Übung 11: Sortieren

Abgabetermin: 19.06.2018

Name: **Matrikelnummer:**

Gruppe: ☐ **G1** Di 10:15-11:00 ~~☐ **G2** Di 11:00-11:45~~ ☐ **G3** Di 10:15-11:00

Aufgabe	Punkte	gelöst	abzugeben schriftlich	abzugeben elektronisch	Korr.	Punkte
Aufgabe 1	8	☐	Java-Programm	Projekt Archiv	☐	
Aufgabe 2	8	☐	Java-Programm	Projekt Archiv	☐	
Aufgabe 3	8	☐	Java-Programm	Projekt Archiv	☐	

Aufgabe 1: MergeSort (8 Punkte)

Implementieren Sie einen *Merge Sort* in der Klasse ***MergeSorter***. Gehen Sie in Ihrem Algorithmus wie in der Vorlesung besprochen vor und teilen Sie das Input Array in zwei Hälften auf. Auf jeder dieser Hälften rufen Sie wieder ***sort(...)*** auf, solange bis das geteilte Array die Länge 1 hat. Danach fügen Sie in einer eigenen Methode ***mergeSorted(Object[] a, Object[] b, Comparator cmp, SortingOrder ordering)*** beide Arrays, unter Einhaltung einer korrekten Sortierung, wieder zu einem neuen Array zusammen.

Abzugeben ist: Projekt Archiv

Aufgabe 2: HeapSort (8 Punkte)

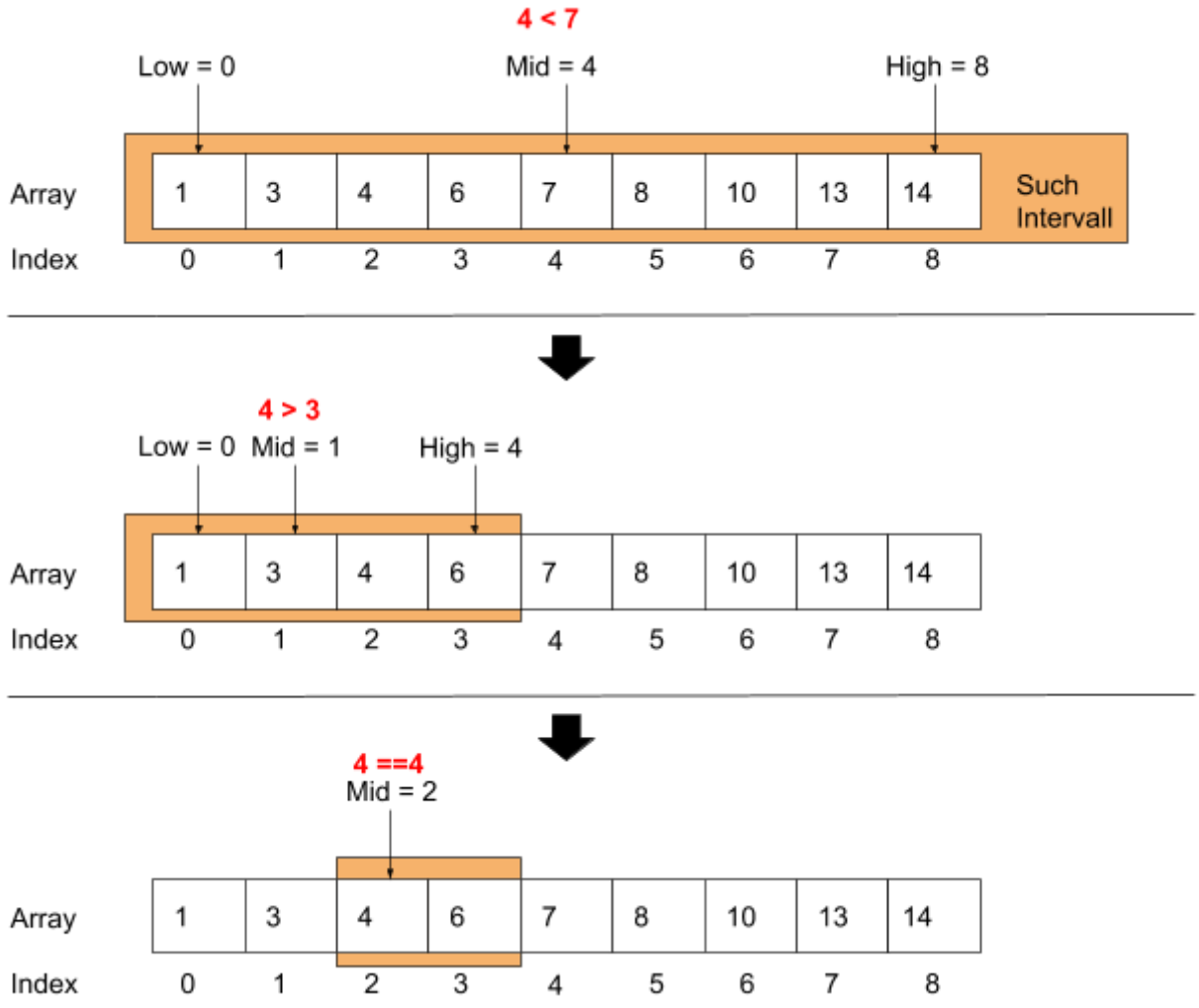
Implementieren Sie eine einfache Variante des *Heap Sort Algorithmus* zum sortieren beliebiger Object Arrays in der Klasse ***HeapSorter***. Sie können die Klasse ***ArrayPriorityQueue,*** die einen Heap implementiert, benutzen. Dafür fügen Sie die Elemente des Arrays zuvor in den Heap ein (unter zuhilfenahme des Comparators der als Parameter übergeben wird) und befüllen danach das Input Array neu.

Abzugeben ist: Projekt Archiv

Aufgabe 3: Binäre Suche (8 Punkte)

Implementieren Sie eine Binärsuche in der Klasse ***BinarySearch***. Die binäre Suche ist ein einfacher aber sehr effizienter Suchalgorithmus zum Suchen in **sortierten** Arrays. Die Idee basiert auf dem **Divide and Conquer**[1] Ansatz. Das Suchproblem wird dabei rekursiv so lange in kleinere Probleme zerteilt, bis eine triviale Lösung gefunden werden kann. BinarySearch teilt das zu durchsende (Sub-)Array immer in 3 Teile auf. Die **Mitte,** links von der Mitte und rechts davon. Zunächst wird das zu suchende Element mit der Mitte des Arrays verglichen. Ist das gesuchte Element kleiner, wird eine neue Suche in der linken Hälfte des Arrays durchgeführt, ansonsten in der rechten Hälfte. Dieses Verfahren wird so lange durchgeführt bis das zu suchende Element gefunden wurde oder das Suchintervall (Sub-Array) leer ist.

Zur Veranschaulichung soll folgendes Beispiel dienen. Wir suchen in einem Array mit 9 Elementen den Index des Wertes **4.** Dafür vergleichen wir 4 zuerst mit der MItte (die Mitte berechnet sich immer als untere Schranke + obere Schranke / 2). 4 ist kleiner 7 somit haben wir ein neues Suchintervall nämlich zwischen Index 0 und Index 4. Wir vergleichen 4 erneut mit der Mitte usw. bis der Wert 4 gefunden ist.



Abzugeben ist: Projekt Archiv

[1] https://de.wikipedia.org/wiki/Teile_und_herrsche_(Informatik)

Implementierungshinweise:

- Verwenden Sie das Vorgabeprojekt **PI2_UE11.zip**.
- Alle Algorithmen die von der Klasse ***Sort*** erben bekommen einen Parameter ***SortingOrder ordering*** der angibt ob das Input Array aufsteigend (***SortingOrder.ASCENDING)*** oder absteigend (***SortingOrder.DECENDING***) sortiert werden soll.
- Sie können die Klasse ***SortSearchMain*** verwenden um die Funktionalität ihrer Implementierungen zu Testen.
- Sie können das Eingabe-Array "***in-place"*** sortieren, also die Elemente des Arrays überschreiben ohne ein neuen Array anzulegen. Der Rückgabewert der Methoden ist stets ein sortiertes Array, egal ob es sich um das Eingabe- oder ein neues Array handelt.
- Es existieren verschiedenste Ressourcen zur Visualisierung von Sortierverfahren, z.B.: https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html.
- Ändern sie **keine public Interfaces** vorgegebener Skeleton Klassen.
- Halten Sie sich an die **Codierungsrichtlinien** auf der Kurs Website.