Algorithmen und Datenstrukturen INF3/ICS3

Wintersemester 2023/24

Prof. Dr. Georg Schied

Lösungen - Aufgabenblatt 4

Abgabetermin: Mo. 13. November 2023, 23:59 Uhr

Zum Bestehen müssen 10 von 20 Punkten erreicht werden.

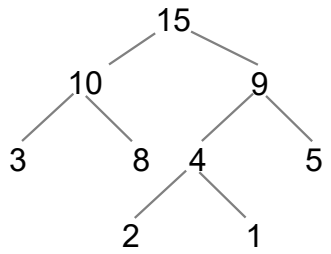
Aufgabe 4.1

Angenommen, die binäre Suche benötigt 69 μs, um in einem Feld von n = 1000 Elementen zu suchen und 92 μs bei n = 10 000 Elementen. Welche Laufzeit ist für n = 100 000 und n = 1 000 000 zu erwarten? Schätzen Sie die Laufzeiten ab ohne einen Taschenrechner oder andere Hilfsmittel zu verwenden!

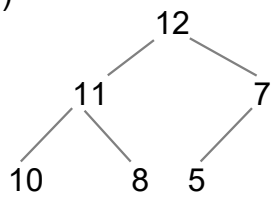
Aufgabe 4.2

a) Welche der folgenden Bäume sind binäre Maximum-Heaps?

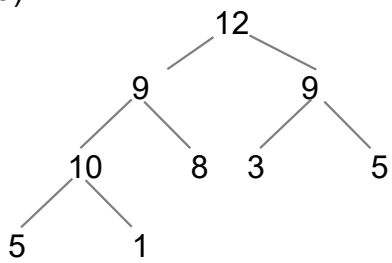
(1)



(2)



(3)

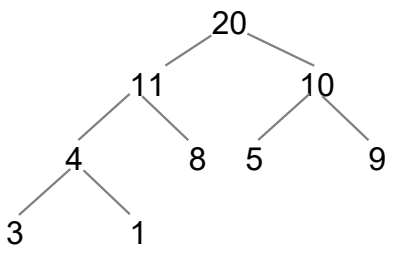


b) Welcher binäre Heap, als Baum dargestellt, wird durch folgendes Array repräsentiert?

10	9	8	3	1	2	9	2

Aufgabe 4.3 - Scheinaufgabe (4 Punkte)

a) Wie würde der folgende Maximum-Heap in einem Array abgespeichert?



b) Welche der folgenden Arrays repräsentieren einen Maximum-Heap? (jeweils mit kurzer Begründung)

(1)

12	6	15	5	9	13

(2)

11	7	10	8	5	6

(3)

12	9	10	2	1	7	9

Aufgabe 4.4

Gegeben ist folgender Array-Inhalt. Zeigen Sie, wie daraus ein binärer Maximum-Heap gebildet wird.

2	8	6	1	9	5	4	3

Aufgabe 4.5 - Scheinaufgabe (8 Punkte)

Zeigen Sie, wie **Heapsort** ein Array mit folgendem Inhalt sortiert:

7	2	4	3	9	1	6	5

Geben Sie die wesentlichen Zwischenschritte als Baum und Array an und kommentieren Sie jeweils kurz die Vorgehensweise, so dass der Ablauf nachvollziehbar ist.

Aufgabe 4.6 - Scheinaufgabe (8 P)

Die Klasse `TaskList` (siehe Moodle) mit den Operationen

```
public TaskList(int capacity)
public void add(Task task)            // adds task to end of list
public int size()                     // number of tasks in the list
public Priority getPriority(int i)    //priority of task at  pos. i
public void swap(int i, int j)        // swaps tasks at positions i and j
public boolean isOrdered()            // are tasks ordered by priority?
```

beschreibt eine Liste von Aufgaben, wobei jede Aufgabe (Task) eine Bezeichnung und eine Priorität HIGH, MEDIUM oder LOW hat.

a) Schreiben Sie in Klasse `TaskDemo` eine möglichst effiziente Methode

```
public static void reorderTasks(TaskList list)
```

die die Tasks nach absteigender Priorität sortiert. Durch Vertauschen sollen alle Tasks innerhalb der List so umordnet werden, dass die Tasks mit Priorität HIGH am Anfang kommen, danach Tasks mit Priorität MEDIUM und am Ende Tasks mit Priorität LOW. Verwenden Sie zum Umordnen die Operationen `getPriority(i)` und `swap(i,j)`, wobei die Positionen ab 0 gezählt werden. Die Klasse `TaskList` darf nicht geändert werden!

Mit der Methode `isOrdered()` können Sie prüfen, ob die Tasks in der Reihenfolge HIGH-MEDIUM-LOW richtig angeordnet sind.

b) Messen Sie die Laufzeit für das Umordnen mit verschiedenen Größen n = 100, 1 000, ..., 10 000 000. In Klasse `TaskDemo` finden Sie eine Messmethode, um eine Liste von Tasks der angegebenen Größe mit zufällig gewählten Prioritäten zu erzeugen und die Laufzeit für das Umordnen zu messen. Welches Laufzeitverhalten ist erkennbar? Schafft es Ihre Implementierung, eine zufällig aufgebaute Reihe von *n* = 10 000 000 Tasks in weniger als 5 Sekunden richtig umzuordnen (gestartet mit Option `-Xint`)?

Tipp: Überlegen Sie sich, ob es einfachere Möglichkeiten als die Verwendung eines gängigen Sortierverfahrens gibt.