Ludwig-Maximilians-Universität München
Institut für Informatik
PD Dr. Matthias Renz
Dr. Erich Schubert, Daniel Kaltenthaler

München, 3.6.2015

Algorithmen und Datenstrukturen

SS 2015

Übungsblatt 7: Heapsort

Besprechung: 15.6.–19.6.

Abgabe aller mit **Punkten** versehenen Aufgaben bis 14.6.2015 17:35

Aufgabe 7-1 HeapSort Implementieren — 7 Punkte

Geben Sie den Java-Code als compilierbare `.java` Datei ab.

(a) Vervollständigen Sie folgende HeapSort-Implementierung:

```
public class HeapSort {
  /** Array of int sortieren mittels HeapSort */
  public static void sort(int[] daten) {
    // Heap erzeugen (Bottom-Up)
    for (int i = daten.length / 2 - 1; i >= 0; --i) {
      korrigiere(daten, i, daten.length);
    }
    // Heap auslesen bis er leer ist:
    for (int i = daten.length - 1; i > 0; --i) {
      SortUtil.vertausche(daten, 0, i);
      korrigiere(daten, 0, i);
    }
  }

  /** Heap an der Position i korrigieren */
  public static void korrigiere(int[] daten, int i, int len) {
    // TODO: Implementieren
  }
}
```

(b) Beachten Sie, dass Java-Arrays mit 0 beginnen!

(c) Compilieren und Testen Sie ihre Implementierung!
Verwenden Sie dazu auch die Datensätze aus den anderen Hausaufgaben.

Nicht compilierender Code muss von den Korrektoren mit 0 Punkten bewertet werden, und der Java-Compiler (insbesondere in ihrer IDE) *hilft* ihnen, Fehler zu finden – ebenso der Test.

Aufgabe 7-2 Binäre Suchbäume — 3 Punkte

In einem binären Suchbaum sind Zahlen zwischen 1 und 1000 gespeichert. Geben Sie an, ob die folgenden Sequenzen auf der Suche nach der Zahl 333 durchlaufen worden sein können. Begründen Sie Ihre Entscheidung.

(a) 788, 398, 195, 307, 23, 353, 320, 333

(b) 283, 571, 312, 451, 437, 344, 314, 333

(c) 795, 351, 552, 113, 203, 289, 299, 333

(d) 151, 820, 813, 277, 367, 304, 350, 333

Aufgabe 7-3 HeapSort Hausaufgabe **9 Punkte**

Sortieren Sie das folgende Array mit dem HeapSort-Verfahren *exakt wie es in der Vorlesung besprochen wurde*.

Geben Sie dazu eine Tabelle ab, mit:

- 1. Spalte: Interne Repräsentation als Array (inkl. serialisiertem Heap und bereits sortierten Daten)
- 2. Spalte: Interpretation als Baumstruktur (graphisch, ohne bereits sortiere Daten)

In die erste Zeile tragen Sie die initialen Daten ein (inkl. graphischer Interpretation als Baum!)

Nun erzeugen Sie den Heap, indem Sie ihn von unten nach oben korrigieren. Geben Sie die Daten in einer neuen Zeile immer dann an, *nachdem* ein Level *fertig korrigiert* ist. Kennzeichnen Sie in beiden Repräsentationen, welcher Teil des Heaps noch nicht korrigiert wurde. Alle Korrekturen im gleichen Level können Sie zusammen in einem Schritt durchführen, da diese voneinander unabhängig sind.

Sie sollten jetzt 4 Zeilen gefüllt haben. Kontrollieren Sie, ob Sie einen korrekten Heap haben!
Wenn Sie mit einem fehlerhaften Heap fortfahren, wird der Rest der Aufgabe mit 0 Punkten bewertet!

Anschließend führen Sie die zweite Phase des HeapSort durch, und zeichnen den Heap *nach* jedem Sortierschritt (d.h. auch *nach* dem Korrigieren des Heaps). Beachten Sie dass nach jedem Schritt ein korrekter Heap vorhanden sein sollte, wenn Sie keinen Fehler gemacht haben.
Wenn Sie mit einem fehlerhaften Heap fortfahren, wird der Rest der Aufgabe mit 0 Punkten bewertet!

Zu sortieren ist die folgende Liste von Zahlen: 15, 13, 45, 33, 1, 82, 64, 5, 83, 78

Aufgabe 7-4 Eigenschaften allgemeiner Bäume **Tutoraufgabe**

Wir betrachten einen Baum vom Grad $d \geq 2$.

(a) Wie groß ist die maximale Anzahl von Knoten auf Level i eines Baumes vom Grad d?

(b) Wie groß ist die maximale Anzahl von Knoten in einem Baum der Höhe h vom Grad d?

(c) Wie groß ist die maximale Höhe eines Baumes vom Grad d mit n Knoten?

(d) Wie groß ist die minimale Höhe eines Baumes vom Grad d mit n Knoten?

Begründen Sie Ihre Aussagen.

Aufgabe 7-5 Binäre Suchbäume 2 **Tutoraufgabe**

Gegeben sind n Zahlen. Geben Sie mit eigenen Worten ein Verfahren an, um einen binären Suchbaum

(a) der größtmöglichen Tiefe zu erzeugen.

(b) der kleinstmöglichen Tiefe zu erzeugen.

Aufgabe 7-6 Binäre Suche **Tutoraufgabe**

Mit dem Algorithmus "binäre Suche" kann ein *sortieres Array* effizient durchsucht werden.
Analysieren Sie folgende Implementierung:

```
public class BinaereSuche {
  /** Binaere Suche in einem Array
   *  @return Position des Elementes, oder -1. */
  public static int binaereSuche(int[] daten, int key) {
    return binaereSuche(daten, key, 0, daten.length);
  }

  /** Rekursive Implementierung
   *  @return Position des Elementes, oder -1. */
  public static int binaereSuche(int[] daten, int key, int l, int r) {
    if (r <= l) { // Such-Intervall ist leer
      return -1;  // Nicht gefunden
    }
    final int m = (l + r) / 2; // Besser: (l + r) >>> 1
    if (key < daten[m]) {
      return binaereSuche(daten, key, l, m);
    }
    else if (key > daten[m]) {
      return binaereSuche(daten, key, m + 1, r);
    }
    return m; // Position zurueckliefern
  }

  /** Einfache Testfunktion, besser waere ein JUnit-Test */
  public static void main(String[] args) {
    int[] daten = new int[]{ 2, 3, 5, 7, 11, 13, 17, 19, 23, 29 };
    System.out.println("Position(1)  = "+binaereSuche(daten, 1) +" soll: -1");
    System.out.println("Position(3)  = "+binaereSuche(daten, 3) +"  soll: 1");
    System.out.println("Position(8)  = "+binaereSuche(daten, 8) +" soll: -1");
    System.out.println("Position(23) = "+binaereSuche(daten, 23)+"  soll: 8");
  }
}
```

- Welche Komplexität hat dieser Algorithmus in O-Notation?
- Dieser Algorithmus interpretiert das Array als binären Suchbaum.
  Zeichnen Sie den entsprechenden Suchbaum für das Array $[2, 3, 5, 7, 11, 13, 17, 19, 23, 29]$.
- Optionale Übungsaufgabe: Implementieren Sie die binäre Suche *ohne* Rekursion.