Übungsblatt 2

Ausgabe: 2.4.2020
Abgabe: 16.4.2020 (7:00 Uhr)
Testat: 17.4.2020 (9:45 - 13:40 Uhr)

Allgemeines

Für die einzelnen Sortierverfahren sollen Sie den jeweiligen Aufwand berechnen, indem Sie:

- die **Anzahl der Schlüsselvergleiche**
- die **Anzahl der Vertauschungen**
- und die **Anzahl der Durchläufe** bzw. **Rekursionen** zählen.

Zum Testen (während der Entwicklung) wäre es für alle Sortierverfahren sinnvoll, die schrittweise Veränderung der Sortierfolge zu protokollieren. D.h. bei einer Vertauschung soll die Sortierfolge, unter besonderer Kennzeichnung der vertauschten Elemente ausgegeben werden. Und nach jedem Durchlauf soll der aktuelle Zustand der Sortierfolge ausgegeben werden.

Dazu schreiben Sie am besten eine geeignete (Hilfs-) Methode.

Sie finden im Team-00 Repo außer den JUnit-Tests sowohl ein `Interface` zum Sortieren, als auch die Klasse `StatObj` mit entsprechenden Attributen (Anzahl Vergleiche, Anzahl Vertauschungen und Anzahl Durchläufe bzw. Rekursionsschritte) und Methoden, um die o.g. Werte zu sammeln.

Implementieren Sie in den jeweiligen Klassen für die nachfolgenden Sortieralgorithmen die Methode `sort` aus dem Interface `SortInterface` und sammeln Sie die Statistikdaten (in dem Statistik-Objekt der Klasse `StatObj`.
Denken Sie bei allen Programmen an Kommentare!

Aufgabe 1: Shaker Sort - *Programmieraufgabe* - 30 Punkte

Als weitere Verbesserung des Bubble Sort wurde der Shaker Sort Algorithmus entwickelt. Kurz beschrieben, arbeitet er wie folgt: Die prinzipielle Arbeitsweise ist die des Bubble Sort, nur ändert sich die Richtung aufeinander-folgender Durchläufe. D.h. einmal läuft man von links nach rechts und vertauscht die Elemente. Beim nächsten Mal läuft man von rechts nach links und vertauscht die Elemente.

Beispiel:

```
F0 =   44    55    12    42    94    18    6     67    (Ausgangsfolge)
F1 =   44    12    42    55    18    6     67    94    (nach dem 1. Durchlauf)
F2 =   6     44    12    42    55    18    67    94    (nach dem 2. Durchlauf)
F3 =   6     12    42    44    18    55    67    94    (nach dem 3. Durchlauf)
F4 =   6     12    18    42    44    55    67    94    (nach dem 4. Durchlauf)
```

Der 1. Durchlauf erfolgt von unten nach oben (links nach rechts), der 2. Durchlauf erfolgt von rechts nach links (oben nach unten) etc.

Implementieren Sie in der Klasse `ShakerSort` die Methode `sort` aus dem Interface `SortInterface` und sammeln Sie die Statistikdaten, wie oben beschrieben.

Aufgabe 2: Insertion Sort - *Programmieraufgabe* - **30 Punkte**

Programmieren Sie den Insertion Sort und dann eine Variante des in der Vorlesung vorgestellten Insertion Sort. Diese Variante unterscheidet sich von der auf dem Folienskript dadurch, dass die Einfügeposition für das aktuelle Element mittels binärer Suche im bereits sortierten Teil der Folge gesucht wird.

Implementieren Sie in der Klasse `InsertionSort` die Methode `sort` aus dem Interface `SortInterface` und sammeln Sie die Statistikdaten, wie oben beschrieben.

Aufgabe 3: Quicksort - *Programmieraufgabe* - **30 Punkte**

In der Vorlesung wurden 3 Varianten des Quicksort-Algorithmus angegeben.

Die Gruppen mit Gruppennummer 1 - 7 implementieren Sie die erste Variante von Quicksort.
Die Gruppen mit Gruppennummer 8 - 14 implementieren Sie die zweite Variante von Quicksort.
Die Gruppen mit Gruppennummer $>$ 14 implementieren Sie die dritte Variante von Quicksort.

Implementieren Sie in der Klasse `QuickSort` die Methode `sort` aus dem Interface `SortInterface` und sammeln Sie die Statistikdaten, wie oben beschrieben.

Aufgabe 4: Shellsort - *Programmieraufgabe* - Zusatzaufgabe **40 Punkte**

In der Vorlesung wurde der Shell-Sort vorgestellt. Der Shell-Sort ist gewissermaßen ein Insertion-Sort über größere Distanzen.

Schreiben sie ein Programm mit einer Methode ShellSort. Als Folge der h-Sortierung sind die Werte `9 7 4 1` fest vorzugeben.

Implementieren Sie in der Klasse `ShellSort` die Methode `sort` aus dem Interface `SortInterface` und sammeln Sie die Statistikdaten, wie oben beschrieben.