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

München, 20.5.2015

Algorithmen und Datenstrukturen

SS 2015

Übungsblatt 6: Sortieralgorithmen III

Besprechung: 8.6.–12.6.

Abgabe aller mit **Punkten** versehenen Aufgaben bis 7.6.2015 17:35

Aufgabe 6-1 Stabilität von QuickSort — 2 Punkte

Der Sortieralgorithmus QuickSort ist nicht stabil.
(In der Praxis sind QuickSort-Varianten aber beliebt wenn Stabilität nicht notwendig ist!)

Demonstrieren Sie dies an einem Gegenbeispiel.

Hinweis: Sie sollten nur 4 Elemente benötigen, davon einen Schlüssel doppelt.
Beispielsweise $11_A \leq 11_B \wedge 11_B \leq 11_A$. Verwenden Sie den Algorithmus aus der Vorlesung.

Aufgabe 6-2 QuickSort Implementieren — 8 Punkte

Geben Sie den Java-Code als compilierbare `.java` Datei ab. Frage (b) beantworten Sie in einer `.pdf` Datei!

(a) Vervollständigen Sie folgende QuickSort-Implementierung:

```java
public class QuickSort {
  /** Array of int sortieren mittels QuickSort */
  public static void sort(int[] daten) {
    quicksort(daten, 0, daten.length - 1);
  }

  /** Teilarray of int sortieren mittels QuickSort */
  public static void quicksort(int[] daten, int start, int ende) {
    if (start >= ende) { return; } // Max 1 Element: fertig!
    // Divide at Pivot:
    final int pivot = daten[start]; // Pivot
    int links = start + 1, rechts = ende;
    while(true) {
      // TODO: Implementieren
      if (links < rechts) {
        // TODO: Implementieren
      } else {
        break;
      }
    }
    // Pivot an der RICHTIGEN Position einsetzen:
    SortUtil.vertausche(daten, /* TODO: Implementieren */);
    // Conquer (kein Merge notwendig, wenn Pivot korrekt):
    quicksort(daten, start, rechts - 1);
    quicksort(daten, rechts + 1, ende);
  }
}
```

(b) Im Vorlesungsskript sind die Variablen anders benannt. Geben Sie an, welche Variablen in diesem Code welchen Variablen im Vorlesungsskript entsprechen. (Als `.pdf` abgeben!)

(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 6-3 QuickSort II **6 Punkte**

Sortieren Sie das folgende Array mit dem QuickSort-Verfahren *exakt wie es in der Vorlesung besprochen wurde*: Wählen Sie stets das erste Element als Pivot-Element.
Verwenden Sie eine Tabelle mit der folgenden Struktur:

Anfang:	81	86	94	71	89	65	51	59	77	49	24	3

Vor Pivot:												
Nach Pivot:												
Vor Pivot:												
Nach Pivot:												
Vor Pivot:												
Nach Pivot:												
Vor Pivot:												
Nach Pivot:												
Vor Pivot:												
Nach Pivot:												
Vor Pivot:												
Nach Pivot:												
Vor Pivot:												
Nach Pivot:												
Vor Pivot:												
Nach Pivot:												

⋮

Markieren Sie für jeden rekursiven Durchlauf des Verfahrens das betroffene Intervall *vor und nach* nach dem Verschieben des Pivot-Elements. Es reicht, wenn Sie das Pivot-Element und diejenigen Elemente eintragen, deren Position sich ändert (um die Schreibarbeit zu reduzieren)!

Streichen Sie *nie* richtige Elemente durch: Durchgestrichenes können wir nicht bewerten, die Lösung muss für uns eindeutig erkennbar sein! Eigene Markierungen (Zeiger, Intervalle, Vertauschungspfeile etc.) können Sie zwischen den Zeilen eintragen. Diese Tabellenform ist darauf ausgelegt, dass Sie nichts durchstreichen müssen.

Aufgabe 6-4 HeapSort Arraydarstellung **Tutoraufgabe**

HeapSort arbeitet *logisch* mit einer Baumdarstellung ("Heap"),
aber *technisch* bleiben die Daten die gesamte Zeit in einem Array gespeichert ("linear").

(a) Machen Sie sich den Zusammenhang zwischen Array-Speicherung und Baum-Interpretation klar. Wie hängen diese beiden Darstellungen zusammen?

(b) Warum wird der Heap derart als Array gespeichert, und nicht z.B. mit `Knoten`-Objekten wie wir sie auf Übungsblatt 4 verwendet haben?

(c) In der Vorlesung wurde die Datenstruktur vorwiegend mathematisch besprochen (d.h. mit $i \in \{1, \ldots, n\}$ indiziert). Nahezu alle Programmiersprachen verwenden jedoch eine mit 0 beginnende indizierung.

Gegeben die Position $p \in \{0, \ldots, n-1\}$ des Vaterknotens, wie berechnen Sie die Position p_l des linken Kindes und die Position p_r des rechten Kindes im Array? Wie berechnet sich die Position des Vaters p_v?

Wie können Sie prüfen, ob das Kind an Position p_l existiert (oder ob die Position leer ist)?

(d) HeapSort bringt zunächst das größte Element an die erste Position, wenn man aufsteigend sortieren will. Warum? Was bedeutet dies für den Fall, dass man bereits richtig (oder fast-richtig) sortierte Daten hat?

Aufgabe 6-5 HeapSort **Tutoraufgabe**

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: 18, 98, 21, 27, 87, 92, 44, 28, 38, 24