Datenstrukturen und effiziente Algorithmen I

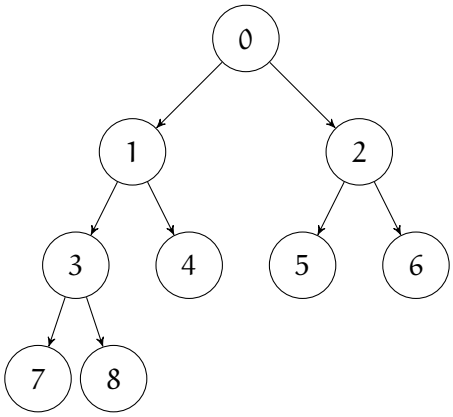
Sommersemester 2023
Martin-Luther-Universität Halle-Wittenberg
Institut für Informatik
Lehrstuhl für Datenstrukturen und effiziente Algorithmen

Übungsblatt 9

Abgabe: 13. Juni 2023

Aufgabe 1: Fast-vollständige Binäre Bäume (3+2 Punkte)

In der Vorlesung wurde gezeigt, wie die n Knoten eines fast-vollständigen binären Baumes auf ein Feld der Länge n abgebildet werden können. Dabei wurden die Knoten des Baumes, bei 0 beginnend, schichtweise von links nach rechts durchnummeriert (siehe Beispiel).



Beweisen Sie die folgenden Sätze im Zusammenhang mit fast-vollständigen Binärbäumen:

(a) Wenn i der Index eines Knotens ist, so ist $\mathit{left}(i) = 2i + 1$ der Index des linken Kindes und $\mathit{right}(i) = 2i + 2$ der Index des rechten Kindes (sofern diese existieren).

(b) Nur Knoten mit einem Index i im Bereich $0 \leq i \leq \lfloor n/2 \rfloor - 1$ haben Kinder.

Aufgabe 2: HEAP-SORT (4 Punkte)

Sortieren Sie das Feld $A = [7, 8, 10, 2, 7, 5, 11]$ manuell mittels HEAP-SORT (siehe Folie 235). Zeichnen Sie das Feld und den dazugehörigen Baum nach jedem Vertauschen zweier Elemente. Markieren Sie jeweils die getauschten Elemente und zusätzlich bei der Sortierung den Bereich, der von HEAPIFY bearbeitet wird.

Aufgabe 3: HEAPSORT: Implementierung (3+1+3 Punkte)

(a) Implementieren Sie den Sortieralgorithmus HEAP-SORT (siehe Folie 235) zum Sortieren ganzer Zahlen. Schreiben Sie dazu eine Klasse `HeapSort` mit der Methode

```
public class HeapSort {
  public static void sort(int[] A) { ... }
}
```

(b) Erweitern Sie Ihre Klasse `HeapSort` um eine `main`-Methode, welche eine Datei mit ganzen Zahlen einliest, diese mittels Ihrer `sort`-Methode sortiert, und in sortierter Reihenfolge wieder zeilenweise ausgibt. Ihr Programm soll von der Kommandozeile mit der folgenden Syntax aufgerufen werden:

```
java HeapSort <filename>
```

wobei `<filename>` den Pfad zu einer Textdatei bezeichnet, in welcher zeilenweise ganze Zahlen gespeichert sind. Testen Sie Ihre Implementierung!

(c) Messen Sie die Laufzeit Ihrer Implementierung für verschiedene, möglichst große Instanzen und stellen Sie diese grafisch dar. Vergleichen Sie die Laufzeit anschließend mit der von MERGE-SORT. Welcher Sortieralgorithmus schlägt sich besser?

Aufgabe 4: HEAPSORT: Best-Case- und Worst-Case-Instanz (4 Punkte)

Überlegen Sie sich zwei Folgen mit je 8 Zahlen $(x_1, x_2, \ldots, x_8), x_i \in \mathbb{N}$, für die HEAPSORT die wenigsten bzw. die meisten Tauschoperationen benötigt. Geben Sie ihre Zahlenfolgen an und schildern Sie nachvollziehbar, warum die Anzahl der Tauschoperationen bei diesen Folgen besonders gering bzw. hoch ist.
Hierbei sind die Tauschoperationen beim Aufbau des Heaps und bei der Sortierung zu berücksichtigen.