Übungsblatt 4

Hinweis:
Die Klasse `LectureWithSortAlgorithms` kann unter Programmbeispielen im Moodle-Bereich heruntergeladen werden.

Aufgabe 1 - Sortieren durch Abzählen

Erweitere die Klasse `LectureWithSortAlgorithms` um eine Methode `countAndCompareSortByName()`, die folgendermaßen sortiert:

- Es wird ein Feld `sortedStudents` mit dem Basistyp `Student` angelegt, das die Länge des Felds `students` besitzt.
- Für jedes über `students` erreichbare Student-Objekt `stud` wird die Zahl `count` derjenigen `Student`-Objekte in `students` ermittelt, deren Name kleiner als der von `stud` ist.
- Das Objekt `stud` wird im Feld `sortedStudents` an dem Index `count` abgelegt. Ist diese Position schon belegt, so treten Namen doppelt auf. Überlege, wo `stud` dann abgelegt werden kann.
- Am Ende muss das Attribut `students` auf das nun sortierte Feld `sortedStudents` gesetzt werden.

Aufgabe 2 - Sortieren durch Vertauschen

Erweitere die Klasse `LectureWithSortAlgorithms` um eine Methode `swapSortBy-Name()`, die folgendermaßen sortiert:

- Die über `students` erreichbaren `Student`-Objekte werden vom kleinsten bis zum größten Index mit ihren direkten Nachbarn solange paarweise verglichen und genau dann vertauscht, wenn der Name des Objekts an dem kleineren Index größer als der Name des Objekts an dem größeren Index ist.
- Findet sich in `students` kein Paar von direkten Nachbarn mehr, das eine Vertauschung erfordert, so ist das Feld sortiert.
- Verdeutliche Dir an einem Beispiel, warum dieses Vorgehen sortiert.
- Überlege, wie Du feststellst, dass keine weitere Vertauschung mehr notwendig ist.

Aufgabe 3 - Sortieren durch Vertauschen (Verbesserung)

Erweitere den Algorithmus aus der vorherigen Aufgabe so, dass die über `students` erreichbaren `Student`-Objekte abwechselnd vom kleinsten bis zun größten und vom größten bis zum kleinsten Index paarweise verglichen werden.
Die Regeln zum Vertauschen bleiben dabei unverändert.
Überlege, welchen Vorteil dieses Verfahren bietet?

Aufgabe 4 - Verbesserung von `insertionSort`

Erweitere die Klasse `LectureWithSortAlgorithms` um eine Abwandlung des `insertionSort Algorithmus`. Die Methode `improvedInsertionSortByName( int[] steps )` soll folgendermaßen vorgehen:

- Zunächst soll eine Methode `insertionSort( int s )` implementiert werden, die die Technik des Sortierens durch Einfügen aus `insertionSort` mit einer Schrittweite von `s` mehrfach auf das Feld `students` anwendet:
  – Der Aufruf `insertionSort( 5 )` sortiert zunächst nur die Objekte an den Indizes `0`, `5`, `10`, `15`, . . ., anschließend werden dann die Objekte mit den Indizes `1`, `6`, `11`, `16`, . . . sortiert und so fort für alle Anfangsindizes, die kleiner `5` sind. So entstehen bei diesem Beispiel also fünf sortierte Stränge innerhalb von students.
  – Der Aufruf `insertionSort( 1 )` entspricht dem aus der Vorlesung bekannten `insertionSort()`.
- Die Methode `improvedInsertionSortByName( int[] steps )` soll nun die Methode `insertionSort( int s )` nacheinander mit den im Feld `steps` gegebenen Werten aufrufen, wobei für `steps` gelten soll:
  – `steps.length > 0`
  – `steps[i] > steps[i+1]`
  – `steps[steps.length-1] == 1`
- Überlege welchen Vorteil dieses kompliziert erscheinende Vorgehen gegenüber dem in der Vorlesung vorgestellten `insertionSort()` besitzt.