Universität Konstanz
Fachbereich Informatik & Informationswissenschaft
Prof. Dr. Ulrik Brandes / Dr. Andreas Karrenbauer / Dr. Jürgen Lerner / Martin Mader

Algorithmen und Datenstrukturen
WS 2010/2011

3. Übungsblatt

Ausgabe: 03. November 2010 **Abgabe:** 10. November 2010, 10 Uhr
Die Bearbeitung in Zweiergruppen ist ausdrücklich erwünscht.

Aufgabe 1: MergeSort – Varianten — 5 Punkte

Im Folgenden werden drei Varianten von MergeSort vorgestellt, die eine unterschiedliche Strategie für die Aufteilung in Teilfolgen benutzen. Geben Sie für jede der Varianten die asymptotische Laufzeit in der Θ-Notation an; begründen Sie jeweils ihre Entscheidung.

(Wir vernachlässigen, dass die Längen der Teilfolgen in einigen Fällen auf ganze Zahlen gerundet werden müssen; dies hat keine Auswirkung auf die asymptotischen Laufzeiten. Genauso vernachlässigen wir die Abbruchbedingungen für kleine n.)

(a) Eine Folge der Länge n wird in zwei Teilfolgen aufgeteilt von denen eine die Länge $2n/3$ und die andere die Länge $n/3$ hat.

(b) Eine Folge der Länge n wird in zwei Teilfolgen aufgeteilt von denen eine die Länge $n - 100$ und die andere die Länge 100 hat.

(c) Eine Folge der Länge n wird in drei Teilfolgen aufgeteilt von denen jede die Länge $n/3$ hat.

Aufgabe 2: Heaps — 3 Punkte

(a) Realisiert folgendes Array einen Heap?

$$A = [23, 17, 14, 6, 13, 10, 1, 5, 7, 12]$$

(b) Führen Sie auf einem anfangs leeren (Max)Heap die folgenden Operationen aus. Geben Sie den Zustand des Heaps nach jeder Operation an (als Array oder als Baum).

`insert(5)`, `insert(1)`, `insert(8)`, `insert(9)`, `insert(3)`, `extractMax()`, `extractMax()`, `insert(7)`, `insert(2)`, `extractMax()`.

Aufgabe 3: Heaps **10 Punkte**

Realisieren Sie einen (Max-)Heap in Java, der analog zur Vorlesung intern ein Array verwendet. Ihre Implementation soll generisch sein, d.h. mit parameterisiertem Typ `<T extends Comparable<T>>` für die Heap-Elemente.

(a) Implementieren Sie das Interface `material.u03.IMaxHeap` in einer Klasse `u03.<name1_name2>.MaxHeap`. Überschreiben Sie die `toString()`-Methode der Klasse, so dass das Teilarray, das tatsächlich Elemente enthält, mittels der `toString()`-Methode aus `java.util.Arrays` zurückgegeben wird. (5 Punkte)

Sie können zunächst davon ausgehen, dass die Größe des internen Arrays fest ist (z.B. 100), und dass Vergleiche zwischen den Elementen auf der `compareTo`-Methode des Typs `T` basieren, oder aber direkt Teilaufgaben (b) und (c) berücksichtigen.

(b) *Dynamische Arrays:* Ändern Sie die Klasse so ab, dass das interne Array je nach Bedarf durch ein Größeres bzw. Kleineres ersetzt wird. (2 Punkte)

- Fügen Sie einen Konstruktor `MaxHeap(int capacity)` hinzu, so dass eine Initial-Kapazität für das intern verwendete Array angegeben werden kann.
- Falls das interne Array bei einem Aufruf von `insert(T t)` zu klein wird, soll ein Array mit doppelter Kapazität verwendet werden.
- Falls bei einem Aufruf von `extractMax()` nur noch ein Viertel des internen Arrays benutzt wird, soll die Größe des Arrays halbiert werden

(c) *Komparatoren:* Ändern Sie die Klasse so ab, dass für den Vergleich der Elemente im Heap ein `java.util.Comparator<T>` verwendet wird. (3 Punkte)

- Fügen Sie einen Konstruktor `MaxHeap(Comparator<T> comparator)` hinzu, dem ein Komparator für den Vergleich der Elemente im Heap übergeben wird. Falls kein Komparator im Konstruktor übergeben wird, soll ein Komparator verwendet werden, der die natürliche Ordnung des Typs `T` berücksichtigt.
- Testen Sie die Klasse, in dem Sie `Double`-Zahlen nicht bezüglich ihrer natürlichen Ordnung, sondern ihres Absolutbetrags verglichen werden. Implementieren Sie dazu eine Klasse `u03.<name1_name2>.DoubleAbsComparator`, der das Interface `Comparator<Double>` implementiert.

- Implementieren und verwenden Sie einen möglichst allgemeinen `u03.<name1_name2>.InverseComparator<T extends Comparable<T>>`, der das Interface `Comparator<T>` implementiert. Mit Diesem soll man aus Ihrem Max-Heap einen Min-Heap machen können, ohne den Code von `MaxHeap` zu ändern.

Hinweise:

- Wegen des generischen Typs des internen Arrays können Sie dieses nicht im Konstruktor initialisieren, sondern erst beim ersten Aufruf von `insert(T t)`. Sie können dazu die Methode `java.lang.reflect.Array.newInstance(t.getClass(), length)` verwenden.
- Es darf das Paket `java.util.Arrays` verwendet werden.

Aufgabe 4: Dynamische Arrays **2 Punkte**

Betrachten Sie die Vergrößerungsstrategie aus Aufgabe 3(b). Nehmen Sie zur Vereinfachung an, dass neue Elemente ans Ende des Arrays geschrieben werden, ohne die Heap-Bedingung wiederherzustellen.

Geben Sie die kleinste obere asymptotische Schranke (in der $\mathcal{O}$-Notation) für die worst-case Laufzeit für das Einfügen von n Elementen in ein anfangs leeres Array der Größe 1 an. Begründen Sie Ihre Entscheidung.