Lucerne University of
Applied Sciences and Arts

HOCHSCHULE
LUZERN

Informatik

Algorithmen & Datenstrukturen (AD)

Übung: Höhere Sortieralgorithmen (A2)

Themen: Quicksort, Mergesort, Datenstruktur Heap, Heapsort, Sortieren mit der Java-Klassenbibliothek.
Zeitbedarf: ca. 300min.

Hansjörg Diethelm, Version 1.00 (FS 2017)

1 Quicksort – theoretisch durchgespielt (ca. 30')

1.1 Lernziel

- Den Sortieralgorithmus "Qicksort" verstehen.

1.2 Grundlagen

Diese Aufgabe basiert auf dem AD-Input "A21_IP_HöhereSortieralgorithmen". Die Aufgabe beinhaltet keine Programmierung.

1.3 Aufgaben

Wir betrachten folgende int-Werte bzw. zu sortierende Folge:

12 10 52_1 9 77 23 18 52_2 11 25 8 5 17

a) Sortieren Sie die Folge "auf Papier" mit dem Quicksort-Sortierverfahren. Das Element ganz rechts wählen Sie jeweils als Trennelement.

b) Bestätigt sich in diesem Beispiel die Tatsache, dass Quicksort instabil ist oder haben wir einfach Glück gehabt?

c) Sortieren Sie nochmals, wählen Sie jetzt aber das Trennelement gemäss "Median of three". Sie müssen sich nicht erneut bis zum Ende durchkämpfen!

2 Quicksort – klassisch programmiert (ca. 60')

2.1 Lernziel

- Den Sortieralgorithmus Quicksort testen.
- Die Zeitkomplexität O(n · log n) von Quicksort verifizieren.

2.2 Grundlagen

Diese Aufgabe basiert auf dem AD-Input "A21_IP_HöhereSortieralgorithmen".

2.3 Aufgaben

Wir erweitern die Bibliotheksklasse `Sort`.

a) Übernehmen Sie die Methode `quickSort(…)` zum Sortieren von Zeichen-Arrays aus dem Input. Testen Sie die Methode.

b) Implementieren Sie basierend auf a) eine Methode `quickSort(final char[] a)`, welche als Parameter einzig das Zeichen-Array benötigt. Testen Sie Ihre Methode erneut.

c) Implementieren Sie eine Methode `char[] randomChars(final int length)`, die ein Array der entsprechenden Länge und mit zufälligen Zeichen zurückliefert.

d) Sortieren Sie nun sehr grosse Arrays und messen Sie die Zeiten. Das Ziel ist, die Zeitkomplexität von Quicksort zu verifizieren.

3 Quick-Insertion-Sort (ca. 60')

3.1 Lernziel

- Den Sortieralgorithmus Quicksort optimieren.

3.2 Grundlagen

Diese Aufgabe basiert auf dem AD-Input "A21_IP_HöhereSortieralgorithmen".

3.3 Aufgaben

Wir erweitern die Bibliotheksklasse `Sort`.

a) Programmieren Sie basierend auf der Aufgabe 2 eine weitere Methode `quickInsertionSort(final char[] a)`. Diese Methode sortiert nicht bis zum bitteren Ende mit Quicksort, sondern sortiert kleine Teilfolgen mit $n < M$ mit "Insertion Sort" zu Ende. Testen Sie die Methode.

b) Sortieren Sie wiederum sehr grosse Zeichen-Arrays und versuchen mit Zeitmessungen einen sinnvollen Schwellwert für M zu bestimmen.

c) Vergleichen Sie `quickInsertionSort(…)` mit `quickSort(…)`. Bringt's die Optimierung?

4 Datenstruktur Heap (ca. 90')

4.1 Lernziel

- Die Datenstruktur "Heap" verstehen.
- Eine Heap-Datenstruktur implementieren, testen und anwenden.

4.2 Grundlagen

Diese Aufgabe basiert auf dem AD-Input "A21_IP_HöhereSortieralgorithmen".

4.3 Aufgaben

In dieser Aufgabe implementieren wir einen Heap für int-Werte. Dem Heap kann jeweils effizient mit O(log n) der grösste Wert entnommen werden.

Falls Sie möchten, dürfen Sie diese Aufgabe natürlich auch generisch umsetzen! Dann werden Sie einmal mehr auf das Interface `Comparable` bauen.

a) Zum Aufwärmen, nur "auf Papier":

Zeichnen Sie ein leeres Array mit der Länge 16 auf. Wir betrachten dieses Array als Heap. Zeichnen Sie darüber, wie im Input gezeigt, den korrespondierenden noch leeren binären Baum auf.

Fügen Sie nun der Reihe nach folgende int-Werte in den Heap bzw. in den Baum/das Array ein. Gegebenenfalls müssen Sie natürlich den Heap reorganisieren.

```
20    10    5     12    7     50
```

Entnehmen Sie nun dem Heap drei Mal den grössten int-Wert. Nach jedem Entnehmen ist in der Regel wieder ein Reorganisieren notwendig.

b) Deklarieren Sie ein sinnvolles Interface `IntegerHeap`.

c) Programmieren Sie eine Klasse `FixedSizeHeap`, welche Ihr Interface von a) implementiert. Der Heap wird mit Hilfe eines Arrays fixer Grösser realisiert, welche man beim Instanziieren angeben kann.

d) Testen Sie Ihren Heap mit Hilfe von JUnit. Teilaufgabe a) kann Ihnen dabei helfen. Denken Sie an folgende Aspekte:

- Elemente einfügen
- Elemente entnehmen
- Heap reorganisieren
- gleich grosse Elemente
- voller Heap
- leerer Heap

e) Programmieren Sie eine ganz einfache Anwender-Klasse `MyApp`, welche von Ihrem Interface `IntegerHeap` sowie von Ihrer Klasse `FixedSizeHeap` Gebrauch macht. Es geht nur darum, einen Heap zu erzeugen und auf diesem ein paar Operationen auszuführen.

f) Zeichnen Sie das UML-Klassendiagramm zu e) auf. Attribute und Operationen müssen Sie darin nicht unbedingt einzeichnen.

5 Übersicht Sortieralgorithmen (ca. 30')

5.1 Lernziel

- Behandelte Sortieralgorithmen reflektieren.
- Merkmale der behandelten Sortieralgorithmen kennen.

5.2 Grundlagen

Diese Aufgabe basiert auf den AD-Inputs "A12_IP_EinfacheSortieralgorithmen" und "A21_IP_HöhereSortieralgorithmen".

5.3 Aufgabe

a) Erstellen Sie gemäss folgender Vorlage eine Übersicht. Ihre Einträge sollten Sie nicht nur "stur" zusammentragen, sondern möglichst auch verstehen!

Eigenschaften:	Zeitkomplexität			stabil (ja/nein)	offenkundig parallelisierbar (ja/nein)	spezielle Merkmale und Hinweise
	Average Case	Worst Case	Best Case			
Algorithmus:						
Direktes Einfügen (Insertion Sort)						
Direktes Auswählen (Selection Sort)						
Direktes Austauschen (Bubble Sort)	$O(n^2)$	$O(n^2)$	$O(n^2)$	ja	nein	Nur benachbartes Vertauschen von Elementen. Option: Falls kein Vertauschen mehr stattgefunden hat, Sortieren abbrechen.
Shellsort						
Quicksort						
Mergesort						
Heapsort						

b) Auf welchen Sortieralgorithmen basieren die angesprochenen Java Sortier-Methoden? Vermerken Sie dies in Ihrer Tabelle entsprechend in der letzten Kolonne.

- `java.util.Arrays.sort(…)`
- `java.util.Collections.sort(…)`

6 Optional: Quicksort – generisch programmiert (ca. 60')

6.1 Lernziel

- Den Sortieralgorithmus Quicksort generisch programmieren.
- Anwenden des Interfaces `Comparable`.
- Umsetzen der Strategie "Median of three".

6.2 Grundlagen

Diese Aufgabe basiert auf dem AD-Input "A21_IP_HöhereSortieralgorithmen".

6.3 Aufgaben

Wir erweitern die Bibliotheksklasse `Sort`.

a) Legen Sie den Methodenkopf für `quickSort(…)` so fest, dass man mit der Methode Arrays von ganz unterschiedlichen Objekten sortieren kann, vorausgesetzt deren Klassen implementieren das Interface `Comparable`.

b) Programmieren Sie jetzt, basierend auf der Aufgabe 2, den Quicksort-Algorithmus generisch.

c) Testen Sie die Methode, indem Sie unterschiedliche Arrays sortieren, z.B. für `String` und für `Integer`.

d) Mit wenig zusätzlichen Programmzeilen können Sie noch die "Median of three" Strategie einbauen, so dass in der Regel eine unglückliche Wahl des Trennelementes vermieden werden kann.