Gebze Technical University
Computer Engineering

CSE 222/505 – Spring 2021

Homework 4 Report

Baran Solmaz
1801042601

1.SYSTEM REQUIREMENTS

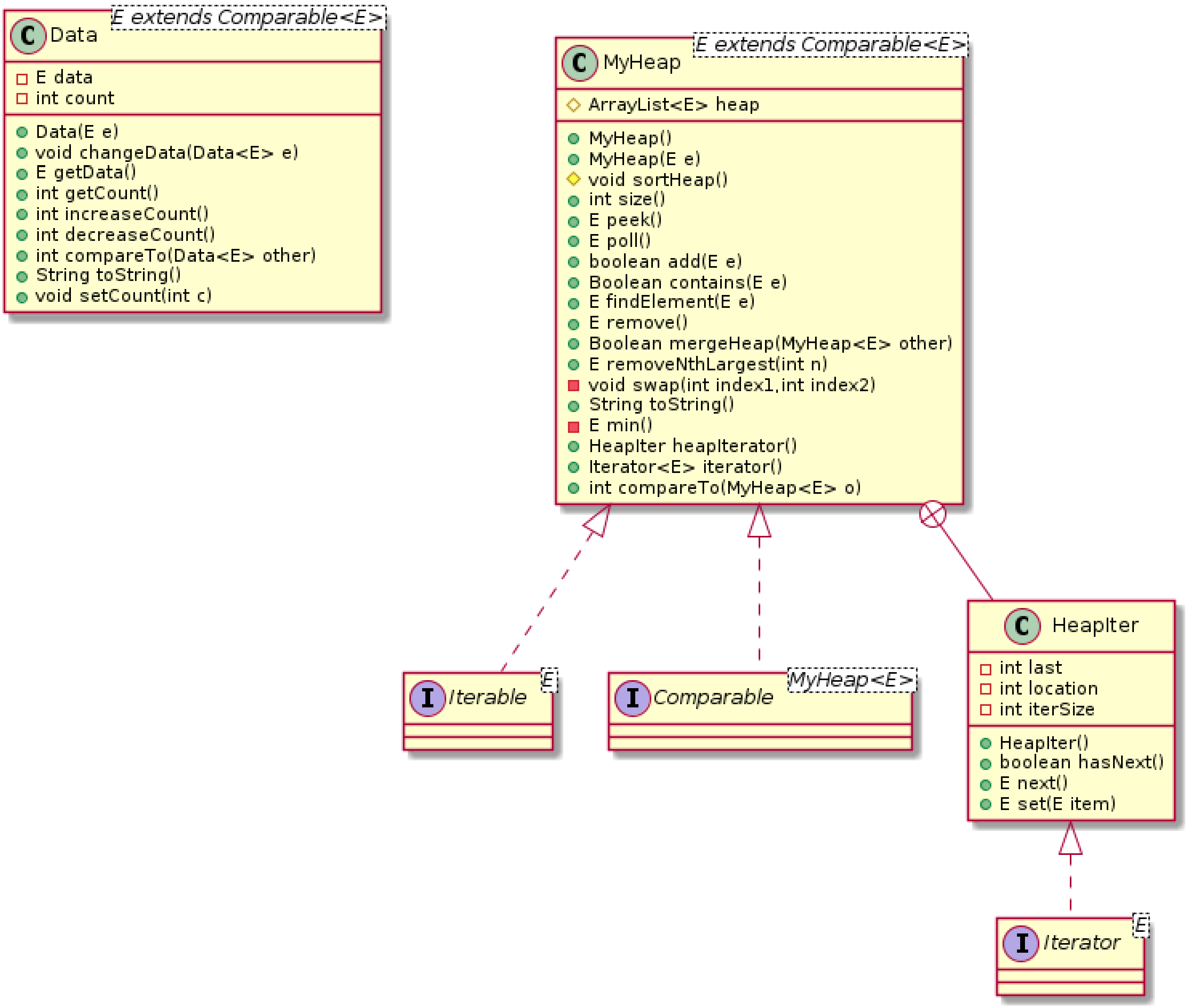
Part 1:

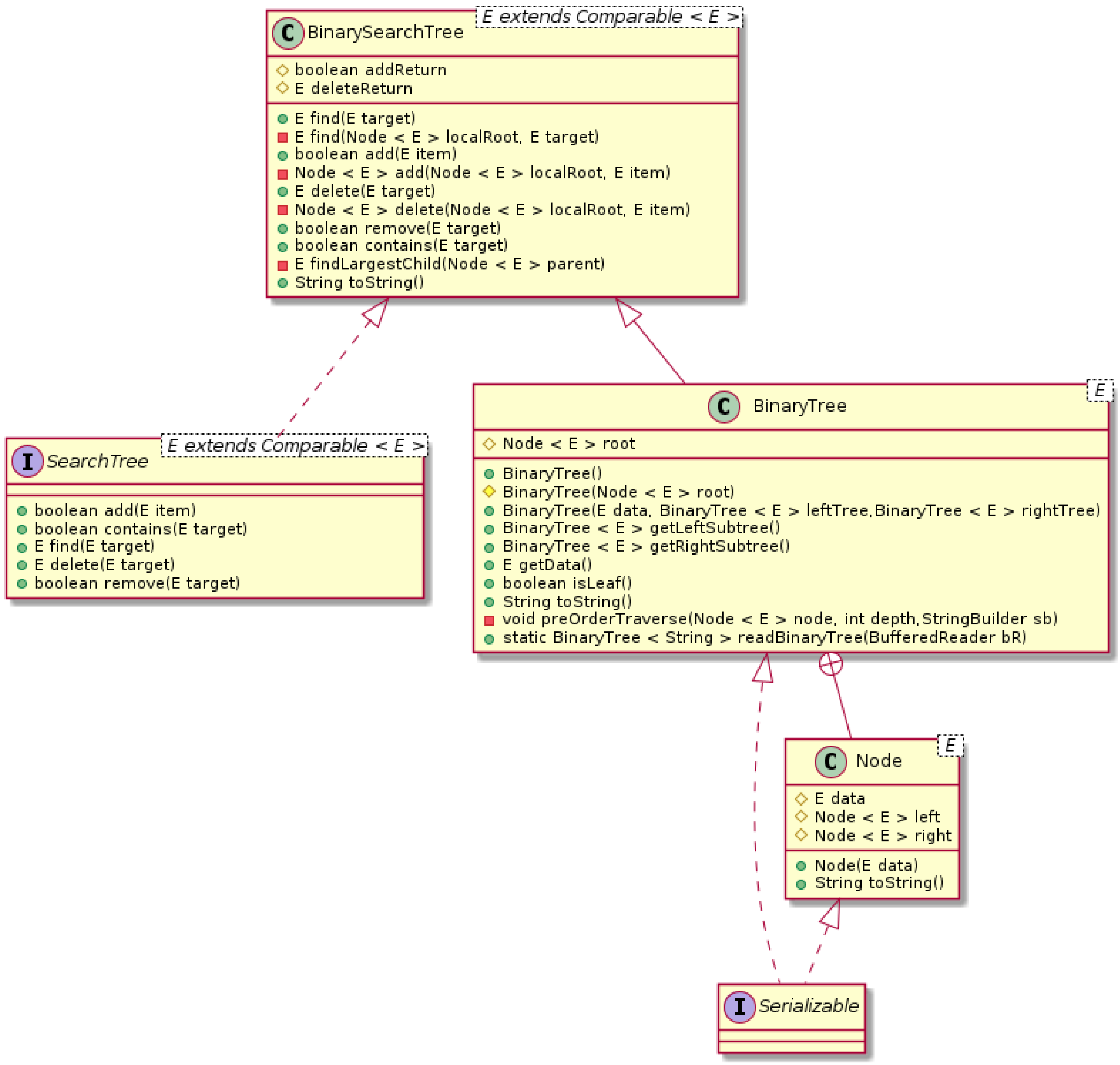
At least 2 not empty MyHeaps.

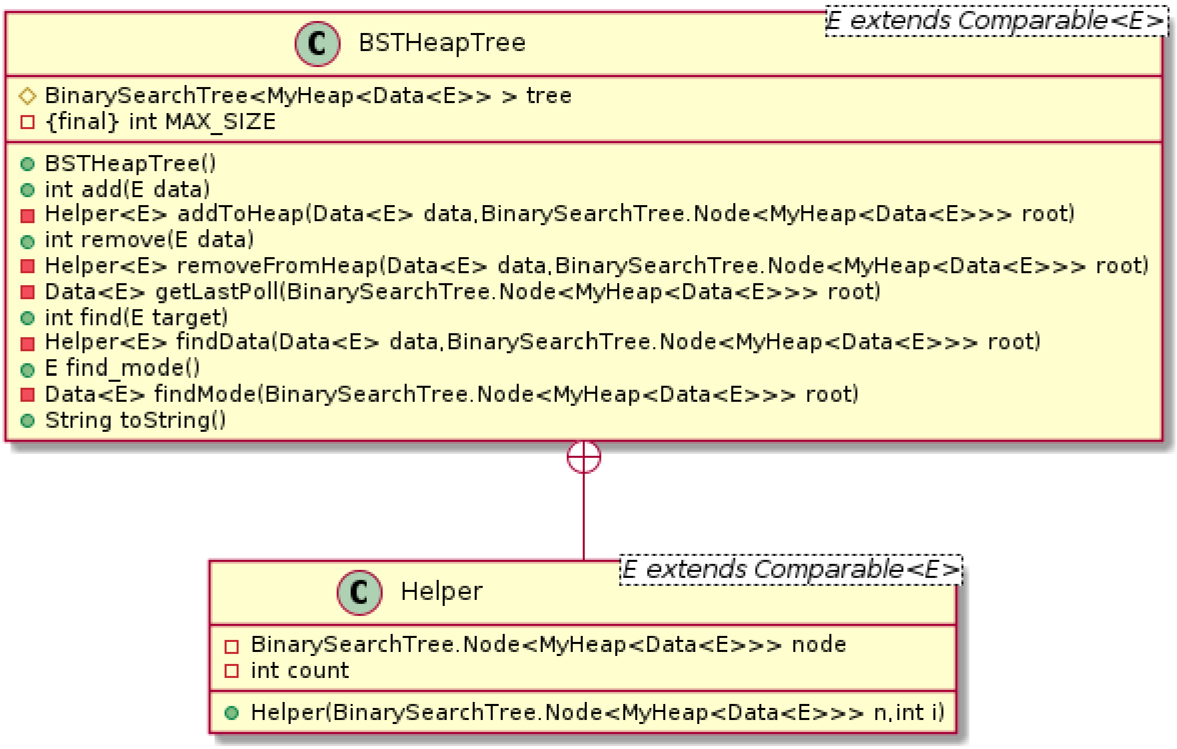
Part 2:

At least 1 not empty BSTHeapTree and ArrayList.

2.Diagrams




BinarySearchTree
E extends Comparable < E >
boolean addReturn
E deleteReturn
E find(E target)
E find(Node < E > localRoot, E target)
boolean add(E item)
Node < E > add(Node < E > localRoot, E item)
E delete(E target)
Node < E > delete(Node < E > localRoot, E item)
boolean remove(E target)
boolean contains(E target)
E findLargestChild(Node < E > parent)
String toString()
SearchTree
E extends Comparable < E >
boolean add(E item)
boolean contains(E target)
E find(E target)
E delete(E target)
boolean remove(E target)
BinaryTree
E
Node < E > root
BinaryTree()
BinaryTree(Node < E > root)
BinaryTree(E data, BinaryTree < E > leftTree,BinaryTree < E > rightTree)
BinaryTree < E > getLeftSubtree()
BinaryTree < E > getRightSubtree()
E getData()
boolean isLeaf()
String toString()
void preOrderTraverse(Node < E > node, int depth,StringBuilder sb)
static BinaryTree < String > readBinaryTree(BufferedReader bR)
Node
E
E data
Node < E > left
Node < E > right
Node(E data)
String toString()
Serializable


BSTHeapTree
E extends Comparable<E>
BinarySearchTree<MyHeap<Data<E>> > tree
{final} int MAX_SIZE
BSTHeapTree()
int add(E data)
Helper<E> addToHeap(Data<E> data,BinarySearchTree.Node<MyHeap<Data<E>>> root)
int remove(E data)
Helper<E> removeFromHeap(Data<E> data,BinarySearchTree.Node<MyHeap<Data<E>>> root)
Data<E> getLastPoll(BinarySearchTree.Node<MyHeap<Data<E>>> root)
int find(E target)
Helper<E> findData(Data<E> data,BinarySearchTree.Node<MyHeap<Data<E>>> root)
E find_mode()
Data<E> findMode(BinarySearchTree.Node<MyHeap<Data<E>>> root)
String toString()
Helper
E extends Comparable<E>
BinarySearchTree.Node<MyHeap<Data<E>>> node
int count
Helper(BinarySearchTree.Node<MyHeap<Data<E>>> n,int i)

3. PROBLEM SOLUTION APPROACH

Part 1:

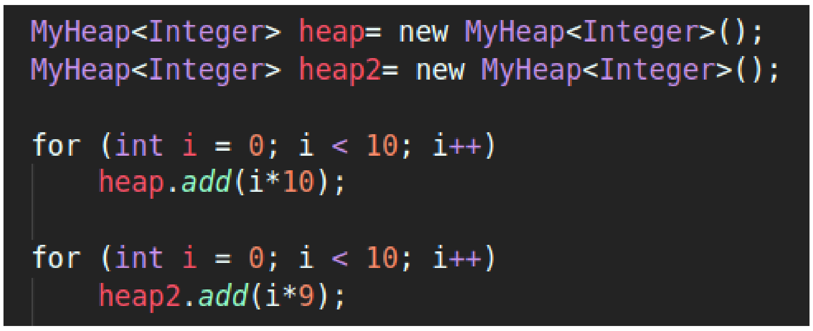
I created my own Max Heap class and inner HeapIter class to use iterator.

Part 2:

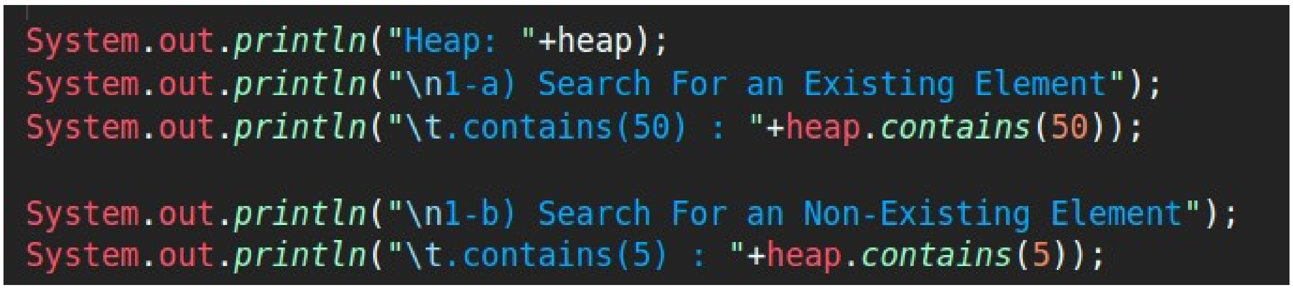
I used MyHeap class and the BinarySearchTree class that in the book to create BSTHeapTree class.

4. Test Cases

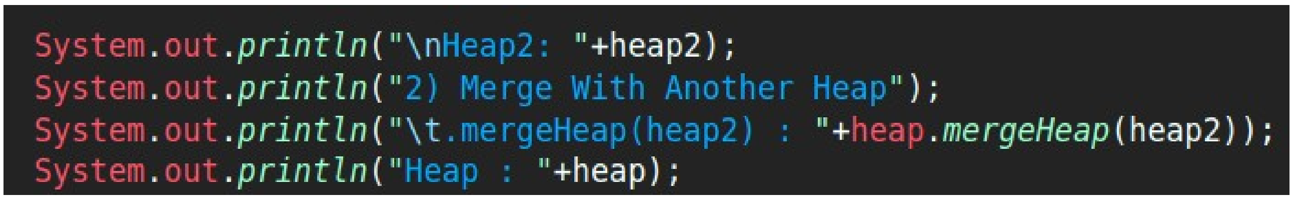
Part 1:



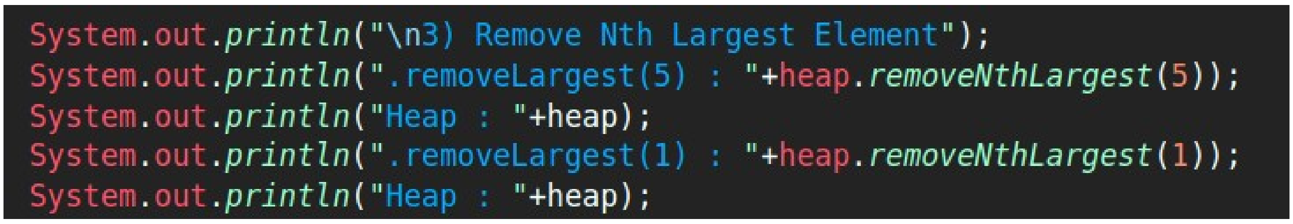
```
MyHeap<Integer> heap= new MyHeap<Integer>();
MyHeap<Integer> heap2= new MyHeap<Integer>();

for (int i = 0; i < 10; i++)
    heap.add(i*10);

for (int i = 0; i < 10; i++)
    heap2.add(i*9);
```



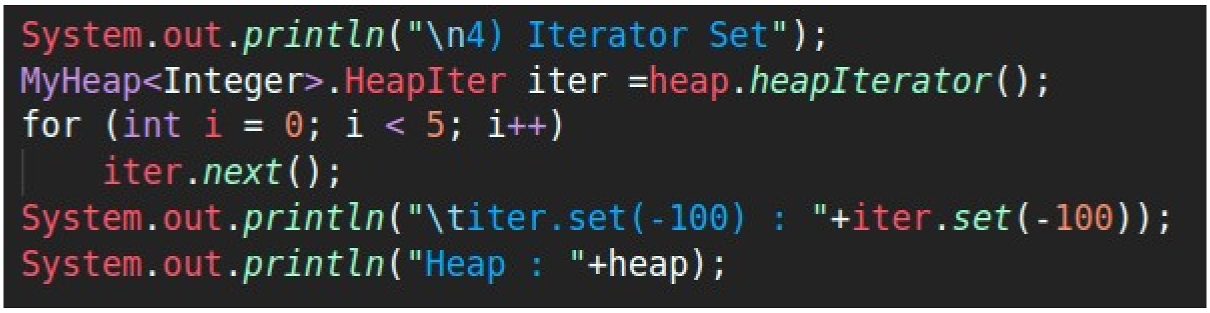
```
System.out.println("Heap:  "+heap);
System.out.println("\n1-a) Search For an Existing Element");
System.out.println("\t.contains(50) : "+heap.contains(50));

System.out.println("\n1-b) Search For an Non-Existing Element");
System.out.println("\t.contains(5) : "+heap.contains(5));
```



```
System.out.println("\nHeap2: "+heap2);
System.out.println("2) Merge With Another Heap");
System.out.println("\t.mergeHeap(heap2) : "+heap.mergeHeap(heap2));
System.out.println("Heap :  "+heap);
```



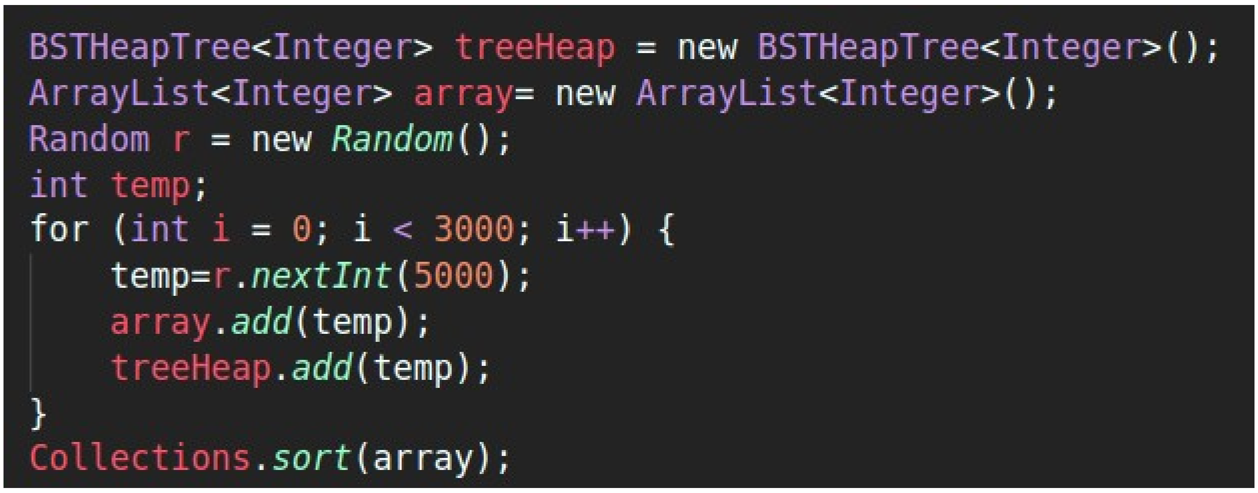
```
System.out.println("\n3) Remove Nth Largest Element");
System.out.println(".removeLargest(5) : "+heap.removeNthLargest(5));
System.out.println("Heap : "+heap);
System.out.println(".removeLargest(1) : "+heap.removeNthLargest(1));
System.out.println("Heap : "+heap);
```



```
System.out.println("\n4) Iterator Set");
MyHeap<Integer>.HeapIter iter =heap.heapIterator();
for (int i = 0; i < 5; i++)
    iter.next();
System.out.println("\titer.set(-100) : "+iter.set(-100));
System.out.println("Heap : "+heap);
```

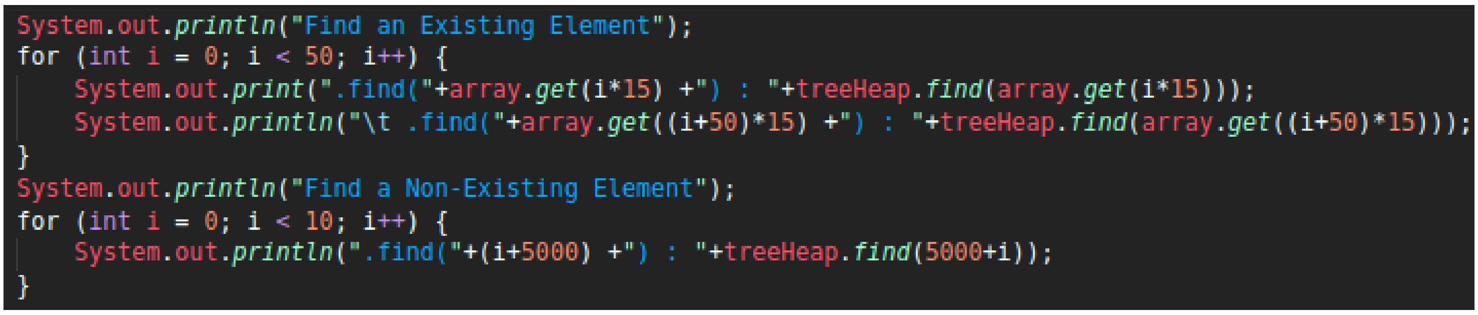
Part 2:

1. Insert the 3000 numbers that are randomly generated in the range 0-5000 into the BSTHeapTree. Store these numbers in an array as well. Sort the numbers to find the number occurrences of all the numbers.



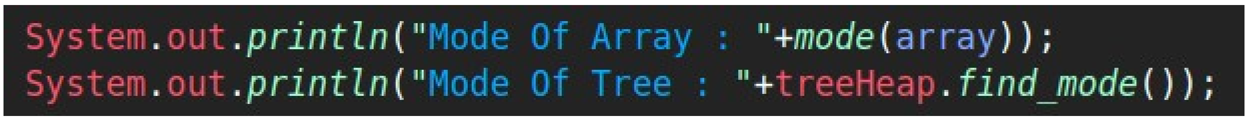
```java
BSTHeapTree<Integer> treeHeap = new BSTHeapTree<Integer>();
ArrayList<Integer> array= new ArrayList<Integer>();
Random r = new Random();
int temp;
for (int i = 0; i < 3000; i++) {
    temp=r.nextInt(5000);
    array.add(temp);
    treeHeap.add(temp);
}
Collections.sort(array);
```

2. Search for 100 numbers in the array and 10 numbers not in the array and make sure that the number of occurrences is correct.



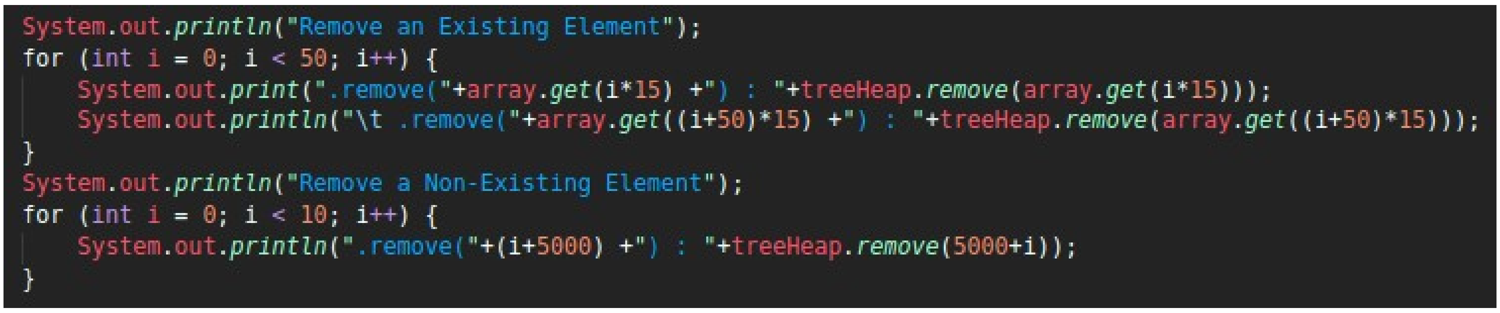
```java
System.out.println("Find an Existing Element");
for (int i = 0; i < 50; i++) {
    System.out.print(".find("+array.get(i*15) +") : "+treeHeap.find(array.get(i*15)));
    System.out.println("\t .find("+array.get((i+50)*15) +") : "+treeHeap.find(array.get((i+50)*15)));
}
System.out.println("Find a Non-Existing Element");
for (int i = 0; i < 10; i++) {
    System.out.println(".find("+(i+5000) +") : "+treeHeap.find(5000+i));
}
```

3. Find the mode of the BSTHeapTree. Check whether the mode value is correct.



```java
System.out.println("Mode Of Array : "+mode(array));
System.out.println("Mode Of Tree : "+treeHeap.find_mode());
```

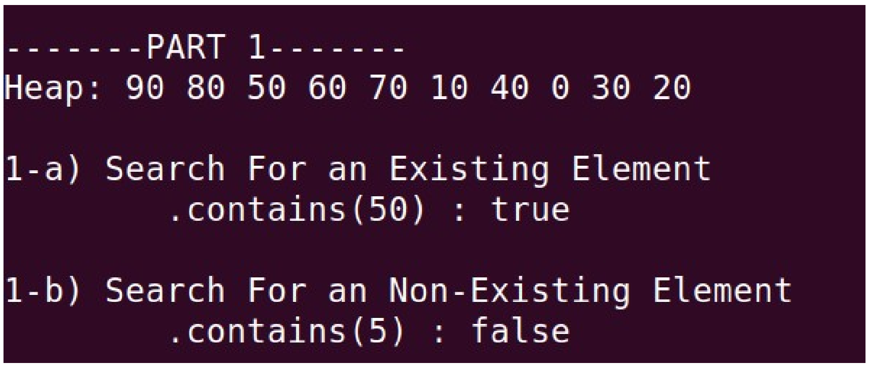
4. Remove 100 numbers in the array and 10 numbers not in the array and make sure that the number of occurrences after removal is correct.



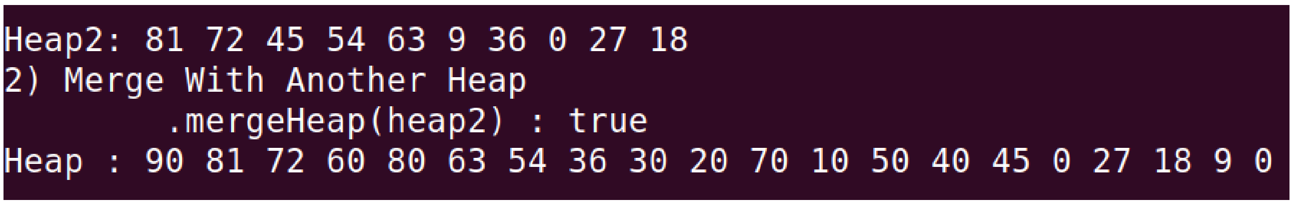
```java
System.out.println("Remove an Existing Element");
for (int i = 0; i < 50; i++) {
    System.out.print(".remove("+array.get(i*15) +") : "+treeHeap.remove(array.get(i*15)));
    System.out.println("\t .remove("+array.get((i+50)*15) +") : "+treeHeap.remove(array.get((i+50)*15)));
}
System.out.println("Remove a Non-Existing Element");
for (int i = 0; i < 10; i++) {
    System.out.println(".remove("+(i+5000) +") : "+treeHeap.remove(5000+i));
}
```

5. Running Commands and Results

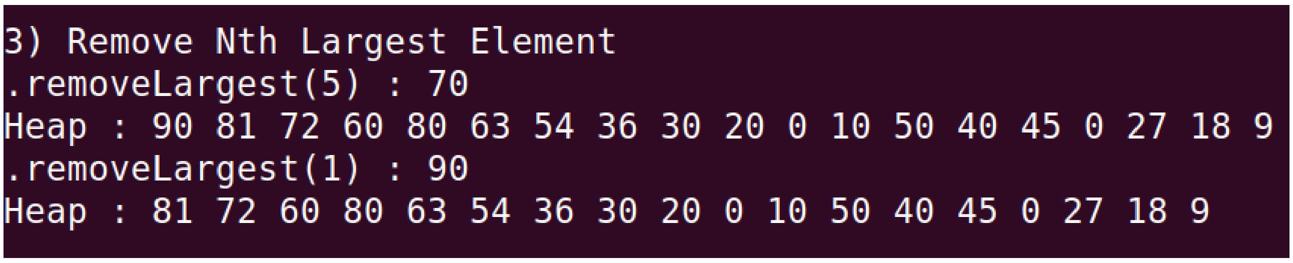
Part 1:



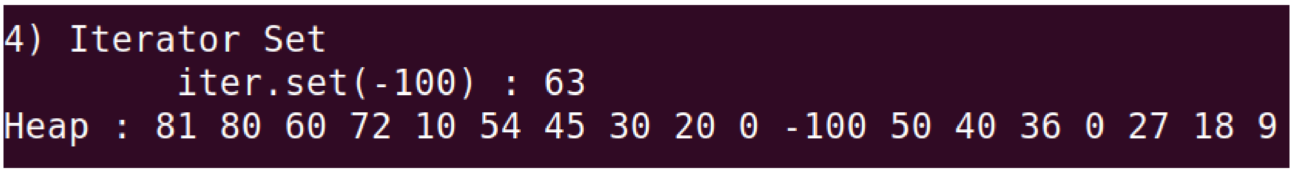
```
-------PART 1-------
Heap: 90 80 50 60 70 10 40 0 30 20

1-a) Search For an Existing Element
        .contains(50) : true

1-b) Search For an Non-Existing Element
        .contains(5) : false
```



```
Heap2: 81 72 45 54 63 9 36 0 27 18
2) Merge With Another Heap
        .mergeHeap(heap2) : true
Heap : 90 81 72 60 80 63 54 36 30 20 70 10 50 40 45 0 27 18 9 0
```

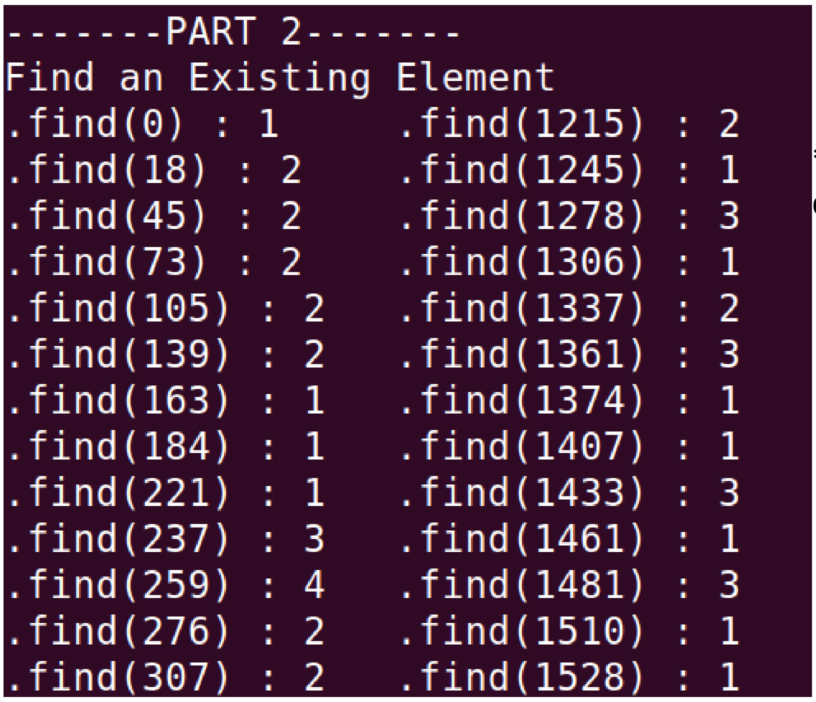


```
3) Remove Nth Largest Element
.removeLargest(5) : 70
Heap : 90 81 72 60 80 63 54 36 30 20 0 10 50 40 45 0 27 18 9
.removeLargest(1) : 90
Heap : 81 72 60 80 63 54 36 30 20 0 10 50 40 45 0 27 18 9
```



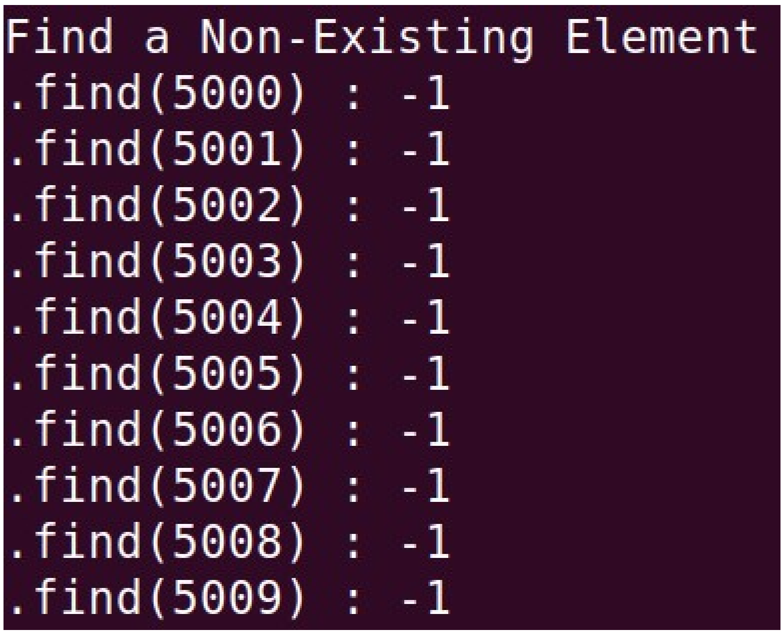
```
4) Iterator Set
        iter.set(-100) : 63
Heap : 81 80 60 72 10 54 45 30 20 0 -100 50 40 36 0 27 18 9
```

Part 2:

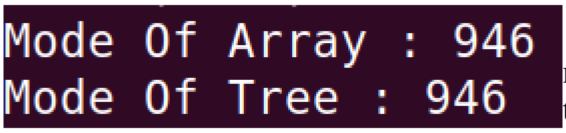


```
-------PART 2-------
Find an Existing Element
.find(0) : 1      .find(1215) : 2
.find(18) : 2     .find(1245) : 1
.find(45) : 2     .find(1278) : 3
.find(73) : 2     .find(1306) : 1
.find(105) : 2    .find(1337) : 2
.find(139) : 2    .find(1361) : 3
.find(163) : 1    .find(1374) : 1
.find(184) : 1    .find(1407) : 1
.find(221) : 1    .find(1433) : 3
.find(237) : 3    .find(1461) : 1
.find(259) : 4    .find(1481) : 3
.find(276) : 2    .find(1510) : 1
.find(307) : 2    .find(1528) : 1
```

*Numbers will be different in every run due to random

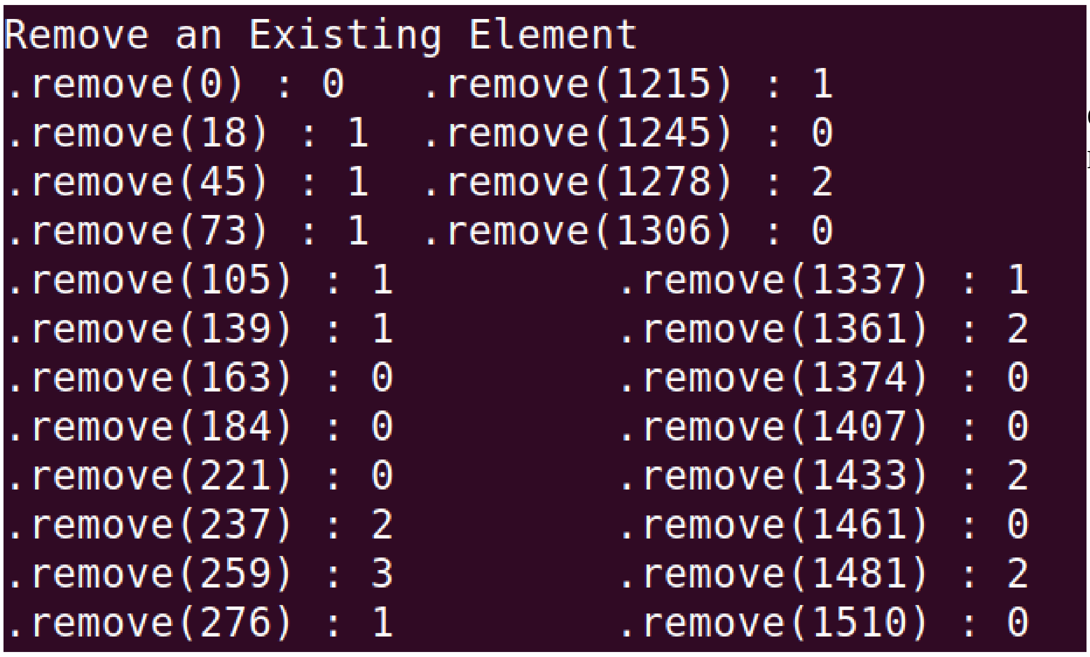


```
Find a Non-Existing Element
.find(5000) : -1
.find(5001) : -1
.find(5002) : -1
.find(5003) : -1
.find(5004) : -1
.find(5005) : -1
.find(5006) : -1
.find(5007) : -1
.find(5008) : -1
.find(5009) : -1
```



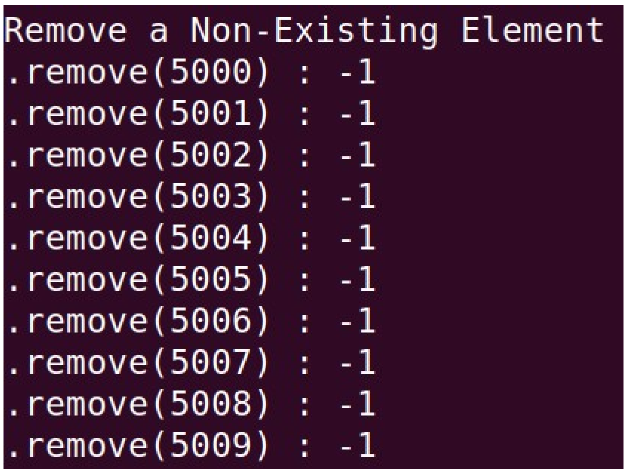
```
Mode Of Array : 946
Mode Of Tree : 946
```

*If the modes are different ,It doesn’t mean that methods are wrong. That means there are multiple numbers that has same amount of occurrence,so both of them are modes.



```
Remove an Existing Element
.remove(0) : 0   .remove(1215) : 1
.remove(18) : 1  .remove(1245) : 0
.remove(45) : 1  .remove(1278) : 2
.remove(73) : 1  .remove(1306) : 0
.remove(105) : 1        .remove(1337) : 1
.remove(139) : 1        .remove(1361) : 2
.remove(163) : 0        .remove(1374) : 0
.remove(184) : 0        .remove(1407) : 0
.remove(221) : 0        .remove(1433) : 2
.remove(237) : 2        .remove(1461) : 0
.remove(259) : 3        .remove(1481) : 2
.remove(276) : 1        .remove(1510) : 0
```

*Numbers will be different in every run due to random



```
Remove a Non-Existing Element
.remove(5000) : -1
.remove(5001) : -1
.remove(5002) : -1
.remove(5003) : -1
.remove(5004) : -1
.remove(5005) : -1
.remove(5006) : -1
.remove(5007) : -1
.remove(5008) : -1
.remove(5009) : -1
```