Lucerne University of
Applied Sciences and Arts

HOCHSCHULE
LUZERN

Informatik

Algorithmen & Datenstrukturen (AD)

Übung: Grundlagen, einfache Sortieralgorithmen (A1)

Themen: Ordnung, Entscheidungsbaum, stabiles/instabiles Sortieren, Direktes Einfügen, Direktes Auswählen, Direktes Austauschen, Shellsort.
Zeitbedarf: ca. 300min.

Hansjörg Diethelm, Version 1.00 (FS 2017)

1 Grundlegende Zusammenhänge (ca. 30')

1.1 Lernziel

- Grundlegende Zusammenhänge verstehen.
- Wichtige Interfaces, Methoden und Klassen auffrischen.

1.2 Grundlagen

Diese Aufgabe basiert auf dem AD-Input "A11_IP_SortierenGrundlagen" [sowie auf dem OOP-Input "O09_IP_ObjectEqualsCompare"]. Die Aufgabe beinhaltet keine Programmierung.

1.3 Aufgaben

a) Wir gehen von einer Klasse `Triangle` aus. Jedes Dreieck-Objekt sei definiert durch seine drei Eckpunkte bzw. durch die entsprechenden (x/y)-Koordinaten. Überlegen Sie sich verschiedene "Totale Ordnungen" bzw. auf welche Arten könnte man Dreiecke in eine sortierte Folge bringen? Was schlagen Sie als "natürliche Ordnung" vor?

b) Um diese "natürliche Ordnung" umsetzen zu können, implementiert die Klasse `Triangle` welches Interface und damit welche Methode? Verstehen Sie Parameter und Rückgabewert dieser Methode?

c) Obige Methode muss den Wert `0` zurückliefern, falls zwei Objekte gleich sind. Apropos Gleichheit bedeutet dies, dass in der Regel auch `equals()` analog implementiert sein muss. Andererseits muss zudem welche andere Methode mit `equals()` korrespondieren? In beiden Fällen überschreibt man die Methoden von welcher Klasse?

d) Sie möchten neben der "natürlichen Ordnung" noch eine weitere Ordnung von a), d.h. noch eine "spezielle Ordnung" implementieren. Wie realisieren Sie dies? Zeichnen Sie ein UML-Klassendiagramm auf, welches die Teilfragen a) bis d) berücksichtigt.

e) Die Klasse `TriangleEditor` realisiere die eigentliche Anwendung. Dazu werden die Dreiecke in einem Array gespeichert. Um das Array mit den Dreiecken sortieren zu können, möchten Sie die JDK-Klasse `Arrays` verwenden. Welche Klassenmethoden darin eignen sich? Was für Sortieralgorithmen verwenden diese Methoden? Arbeiten diese stabil oder instabil?

f) Vervollständigen Sie entsprechend e) Ihr UML-Klassendiagramm von d).

2 Entscheidungsbaum (ca. 30')

2.1 Lernziel

- Die inhärente Zeitkomplexität vergleichsbasierter Sortieralgorithmen verstehen.

2.2 Grundlagen

Diese Aufgabe basiert auf dem AD-Input "A11_IP_SortierenGrundlagen". Die Aufgabe beinhaltet keine Programmierung.

2.3 Aufgaben

a) Zeichnen Sie einen vollständigen Entscheidungsbaum für $n = 3$ Datenelemente auf. [D1, D2, D3] wäre dann die sortierte Folge.

b) Für einen Binärbaum mit B Blättern gilt bekanntlich $h \geq \log_2(B)+1$. Stimmt dies für Ihren aufgezeichneten Entscheidungsbaum?

c) Wie viele Vergleiche sind also mindestens notwendig, um jeden der n! möglichen Ausgangssituationen differenzieren zu können?

3 Aus instabil mach stabil (ca. 30')

3.1 Lernziel

- Genau verstehen, was es mit stabil und instabil auf sich hat.
- Sich im algorithmischen Denken üben.

3.2 Grundlagen

Diese Aufgabe basiert auf dem AD-Input "A11_IP_SortierenGrundlagen". Die Aufgabe beinhaltet keine Programmierung.

3.3 Aufgaben

a) Angenommen, Ihnen steht ein super Sortieralgorithmus mit einer Laufzeitkomplexität von O(g(n)) zur Verfügung. Der Algorithmus ist aber leider instabil! Wie liesse sich erreichen, dass schlussendlich trotzdem ein stabiles Sortierergebnis vorliegt? Dokumentieren Sie die Modifikationen gut nachvollziehbar (vgl. Pseudocode, Beispiel, Grafik, Prosa).

b) Welche Laufzeitkomplexität hätte Ihr modifizierter Algorithmus? Machen Sie dazu Überlegungen anhand von Extremfällen. Dokumentieren Sie Ihre Findings gut nachvollziehbar.

4 Direktes Einfügen (ca. 60')

4.1 Lernziel

- Den Sortieralgorithmus "Direktes Einfügen" verstehen und optimieren.
- Eine Sortiermethode adäquat testen.
- Das Laufzeitverhalten verifizieren.

4.2 Grundlagen

Diese Aufgabe basiert auf dem AD-Input "A12_IP_EinfacheSortieralgorithmen".

4.3 Aufgaben

Wir beginnen mit dem Programmieren einer Bibliotheksklasse `Sort`. Diese Klasse soll schlussendlich verschiedene Klassenmethoden zum Sortieren von Arrays bereitstellen. Im einfachsten Fall implementieren Sie nur Methoden für das Sortieren von `int`-Arrays. Natürlich dürfen Sie auch die Herausforderung packen und generische Methoden implementieren, so dass man Arrays für beliebige Objekte sortieren kann, falls sie `Comparable` sind.

a) Implementieren Sie die Methode `insertionSort2(…)` aus dem Input. Modifizieren Sie aber die Methode geringfügig, sodass übergebene Arrays wie üblich vom Index `0` an sortiert werden.

b) Testen Sie die Methode mit adäquaten Testfällen. Sie können dies mit Hilfe von JUnit tun oder direkt in der `main(…)`-Methode mit entsprechenden Hilfsmethoden. Denken Sie bereits daran, dass es später noch weitere Sortiermethoden zu testen gilt!

c) Machen Sie ein paar Laufzeitmessungen für grosse Arrays, und zwar bei verschiedenen n und bei verschiedenen Ausgangslagen (zufällig, bereits sortiert, umgekehrt sortiert). Können Sie das erwartete Laufzeitverhalten verifizieren? Die Methode `System.currentTimeMillis(…)` ist dabei nützlich.

d) Optional: Beim "Direkten Einfügen" wird jeweils ein Element in den bereits sortierten Teil eingefügt. Beschleunigen Sie dieses Einfügen, indem Sie bei `insertionSort3(…)` die Einfügestelle binär suchen.

5 Direktes Auswählen (ca. 90')

5.1 Lernziel

- Den Sortieralgorithmus "Direktes Auswählen" verstehen, implementieren und analysieren.
- Das Laufzeitverhalten verifizieren.

5.2 Grundlagen

Diese Aufgabe basiert auf dem AD-Input "A12_IP_EinfacheSortieralgorithmen".

5.3 Aufgaben

Wir erweitern die Bibliotheksklasse `Sort`.

a) Zeigen Sie "auf Papier" an einem konkreten Beispiel gut nachvollziehbar auf, dass der Sortieralgorithmus "Direktes Auswählen" instabil ist.

b) Beim "Direkten Einfügen" haben wir bei der Analyse zwischen Best Case, Worst Case und Average Case unterschieden. Macht dieses Unterscheiden beim "Direkten Auswählen" auch Sinn? Begründen Sie gut nachvollziehbar.

c) Implementieren und testen Sie eine Methode `selectionSort(…)` gemäss Input.

d) Machen Sie ein paar Laufzeitmessungen für grosse Arrays, und zwar bei verschiedenen n und bei verschiedenen Ausgangslagen (zufällig, bereits sortiert, umgekehrt sortiert). Können Sie das erwartete Laufzeitverhalten verifizieren?

e) Analysieren Sie Ihren Programmcode. Was für eine Laufzeitkomplexität leiten Sie "mathematisch" her (vgl. Input)?

6 Direktes Austauschen (ca. 60')

6.1 Lernziel

- Den Sortieralgorithmus "Direktes Austauschen" verstehen, implementieren und optimieren.
- Das Laufzeitverhalten verifizieren.

6.2 Grundlagen

Diese Aufgabe basiert auf dem AD-Input "A12_IP_EinfacheSortieralgorithmen".

6.3 Aufgaben

Wir erweitern die Bibliotheksklasse `Sort`.

a) Implementieren und testen Sie eine Methode `bubbleSort(…)` gemäss Input.

b) Machen Sie ein paar Laufzeitmessungen für grosse Arrays, und zwar bei verschiedenen n und bei verschiedenen Ausgangslagen (zufällig, bereits sortiert, umgekehrt sortiert). Können Sie das erwartete Laufzeitverhalten verifizieren?

c) Optional: Bei "Bubble Sort" schlägt sich jeweils das grössere von zwei benachbarten Elementen nach rechts durch bzw. die beiden Elemente werden miteinander vertauscht. Mit dem Sortieren kann man aufhören, sobald bei einem Durchlauf kein Vertauschen mehr notwendig war. Implementieren und testen Sie eine entsprechende Methode `bubbleSort2(…)`.

7 Optional: Shellsort

7.1 Lernziel

- Den Sortieralgorithmus "Shellsort" verstehen und implementieren.
- Das Laufzeitverhalten untersuchen.

7.2 Grundlagen

Diese Aufgabe basiert auf dem AD-Input "A12_IP_EinfacheSortieralgorithmen".

7.3 Aufgaben

Wir erweitern die Bibliotheksklasse `Sort`.

a) Implementieren und testen Sie eine Methode `shellSort(…)` gemäss Input.

b) Machen Sie ein paar Laufzeitmessungen für grosse Arrays, und zwar bei verschiedenen n und bei verschiedenen Ausgangslagen (zufällig, bereits sortiert, umgekehrt sortiert). Wie schneidet Shellsort im Vergleich zu den direkten Sortieralgorithmen ab?