**GTU Department of Computer Engineering**

**CSE 222/505 - Spring 2021**

**Homework #4 Report**

**Gökbey Gazi Keskin**

**1901042631**

1. **Introduction**
   1. **Problem Definition**

First need of the homework is a Heap class which can:

i. Search for an element

ii. Merge with another heap

iii. Removing ith largest element from the Heap

iv. Extend the Iterator class by adding a method to set the value (value passed as parameter) of the last element returned by the next methods.

Then using this Heap class and a Binary Search Tree class, It is expected to implement a hybrid container of Nodes which holds a generic data and occurrence amount of this data. Required container is a Binary Search Tree which holds a Heap which holds at max 7 nodes at it’s data field. Also, the maximum occurring element should be kept in data fields.

* 1. **Detailed System Requirements**

**1.2.a) Requirements of the MyHeap class**

First of all, a comparable max heap(descending-ordered heap) has to be created. The reason why it should be comparable is, the binary search tree in Part 2 should be able to sort these heaps.



**** Heap should be non-empty for operations peek poll, search and remove (otherwise they return null). The element must be specified for removing and adding elements.



Index has to be specified for remove Ith element operation.



The other Heap to be merged into first Heap should be specified for merge operation.



**System Requirements of private inner class HeapIterator**

First, a HeapIterator must be created.

The presence of the next element must be checked using hasNext() method every time before iterating with next() method.

****

The element must be specified for the set operation



**1.2.b) Requirements of the BSTHeapTree class**

First, a BSTHeapTree must be created.

Element should be specified for add, remove and find operations.

To insert more then one element, additionally amount should be specified

**System Requirements of private inner class Node**

First, a node must be created

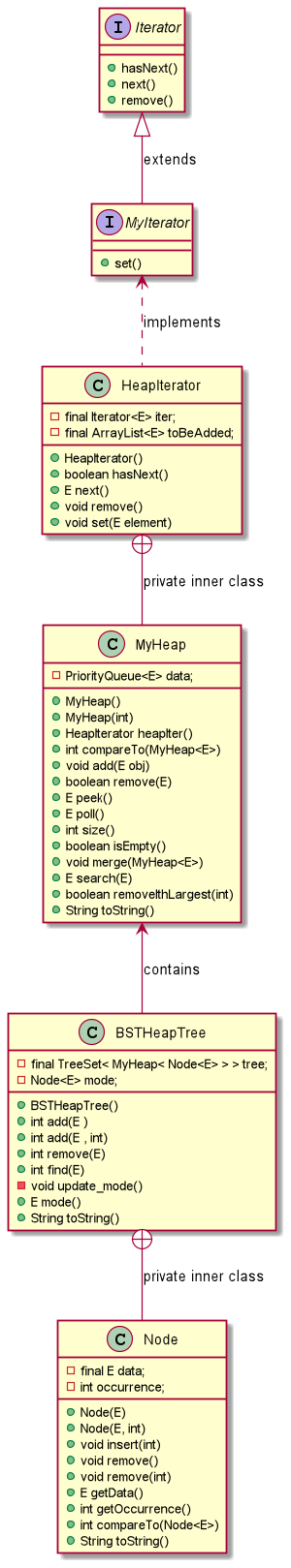
To insert or remove more than one element, amount should be specified.

In order the compare nodes, another node should be given as a parameter.



**INTENTIONALLY EMPTY**

1. **Class Diagram**
2. **Problem Solution Approach**

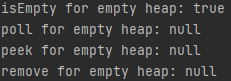
For the first part, I used a Priority Queue which has a Heap implementation in Java Collections Interface and created MyHeap Class. I delegated Priority Queue methods and improved them to suit Problem definition. Set method was problematic because changing a value while iterating was causing Exceptions. So, I removed the old values and hold the new values in an ArrayList. When there are no other elements to iterate, hasNext() method adds new elements to the heap.

For the second part I used a TreeSet which is a self-balancing binary tree in Java Collections Interface. TreeSet holds MyHeap objects. MyHeap objects needed to hold a value and its occurrence amount, so I created a basic class called Node which holds these two values. So TreeSet holds MyHeap objects and MyHeap objects holds Node objects. Adding an existing element again was problematic because same value could’ve been in a different heap. So instead of adding into the latest heap, I had to iterate through all heaps to see if the value exists or not, before adding. Instead of calculating mode everytime it is needed, I checked the occurrence of the element when a new element is added and compared it with existing mode’s occurrence. So, updating mode on add operation is constant time. But when an element is removed and if it was the recent mode, I had to search for a mode all over again so removing the mode is a little expensive.

1. **Test-Cases and Their Results**

**4.1) Test Cases for Part 1**

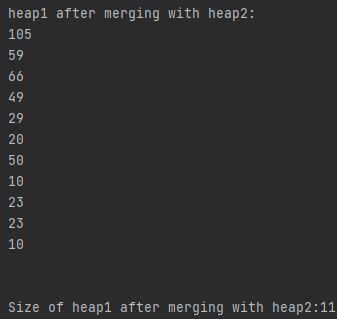
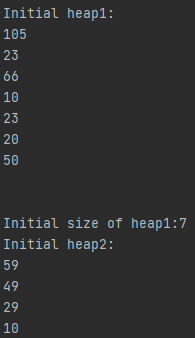
1 – Operations on empty heap



2- isEmpty on non-empty heap

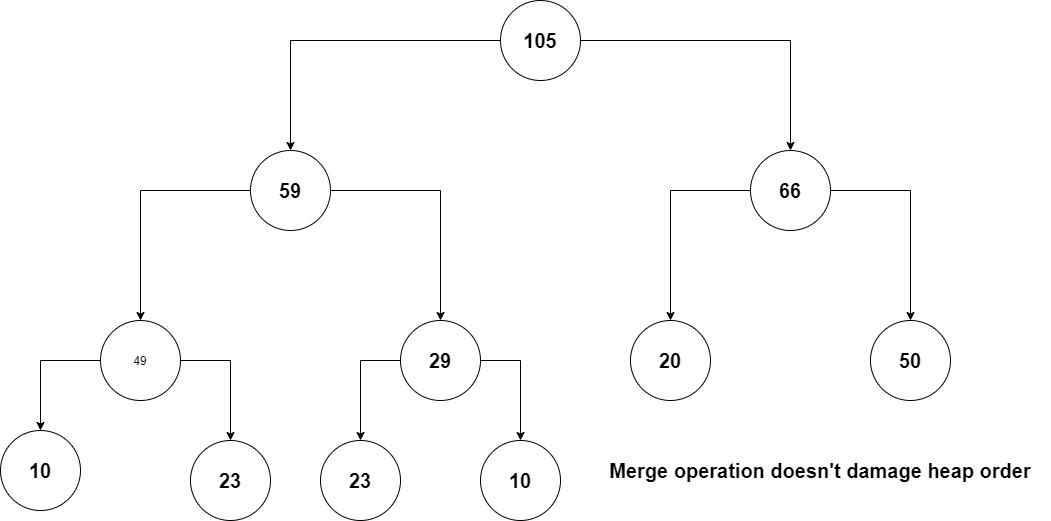


3- Merge operation

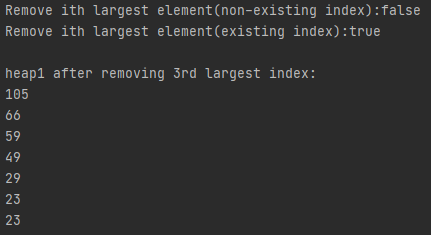








4)Remove

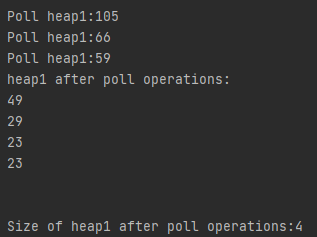
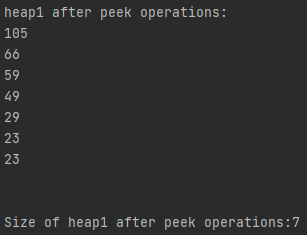


3rd largest index (50) of the merged heap is removed.

5)Search



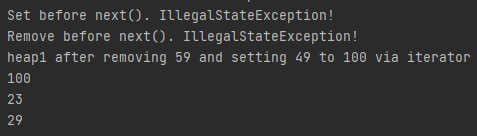
1. Peek and Poll



1. Remove on existing element



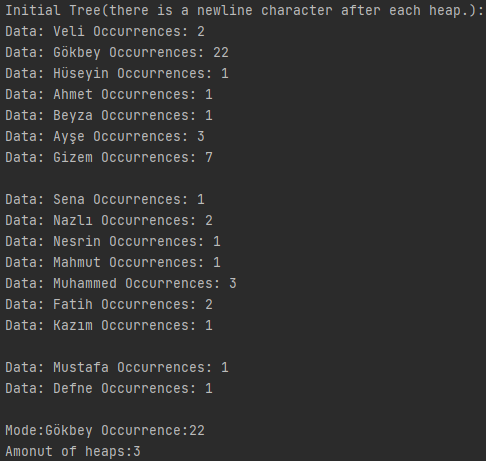
1. Iterator



**4.2) Test Cases for Part 2**

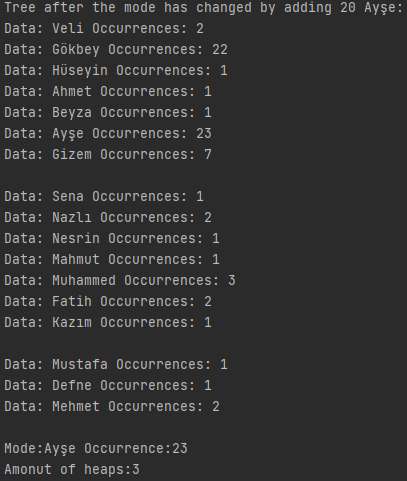
**4.2.a) String Test**

1-toString() method of BSTHeapTree

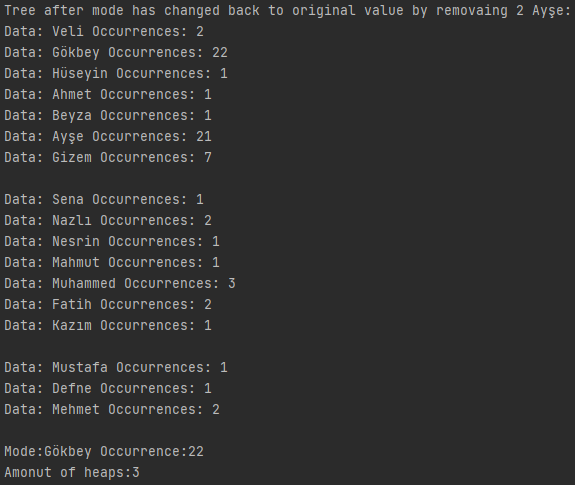
****

2-Return values of add and remove

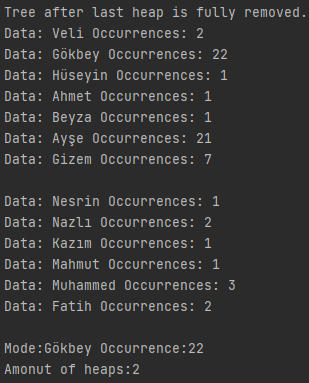
3-Dynamic mode after insertion



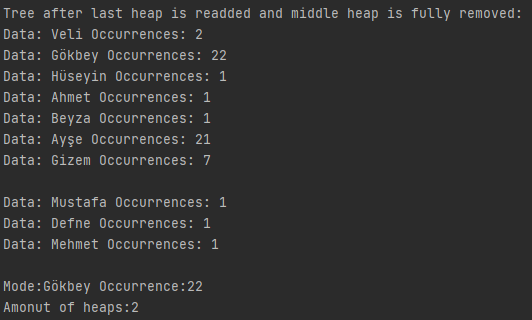
4- Dynamic mode after removal



5- Auto removal of the last Heap when all of its elements are removed.



6- Auto removal of a heap from middle when all of its elements are removed.



7- find method



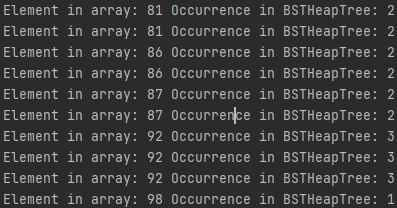


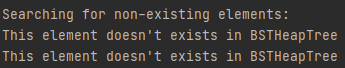
8- Error handling of remove method

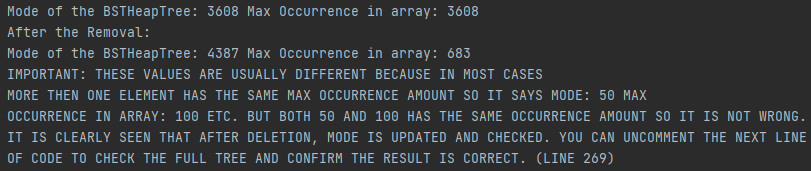
**4.2.b) Integer test with additional ArrayList**

In this test, there is a sorted ArrayList which holds the same data as BSTHeapTree for comparison Both are filled with same random values.

1- find method





2- Checking the mode

3- Removing non-existing elements

