Algorithmen und Datenstrukturen

Wintersemester 20/21

Prof. Dr. Georg Schied

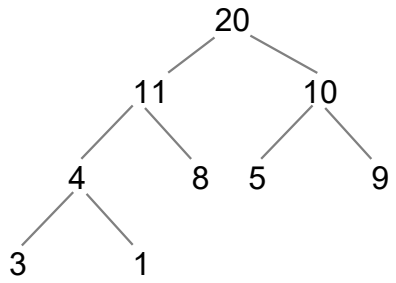
Aufgabenblatt 4

Abgabetermin: Do. 5. November 2020, 23:59 Uhr

Zum Bestehen müssen 9 von 18 Punkten erreicht werden.

Aufgabe 4.1

a) Wie würde der rechts angegebene Maximum-Heap in einem Feld abgespeichert?



b) Welche der in den folgenden Feldern gespeicherten Werte bilden einen Maximum-Heap? (kurze Begründung)

15	8	17	5	9	13

20	5	10	3	1	8	9

10	7	9	8	5	1

Aufgabe 4.2 - Scheinaufgabe (6 Punkte)

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

2	8	4	3	9	1

Geben Sie die wesentlichen Zwischenschritte als Baum und Array an und kommentieren Sie kurz die Vorgehensweise.

Aufgabe 4.3 - Scheinaufgabe (6 Punkte)

Gegeben ist ein Array mit den Werten

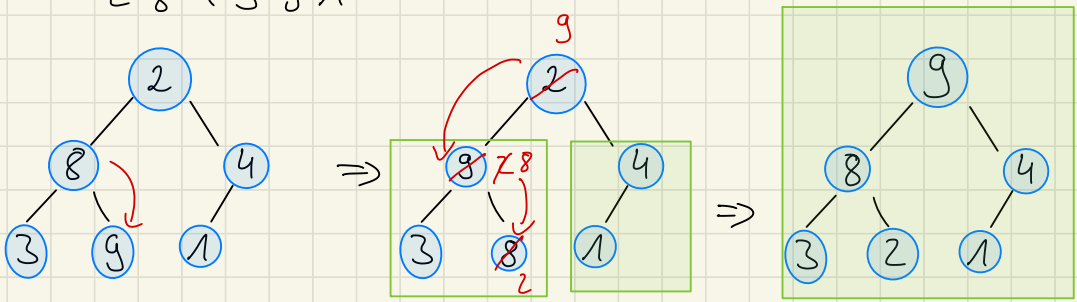
5, 9, 4, 7, 3, 6, 8.

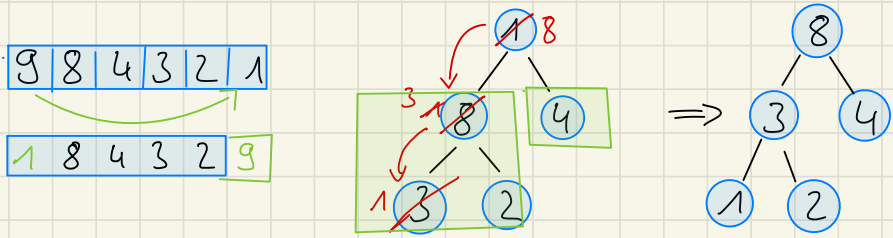
a) Stellen Sie dar, wie die Daten mit **Quicksort** sortiert werden. Das Aufteilen muss *nicht exakt* nach der in der Vorlesung vorgestellten Methode vorgenommen werden. Es soll aber erkennbar sein, nach welcher Regel das Pivotelement gewählt wurde.

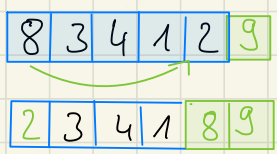
b) Stellen Sie dar, wie die Daten mit **Mergesort** sortiert werden.

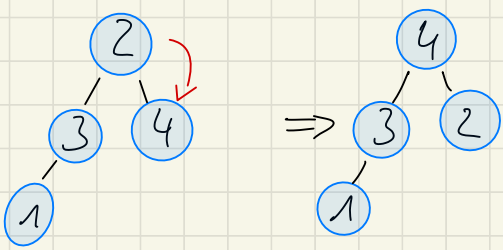
Aufgabe 4.2.

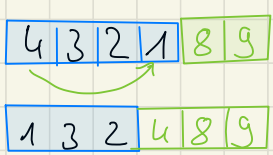
2 8 4 3 9 1

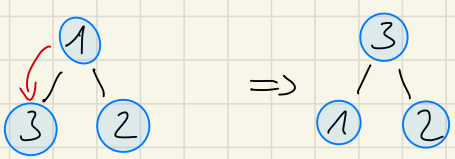


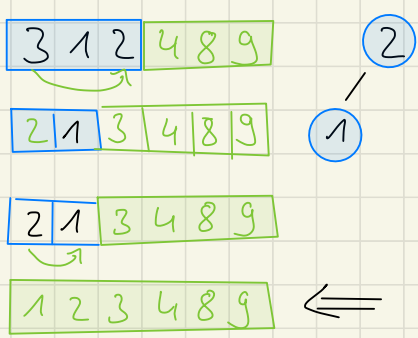






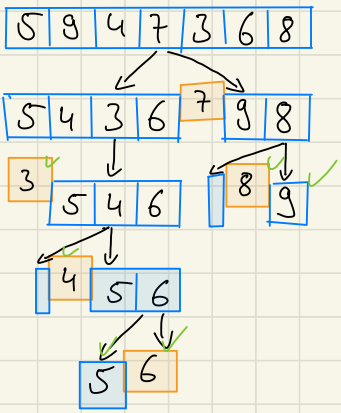






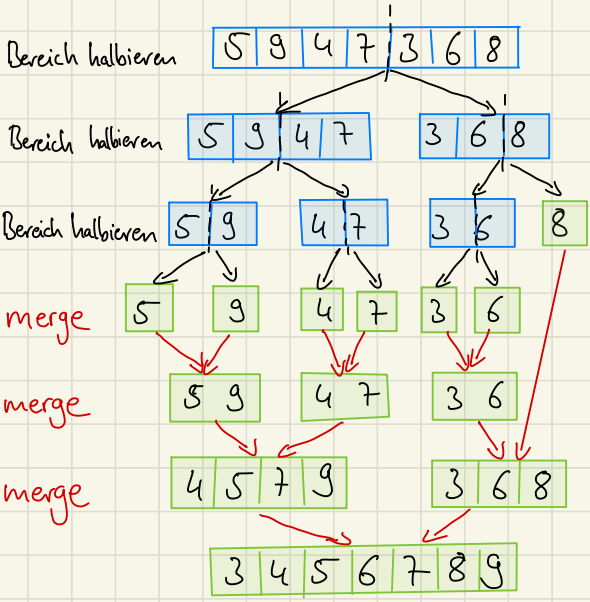
Aufgabe 4.3

a.)



Als Array: 3 4 5 6 7 8 9

b.)



Geben Sie jeweils die wesentlichen Zwischenschritte an, so dass das Teile-und-Herrsche-Prinzip erkennbar ist.

Aufgabe 4.4 - Scheinaufgabe (6 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.