen mit ...“
- Programmiert wird auf <https://fbevirt.hs-bochum.de/#/>, auf der Maschine, die mit Ihrer HS-Kennung benannt ist.
- Die Aufgabenstellung besteht aus:
 - einer Hauptaufgabe (1 Punkt)
 - 2 zur Hauptaufgabe alternativen kleineren Aufgaben (je 1/3 Punkt). Punkte gibt es für diese nur, wenn die Hauptaufgabe **nicht** erfolgreich bearbeitet wurde.
 - und einer optionalen Zusatzaufgabe (je 1/3 Punkt). Sollten Sie eine oder mehrere der Zusatzaufgaben erledigen, können Sie hiermit Schwächen in der Hauptaufgabe (in diesem oder zukünftigen Praktikumsterminen ausgleichen).

Aufgabenbeschreibungen:

Hauptaufgabe H: Sortieren (1 Punkt)

- Schreiben Sie eine Anwendung, welche den folgenden Sortieralgorithmus umsetzt, um die Eingabe in absteigender (und nicht wie in der Vorlesung: in aufsteigender) Reihenfolge zu sortieren.**
- Verifizieren Sie, dass der Algorithmus das angegebene Beispielfeld korrekt sortiert, d.h. in absteigender Reihenfolge der Eingabezahlen.

Die main-Methode ist bereits vorgegeben, siehe folgenden Code-Abschnitt.

Sie müssen noch die Methode `static void sortiere(int[] a, int begin, int end)` implementieren, die als Parameter neben dem Eingabefeld mindestens noch 2 Indizes hat, die den Teilbereich des Eingabefeldes beschreiben, der im Aufruf bearbeitet werden soll.

Hinweis: Wenn Sie sich entscheiden, eine Quicksort-Variante umzusetzen, sind Sie frei bei der Wahl des Pivot-Elementes. Insbesondere müssen Sie dieses nicht optimal wählen.

```
public static void main(String[] args) { int[] a = {99, 0, 33, 27, 87, 80, 65, 77, 39, 88};
    System.out.println("Zu sortierendes Array:");
    for(int i = 0; i < a.length; i++){
        System.out.print(a[i] + " ");
    }
    System.out.println("");
    sortiere(a, 0, a.length-1);
    System.out.println("Sortiertes Array:");
    for(int i = 0; i < a.length; i++){
        System.out.print(a[i] + " ");
    }
}
```

Beschreibung des Algorithmus zur Hauptaufgabe.

Beachte: Diese Variante ist für das Sortieren in **aufsteigender** (und nicht wie in der Hauptaufgabe **gefordert**: absteigender) Reihenfolge beschrieben.

Hilfestellung: Der Algorithmus nutzt dieselbe rekursive Struktur wie der QuicksortAlgorithmus aus der Vorlesung.

1. Schritt zur Implementierung der Methode *sortiere*

Testen Sie zunächst, ob *begin* < *end* ist. Falls ja, teilen Sie das Eingabefeld mit Hilfe eines gewählten Pivot-Elementes *x* aus dem Eingabefeld auf, entsprechend etwa der folgenden Anleitung.

2. Schritt zur Implementierung der Methode *sortiere*

Anleitung zur Aufteilung des Eingabefeld mit Hilfe eines gewählten Pivot-Elementes *x* aus dem Eingabefeld:

Linker Pfeil: Wandert (von links) so viele Positionen nach rechts, bis angezeigte Zahl größer oder gleich *x* ist.

Rechter Pfeil: Analog, nur von rechts nach links, und auf der Suche nach Elementen kleiner oder gleich *x*.

Tausch und weiterwandern um jeweils eine Position, falls Position des linken Pfeils kleiner gleich Position des rechten Pfeils.

Die Pfeile können Sie als int-Variablen implementieren.

3. Schritt zur Implementierung der Methode *sortiere*

Führen Sie den 2. Schritt innerhalb einer Schleife durch, welche beendet wird, wenn die Seiten der Pfeile vertauscht sind. Dann ist ein Durchlauf implementiert, siehe Vorlesung.

4. Schritt zur Implementierung der Methode *sortiere*

Machen Sie aus *sortiere* eine rekursive Methode, in der die obigen Schritte ausgeführt werden. Sie ruft sich mit dem Array, *begin* und der aktuellen Position des ursprünglich rechten Pfeils als Parameter auf, um den linken Teil des aktuellen Array-Abschnitts weiter zu bearbeiten. Sie ruft sich mit dem Array, der aktuellen Position des ursprünglich linken Pfeils und *end* als Parameter auf, um den rechten Teil des aktuellen Array-Abschnitts weiter zu bearbeiten.

Ersatz-Teilaufgabe a) zur Hauptaufgabe H (1/3 Punkt): Implementieren Sie eine (im *average-case* laufzeiteffiziente) Quicksort-Variante Ihrer Wahl und verifizieren Sie Ihre Implementierung, wie in der Hauptaufgabe, in ii, beschrieben. Wie in der Hauptaufgabe soll die Eingabe in absteigender (und nicht wie in der Vorlesung: in aufsteigender) Reihenfolge sortiert werden.

Ersatz-Teilaufgabe b) zur Hauptaufgabe H (1/3 Punkt): Implementieren Sie eine (im *worst-case* laufzeiteffiziente) Mergesort-Variante Ihrer Wahl und verifizieren Sie Ihre Implementierung, wie in der Hauptaufgabe, in ii, beschrieben. Wie in der Hauptaufgabe soll die Eingabe in absteigender (und nicht wie in der Vorlesung: in aufsteigender) Reihenfolge sortiert werden.

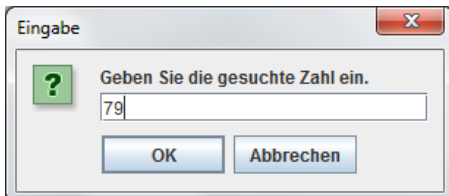
Optionale Zusatzaufgabe Z1: Schreiben Sie eine Anwendung, welche die **binäre Suche**

umsetzt.

Beachten Sie: Der Such-Algorithmus soll als Eingabe ein nicht aufsteigend, sondern absteigend sortiertes Array enthalten!

Zum Testen deklarieren und implementieren Sie dazu in der main-Methode das Array mit den Zahlen **99,90,79,75,70,60,55,37,23,17,16,9,8,6,5,1**.

Der Inhalt des Arrays soll mit *System.out.print* ausgegeben werden, siehe entsprechende Zeile der Eingabeaufforderung unten. Dann soll ein Eingabefenster erscheinen, in welchem der Anwender eine ganze Zahl eingeben kann. Die Position der Zahl in dem Array wird dann als Ergebnis in einem Ausgabefenster ausgegeben. Falls die Zahl nicht gefunden wird, wird -1 ausgegeben.



Die Berechnung des Ergebnisses erfolgt in einer rekursiven Methode, wie im in der Vorlesung als Algorithmus besprochen.

Die Zwischenergebnisse sollen mittels *System.out.println* ausgegeben werden.

Hinweis: Zur Erzeugung der graphischen Elemente nutzen Sie z.B. `JOptionPane.showMessageDialog()` bzw. `JOptionPane.showInputDialog()`.