CSE222

DATA STRUCTURES AND ALGORITHMS

HOMEWORK 4 REPORT

HATİCE SEVRA GENÇ

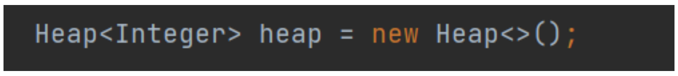
1801042611

Contents

A) PART1 REPORT

1. System Requirements

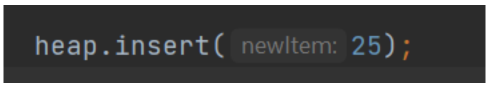
All classes in homework are of a generic type. Therefore, the type must be specified when declaring new objects.



```java
Heap<Integer> heap = new Heap<>();
```

Must create Heap object to start.

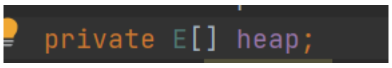
Must add an element to the heap with insert method.



```java
heap.insert( newItem: 25);
```

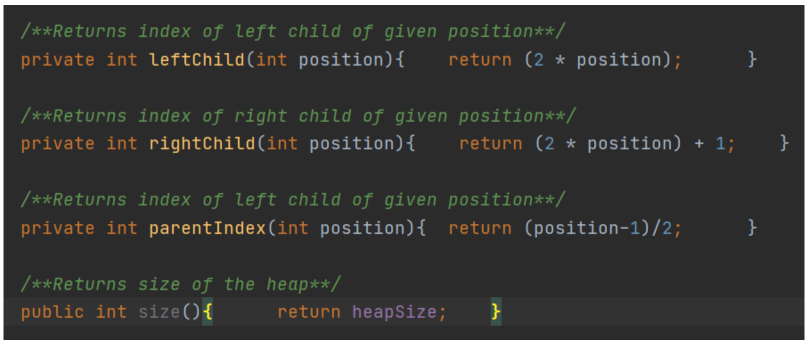
A driver function has been written to test functions easily. Functions are tested ubuntu and intellij.

2. Problem Solution Approach



```java
private E[] heap;
```

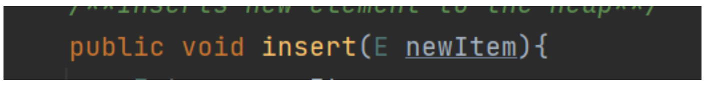
I implement a min-heap by using proper array. Array has one dimension which holds data. When it is necessary to change some part of data, it can easily do by index.



```java
/**Returns index of left child of given position**/
private int leftChild(int position){    return (2 * position);      }

/**Returns index of right child of given position**/
private int rightChild(int position){    return (2 * position) + 1;    }

/**Returns index of left child of given position**/
private int parentIndex(int position){  return (position-1)/2;      }

/**Returns size of the heap**/
public int size(){      return heapSize;    }
```

The leftChild() and rightChild() methods were written to easily reach the left and right children of a specific element. parentIndex() method also works vice versa.

depth() , size() , getRoot() and isEmpty() informations kept in own functions.



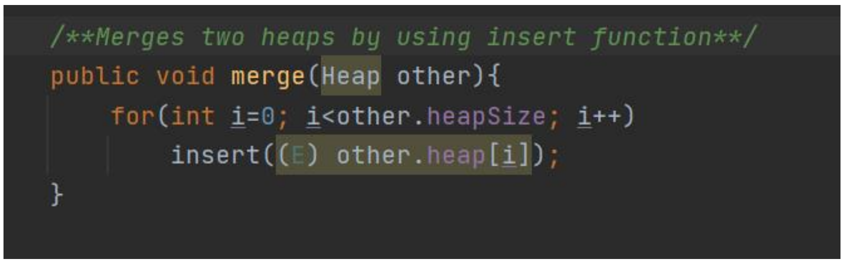
```java
public void insert(E newItem){
```

insert() function gets item to be add. This function doesn't check duplicate. Because heap allows it. insert () reallocates the array and adds a new element at the end of the heap. Then, it checks if the new array complies with the heap rules and makes the necessary swaps.

search() function try to find the given element. If can find prints the path from element to root and returns true. Otherwise prints error message end return false

Project have 2 remove functions with different names. The first one is removeWithIndex() . This function sorts the elements of the heap in ascending order with the bubbleSort() method. Then it finds the element to be removed based on the information given as parameter (xth largest element). Passes this element as a parameter to the other remove function-> removeItem(). Actual deletion and resort operations are done in removeItem function.

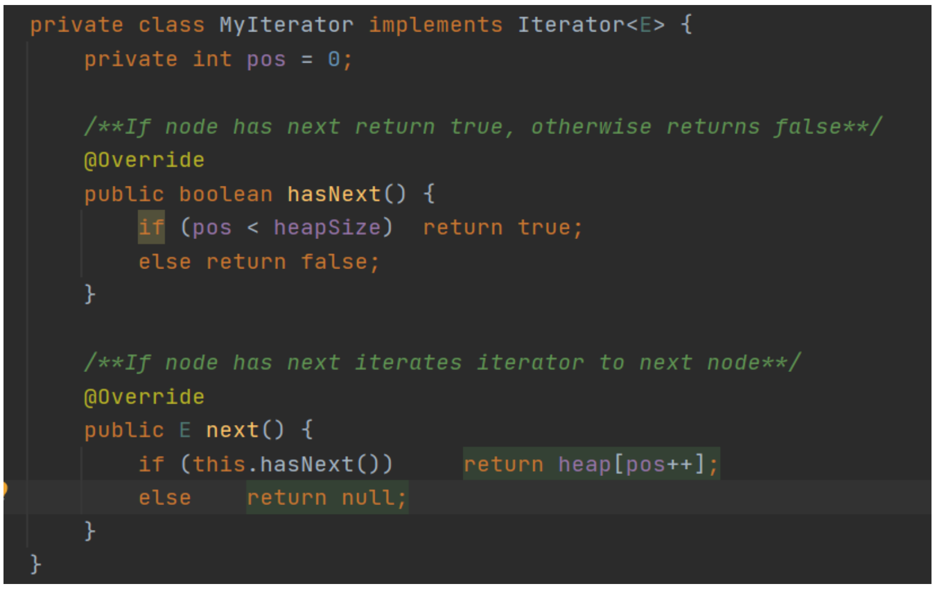
hasChildren() method is important to indicate end of the heap.
compare() method uses comparable interface. It's mission is to find the difference between generic parameters.



```java
/**Merges two heaps by using insert function**/
public void merge(Heap other){
    for(int i=0; i<other.heapSize; i++)
        insert((E) other.heap[i]);
}
```

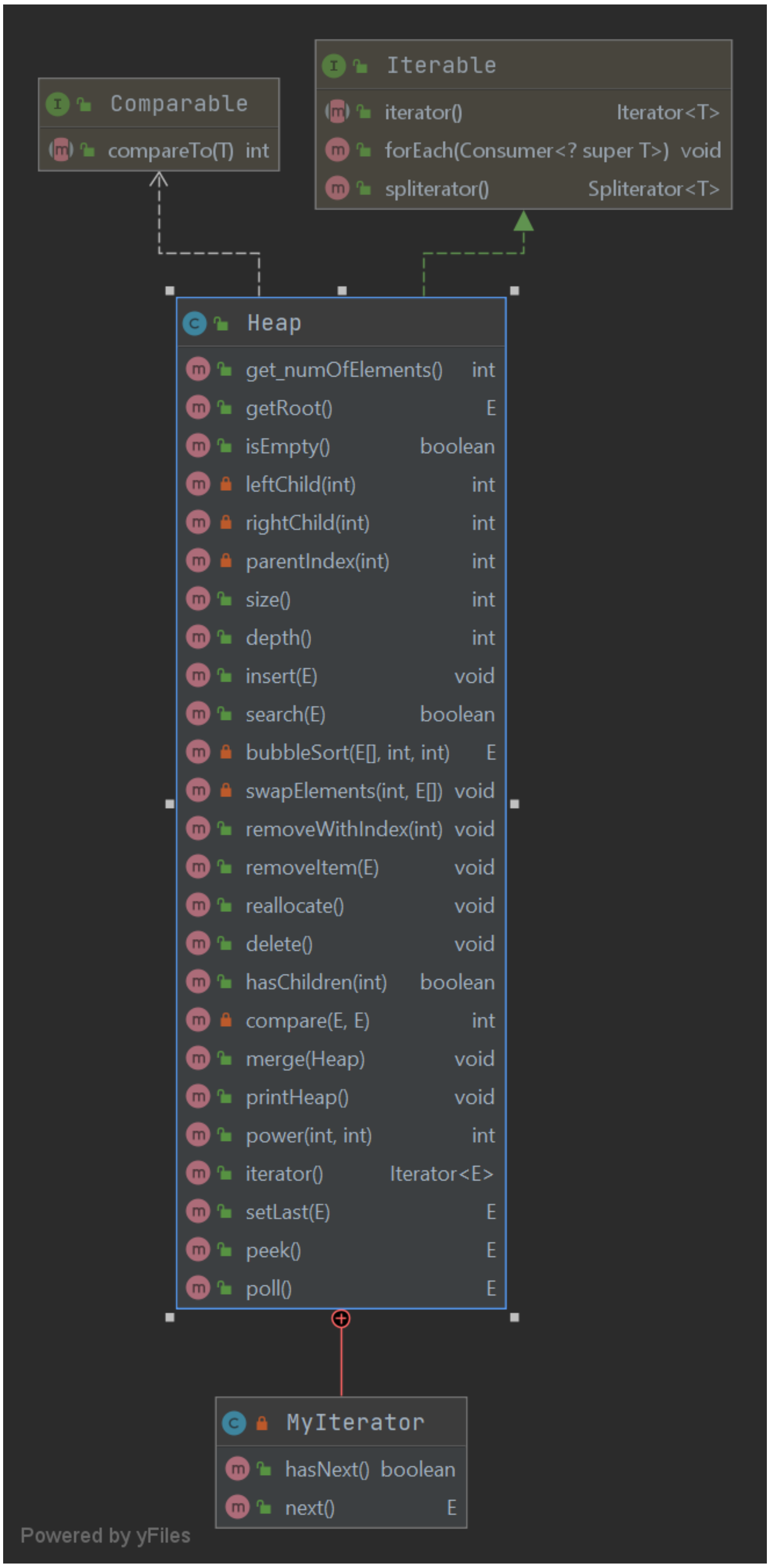
merge() method gets another heap object as parameter. With the help of the insert method, the two heap is combined.

I implement MyIterator class which implements Iterator<E> class. Through hasNext() and next() functions iterator visits all heap. In setLast() function, we find the last element using iterator. After set the last element, we return data before being changed.

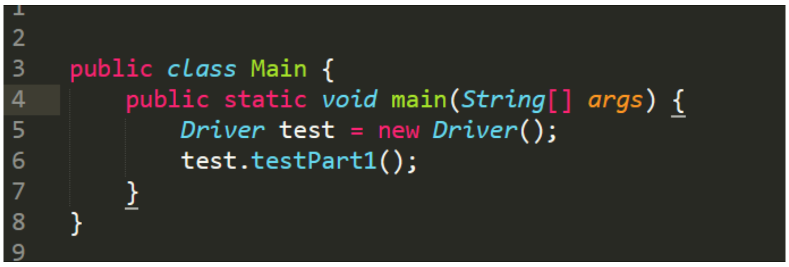


```java
private class MyIterator implements Iterator<E> {
    private int pos = 0;

    /**If node has next return true, otherwise returns false**/
    @Override
    public boolean hasNext() {
        if (pos < heapSize)  return true;
        else return false;
    }

    /**If node has next iterates iterator to next node**/
    @Override
    public E next() {
        if (this.hasNext())     return heap[pos++];
        else    return null;
    }
}
```

3. Class Diagrams



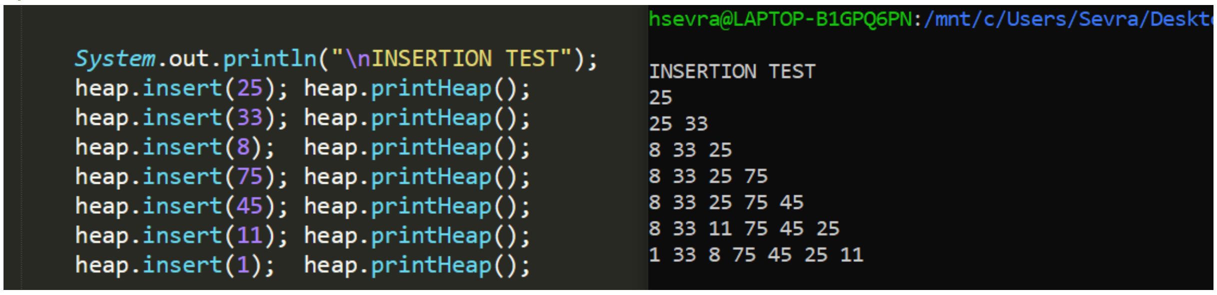
4. Test Cases and Running Results



```java

public class Main {
    public static void main(String[] args) {
        Driver test = new Driver();
        test.testPart1();
    }
}

```

Driver function runs in the main.

a) Insertion Test

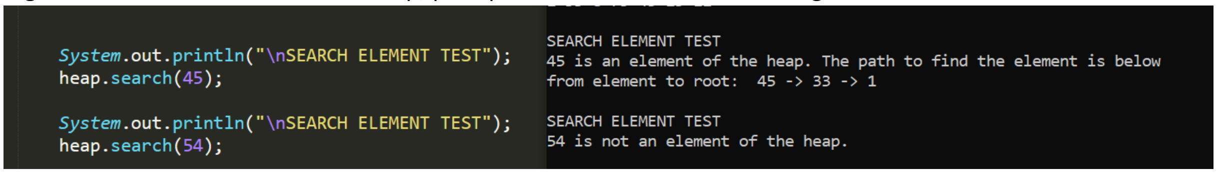


```java
System.out.println("\nINSERTION TEST");
heap.insert(25); heap.printHeap();
heap.insert(33); heap.printHeap();
heap.insert(8);  heap.printHeap();
heap.insert(75); heap.printHeap();
heap.insert(45); heap.printHeap();
heap.insert(11); heap.printHeap();
heap.insert(1);  heap.printHeap();
```

```
hsevra@LAPTOP-B1GPQ6PN:/mnt/c/Users/Sevra/Deskt

INSERTION TEST
25
25 33
8 33 25
8 33 25 75
8 33 25 75 45
8 33 11 75 45 25
1 33 8 75 45 25 11
```

b)Search Element Test

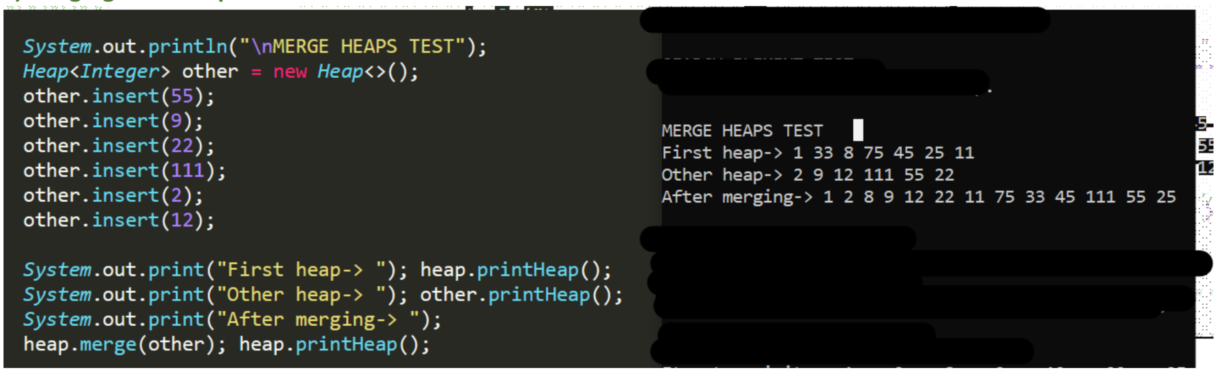
If given item is an element of the heap, path prints. Otherwise error message seen on the screen.



```java
System.out.println("\nSEARCH ELEMENT TEST");
heap.search(45);

System.out.println("\nSEARCH ELEMENT TEST");
heap.search(54);
```

```
SEARCH ELEMENT TEST
45 is an element of the heap. The path to find the element is below
from element to root:  45 -> 33 -> 1

SEARCH ELEMENT TEST
54 is not an element of the heap.
```

c)Merging Two Heaps Test

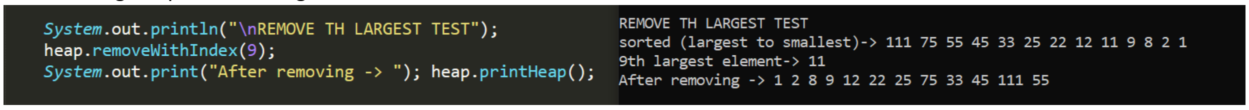


```java
System.out.println("\nMERGE HEAPS TEST");
Heap<Integer> other = new Heap<>();
other.insert(55);
other.insert(9);
other.insert(22);
other.insert(111);
other.insert(2);
other.insert(12);

System.out.print("First heap-> "); heap.printHeap();
System.out.print("Other heap-> "); other.printHeap();
System.out.print("After merging-> ");
heap.merge(other); heap.printHeap();
```

```
[redacted]

MERGE HEAPS TEST
First heap-> 1 33 8 75 45 25 11
Other heap-> 2 9 12 111 55 22
After merging-> 1 2 8 9 12 22 11 75 33 45 111 55 25
[redacted]
```

d)Removing x^{th} Element Test

After sorting heap in ascending order

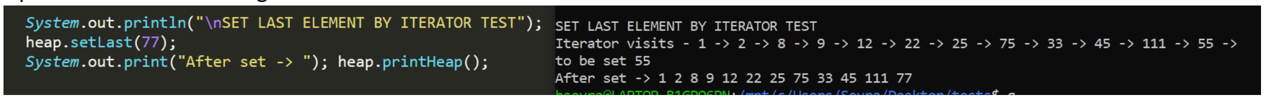


```java
System.out.println("\nREMOVE TH LARGEST TEST");
heap.removeWithIndex(9);
System.out.print("After removing -> "); heap.printHeap();
```

```
REMOVE TH LARGEST TEST
sorted (largest to smallest)-> 111 75 55 45 33 25 22 12 11 9 8 2 1
9th largest element-> 11
After removing -> 1 2 8 9 12 22 25 75 33 45 111 55
```

e) Set Last Element Test

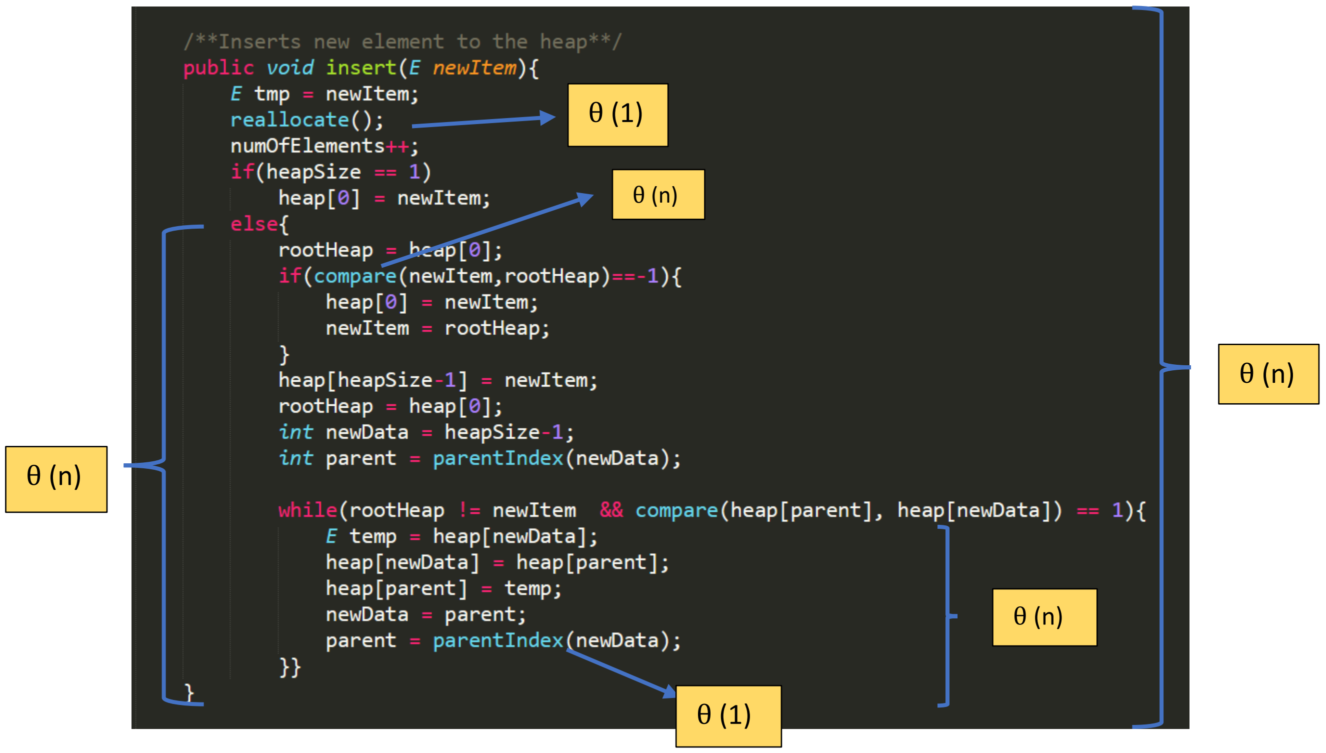
Updates last element using iterator



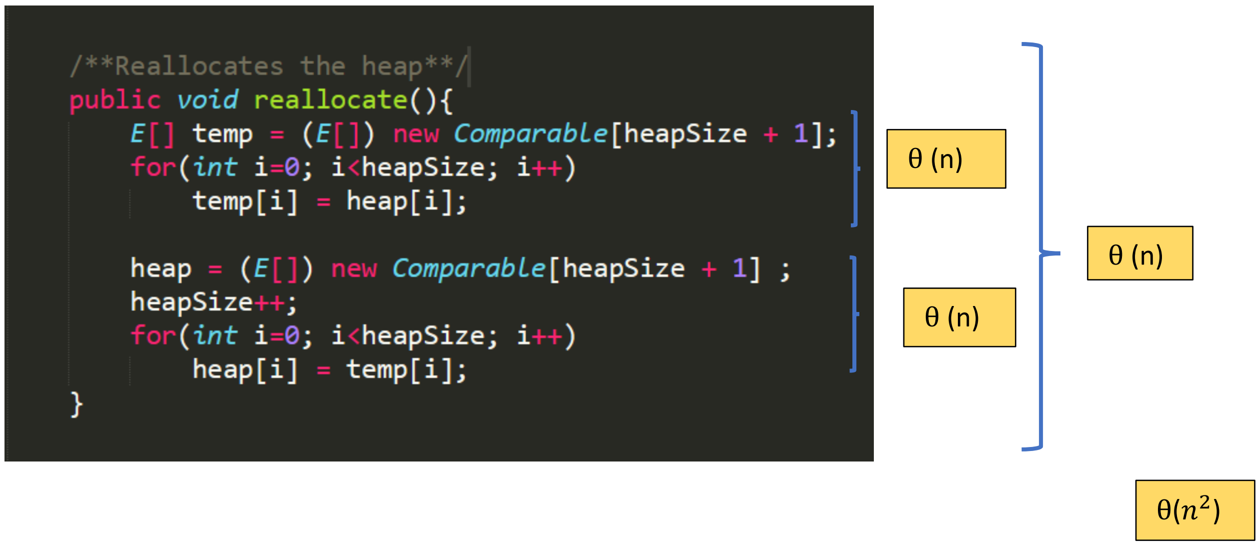
```java
System.out.println("\nSET LAST ELEMENT BY ITERATOR TEST");
heap.setLast(77);
System.out.print("After set -> "); heap.printHeap();
```

```
SET LAST ELEMENT BY ITERATOR TEST
Iterator visits - 1 -> 2 -> 8 -> 9 -> 12 -> 22 -> 25 -> 75 -> 33 -> 45 -> 111 -> 55 ->
to be set 55
After set -> 1 2 8 9 12 22 25 75 33 45 111 77
```

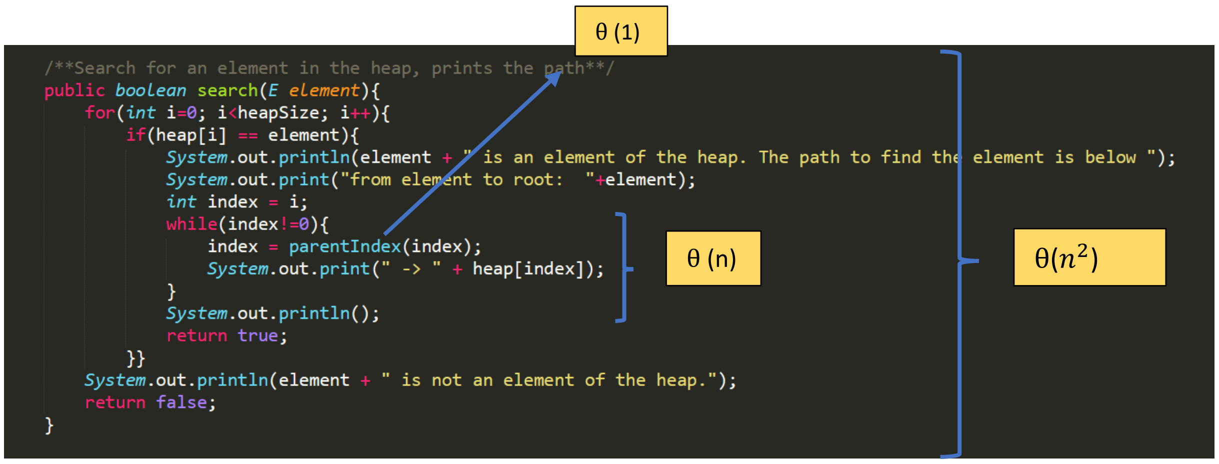
B) ANALYSIS OF PART1



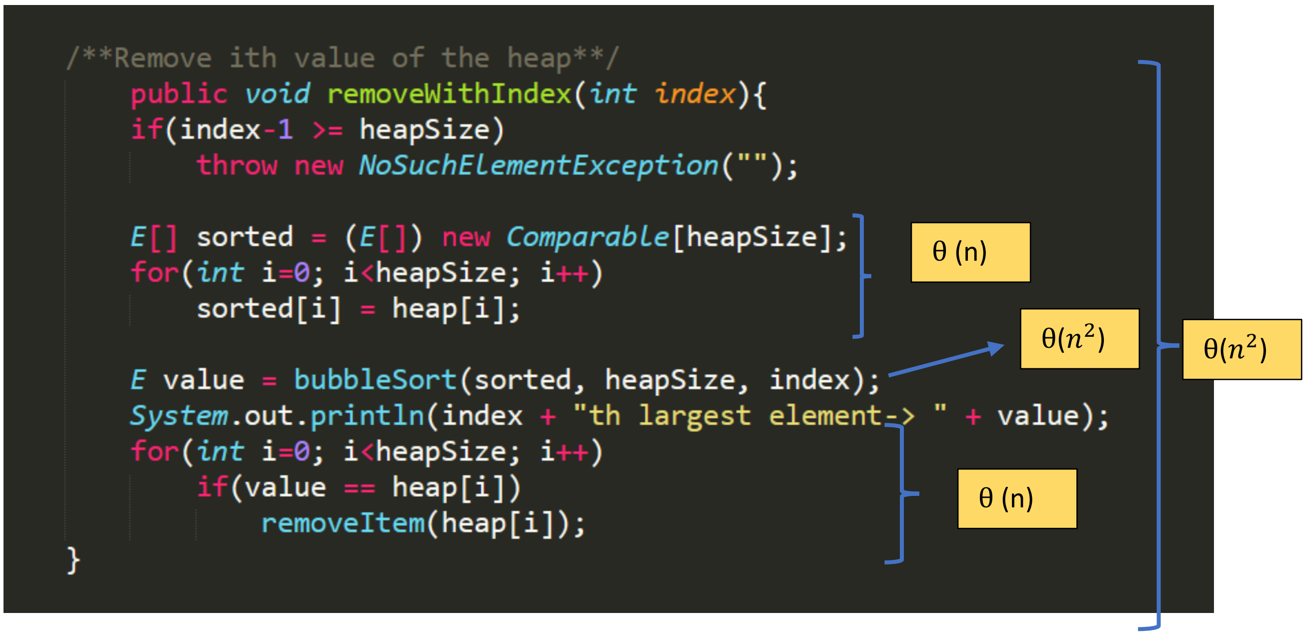
```java
/**Inserts new element to the heap**/
public void insert(E newItem){
    E tmp = newItem;
    reallocate();
    numOfElements++;
    if(heapSize == 1)
        heap[0] = newItem;
    else{
        rootHeap = heap[0];
        if(compare(newItem,rootHeap)==-1){
            heap[0] = newItem;
            newItem = rootHeap;
        }
        heap[heapSize-1] = newItem;
        rootHeap = heap[0];
        int newData = heapSize-1;
        int parent = parentIndex(newData);

        while(rootHeap != newItem  && compare(heap[parent], heap[newData]) == 1){
            E temp = heap[newData];
            heap[newData] = heap[parent];
            heap[parent] = temp;
            newData = parent;
            parent = parentIndex(newData);
        }}
}
```



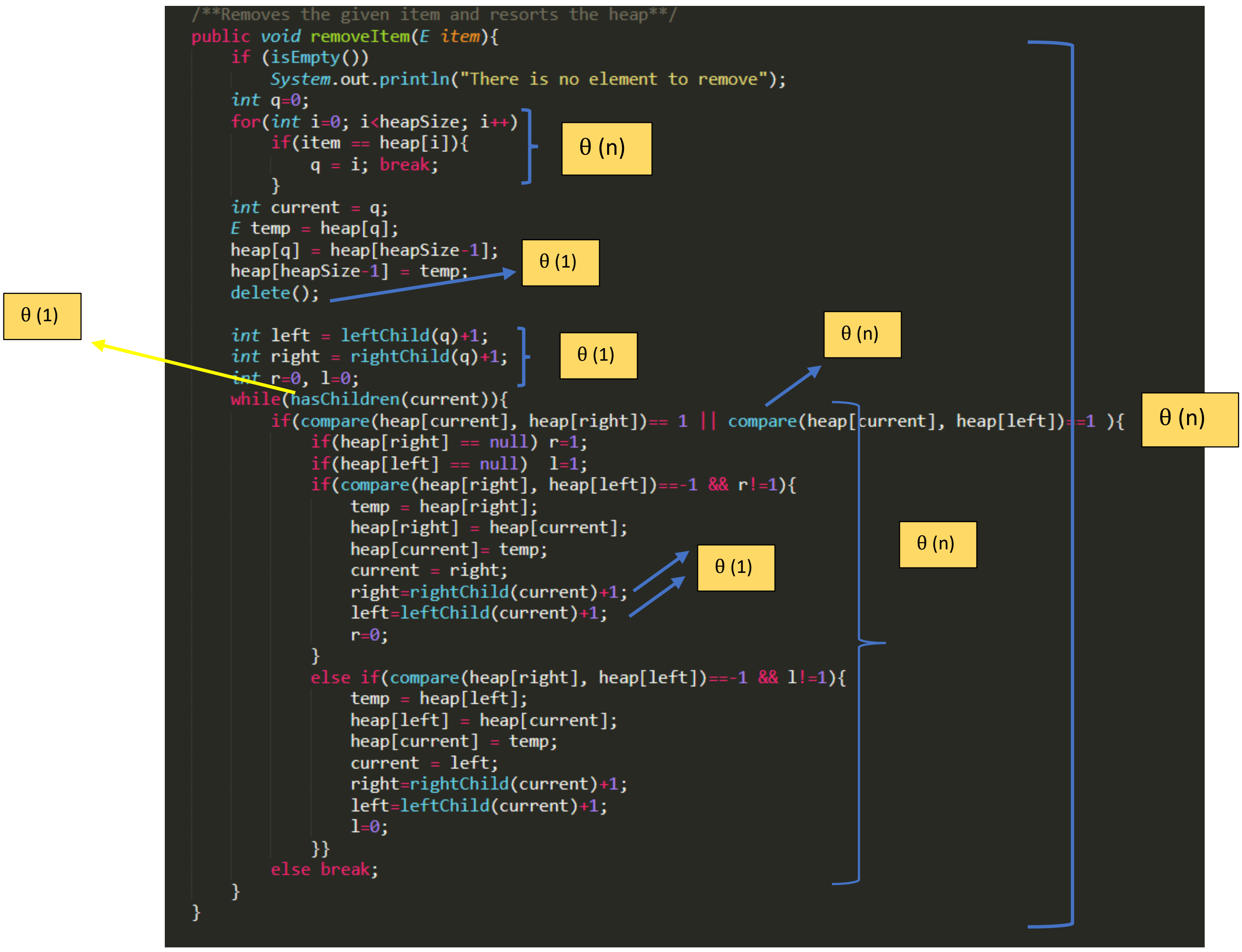
```java
/**Reallocates the heap**/
public void reallocate(){
    E[] temp = (E[]) new Comparable[heapSize + 1];
    for(int i=0; i<heapSize; i++)
        temp[i] = heap[i];

    heap = (E[]) new Comparable[heapSize + 1] ;
    heapSize++;
    for(int i=0; i<heapSize; i++)
        heap[i] = temp[i];
}
```



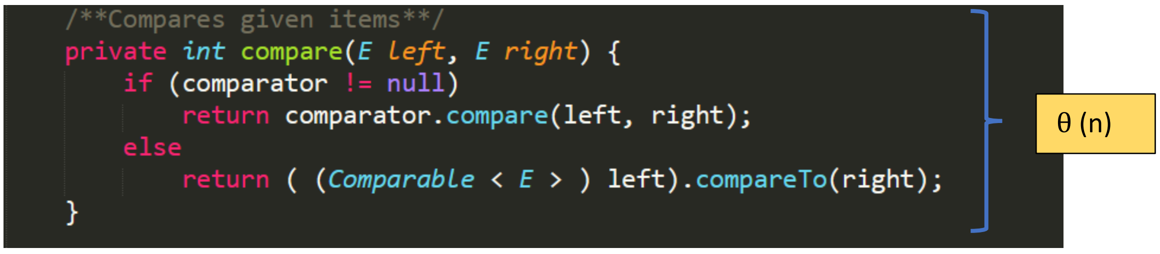
```java
/**Search for an element in the heap, prints the path**/
public boolean search(E element){
    for(int i=0; i<heapSize; i++){
        if(heap[i] == element){
            System.out.println(element + " is an element of the heap. The path to find the element is below ");
            System.out.print("from element to root:  "+element);
            int index = i;
            while(index!=0){
                index = parentIndex(index);
                System.out.print(" -> " + heap[index]);
            }
            System.out.println();
            return true;
        }}
    System.out.println(element + " is not an element of the heap.");
    return false;
}
```



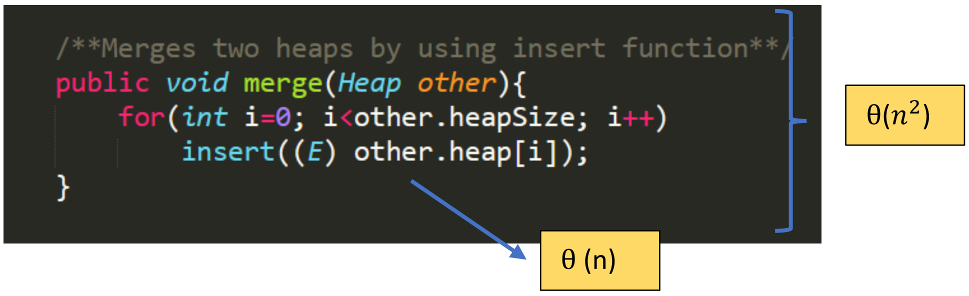
```java
/**Remove ith value of the heap**/
    public void removeWithIndex(int index){
    if(index-1 >= heapSize)
        throw new NoSuchElementException("");

    E[] sorted = (E[]) new Comparable[heapSize];
    for(int i=0; i<heapSize; i++)
        sorted[i] = heap[i];

    E value = bubbleSort(sorted, heapSize, index);
    System.out.println(index + "th largest element-> " + value);
    for(int i=0; i<heapSize; i++)
        if(value == heap[i])
            removeItem(heap[i]);
}
```



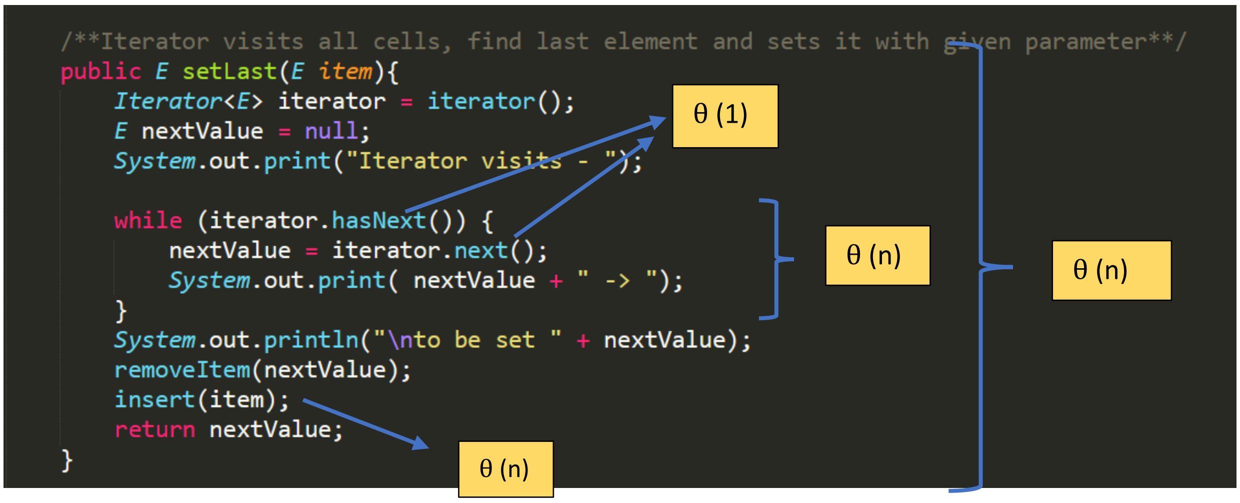
```java
/**Removes the given item and resorts the heap**/
public void removeItem(E item){
    if (isEmpty())
        System.out.println("There is no element to remove");
    int q=0;
    for(int i=0; i<heapSize; i++)
        if(item == heap[i]){
            q = i; break;
        }
    int current = q;
    E temp = heap[q];
    heap[q] = heap[heapSize-1];
    heap[heapSize-1] = temp;
    delete();

    int left = leftChild(q)+1;
    int right = rightChild(q)+1;
    int r=0, l=0;
    while(hasChildren(current)){
        if(compare(heap[current], heap[right])== 1 || compare(heap[current], heap[left])==1 ){
            if(heap[right] == null) r=1;
            if(heap[left] == null)  l=1;
            if(compare(heap[right], heap[left])==-1 && r!=1){
                temp = heap[right];
                heap[right] = heap[current];
                heap[current]= temp;
                current = right;
                right=rightChild(current)+1;
                left=leftChild(current)+1;
                r=0;
            }
            else if(compare(heap[right], heap[left])==-1 && l!=1){
                temp = heap[left];
                heap[left] = heap[current];
                heap[current] = temp;
                current = left;
                right=rightChild(current)+1;
                left=leftChild(current)+1;
                l=0;
            }}
        else break;
    }
}
```



```java
/**Compares given items**/
private int compare(E left, E right) {
    if (comparator != null)
        return comparator.compare(left, right);
    else
        return ( (Comparable < E > ) left).compareTo(right);
}
```



```java
/**Merges two heaps by using insert function**/
public void merge(Heap other){
    for(int i=0; i<other.heapSize; i++)
        insert((E) other.heap[i]);
}
```



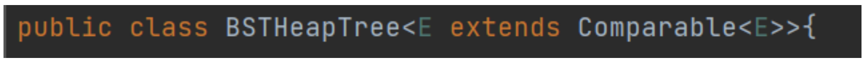
```java
/**Iterator visits all cells, find last element and sets it with given parameter**/
public E setLast(E item){
    Iterator<E> iterator = iterator();
    E nextValue = null;
    System.out.print("Iterator visits - ");

    while (iterator.hasNext()) {
        nextValue = iterator.next();
        System.out.print( nextValue + " -> ");
    }
    System.out.println("\nto be set " + nextValue);
    removeItem(nextValue);
    insert(item);
    return nextValue;
}
```

C) PART2 (NOT FINISHED)

1. System Requirements

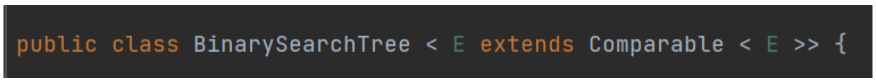
All Part2 classes are generic type.
In this part, our aim was to keep the heap structure in the nodes of Binary Search Trees within the BSTHeapTree structure.

BSTHeapTree Class defined :



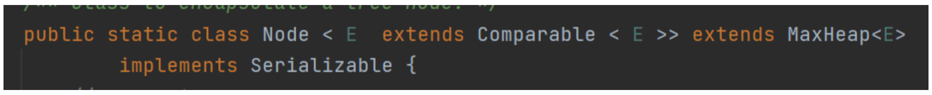
```java
public class BSTHeapTree<E extends Comparable<E>>{
```

BinarySearchTree Class defined:



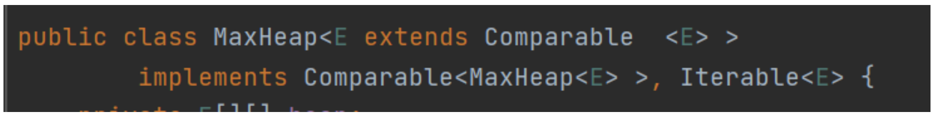
```java
public class BinarySearchTree < E extends Comparable < E >> {
```

BinarySearchTree has inner class named Node<E>



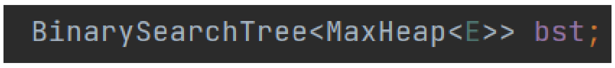
```java
public static class Node < E  extends Comparable < E >> extends MaxHeap<E>
        implements Serializable {
```

MaxHeap Class defined:



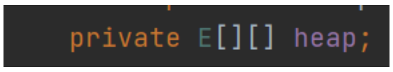
```java
public class MaxHeap<E extends Comparable  <E> >
        implements Comparable<MaxHeap<E> >, Iterable<E> {
```

To keep MaxHeap in BinarySearchTree, I implement a tree named bst in the BSTHeapTree class. And started in constructor.



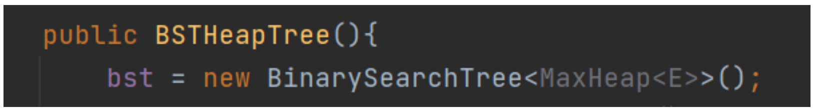
```java
BinarySearchTree<MaxHeap<E>> bst;
```

Heap array in the maxHeap class has two dimension. First one holds the data, the other one holds number of occurances of the data.



```java
private E[][] heap;
```

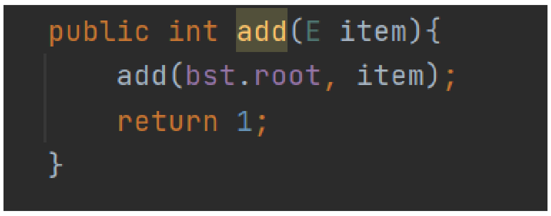
2. Problem Solution Approach



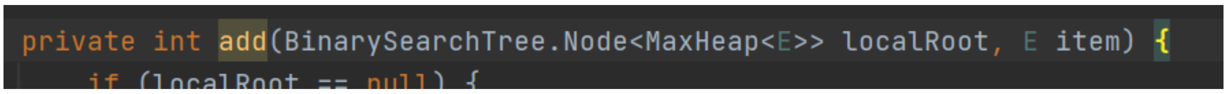
```java
public BSTHeapTree(){
    bst = new BinarySearchTree<MaxHeap<E>>();
```

I initialized the bst in the constructor

Call add method -with 1 param- to insert a new element to bst (and heap)

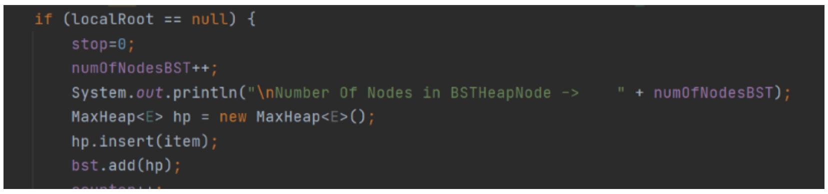


```java
public int add(E item){
    add(bst.root, item);
    return 1;
}
```



```java
private int add(BinarySearchTree.Node<MaxHeap<E>> localRoot, E item) {
    if (localRoot == null) {
```

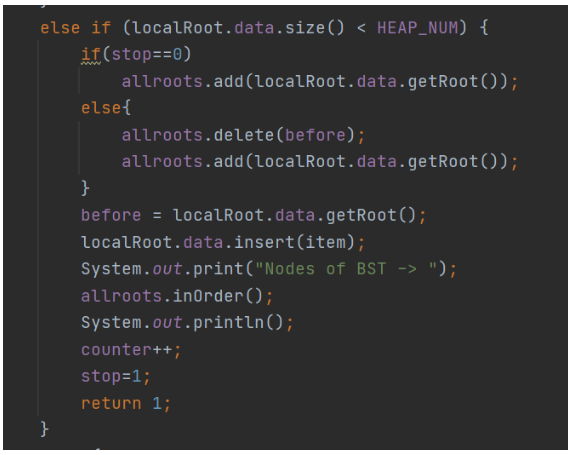
1 parameter add calls 2 parameter add method. localRoot is our key. We can reach bst, node and heap with it. In 2 parameter add method has ‘if - else if - else’ block. If new node should be create:



```java
if (localRoot == null) {
    stop=0;
    numOfNodesBST++;
    System.out.println("\nNumber Of Nodes in BSTHeapNode ->    " + numOfNodesBST);
    MaxHeap<E> hp = new MaxHeap<E>();
    hp.insert(item);
    bst.add(hp);
```

if block runs.

Creates new MaxHeap and inserts the item to be add. Then adds the heap to bst. (This is what homework asks us to do-> MaxHeap in the BinarySearchTree).

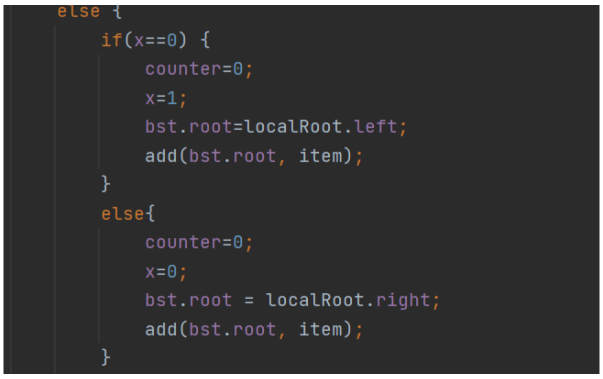


```java
else if (localRoot.data.size() < HEAP_NUM) {
    if(stop==0)
        allroots.add(localRoot.data.getRoot());
    else{
        allroots.delete(before);
        allroots.add(localRoot.data.getRoot());
    }
    before = localRoot.data.getRoot();
    localRoot.data.insert(item);
    System.out.print("Nodes of BST -> ");
    allroots.inOrder();
    System.out.println();
    counter++;
    stop=1;
    return 1;
}
```

If current heap is not full (smaller than 7):

Else if block runs.

We insert all the elements to the current heap until it is full. Nodes Of BST prints the roots of heaps (or nodes of binary search tree).

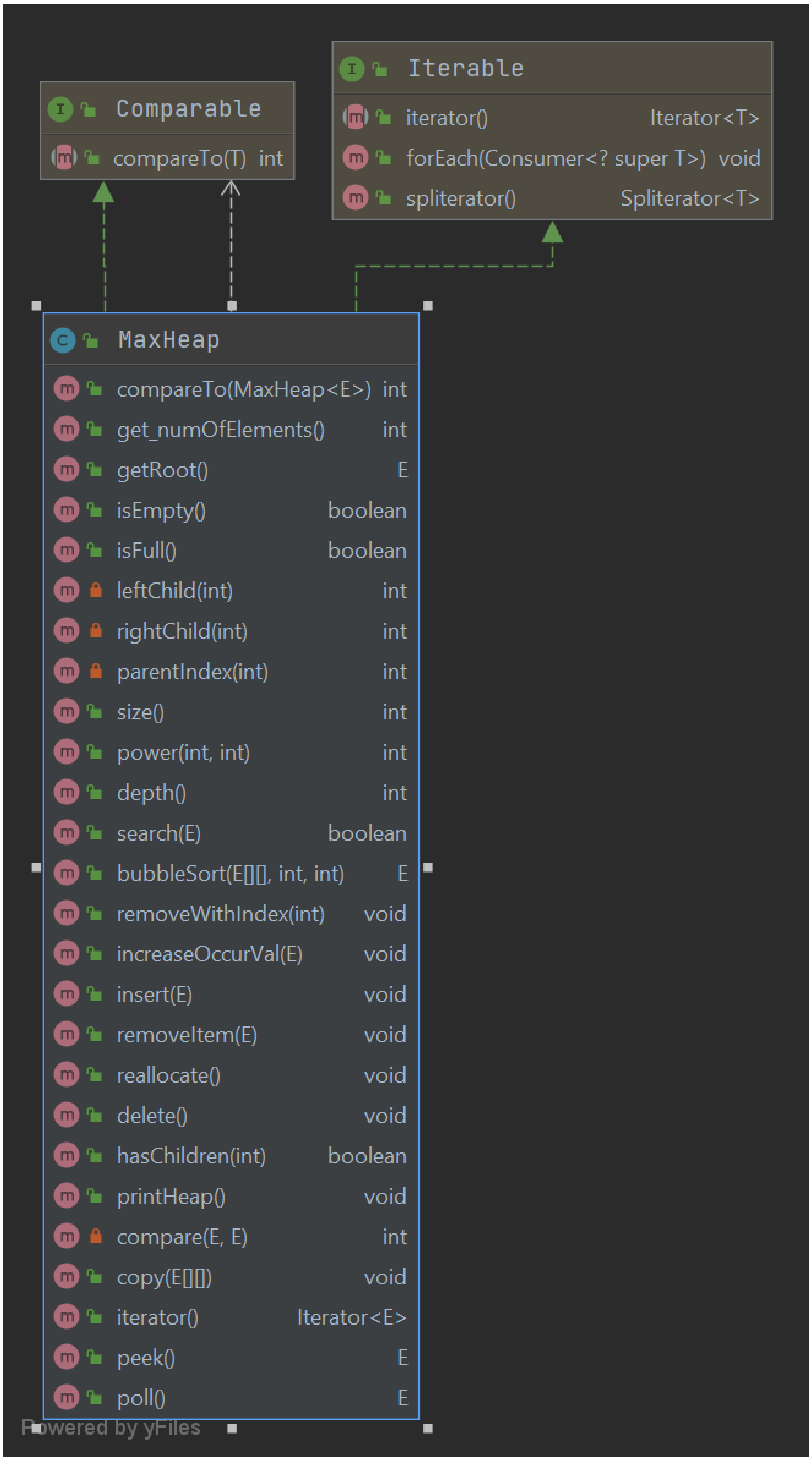


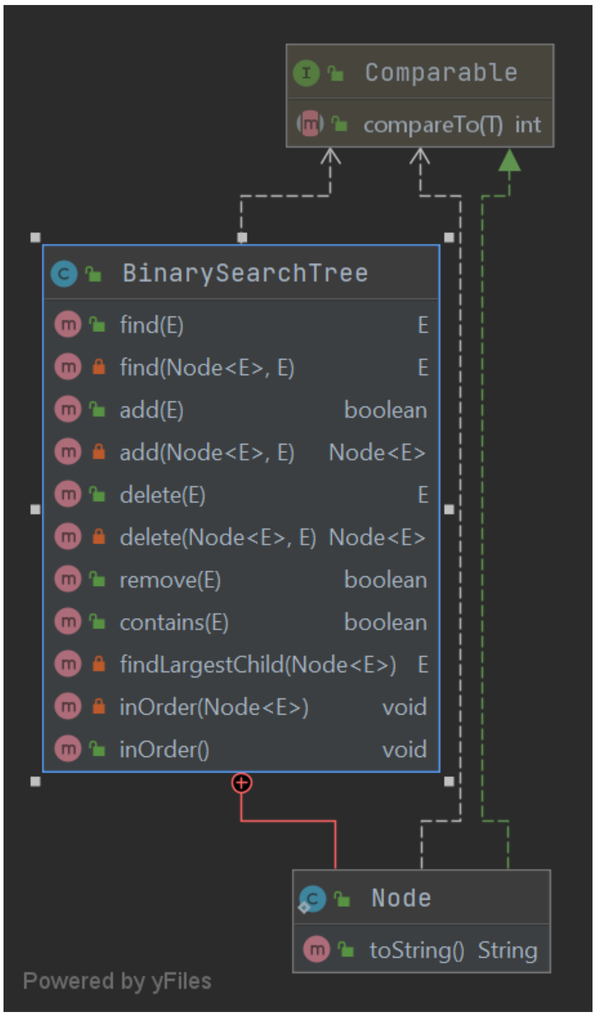
```java
else {
    if(x==0) {
        counter=0;
        x=1;
        bst.root=localRoot.left;
        add(bst.root, item);
    }
    else{
        counter=0;
        x=0;
        bst.root = localRoot.right;
        add(bst.root, item);
    }
```

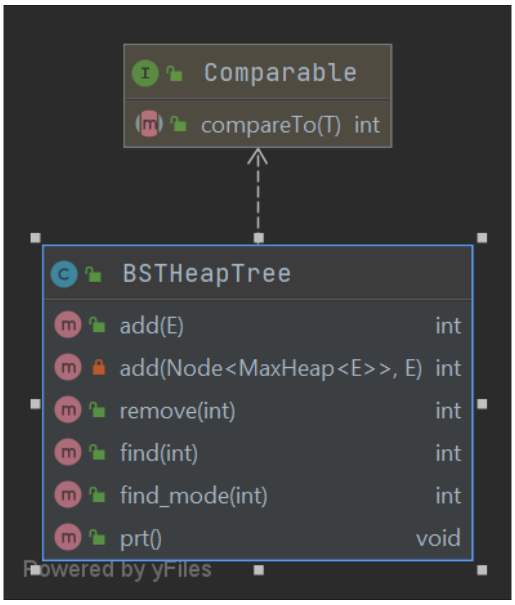
If heap is full:
Else block is run.

It finds which side of the tree is empty to create new bst node (alsa new MaxHeap).

3. Class Diagrams



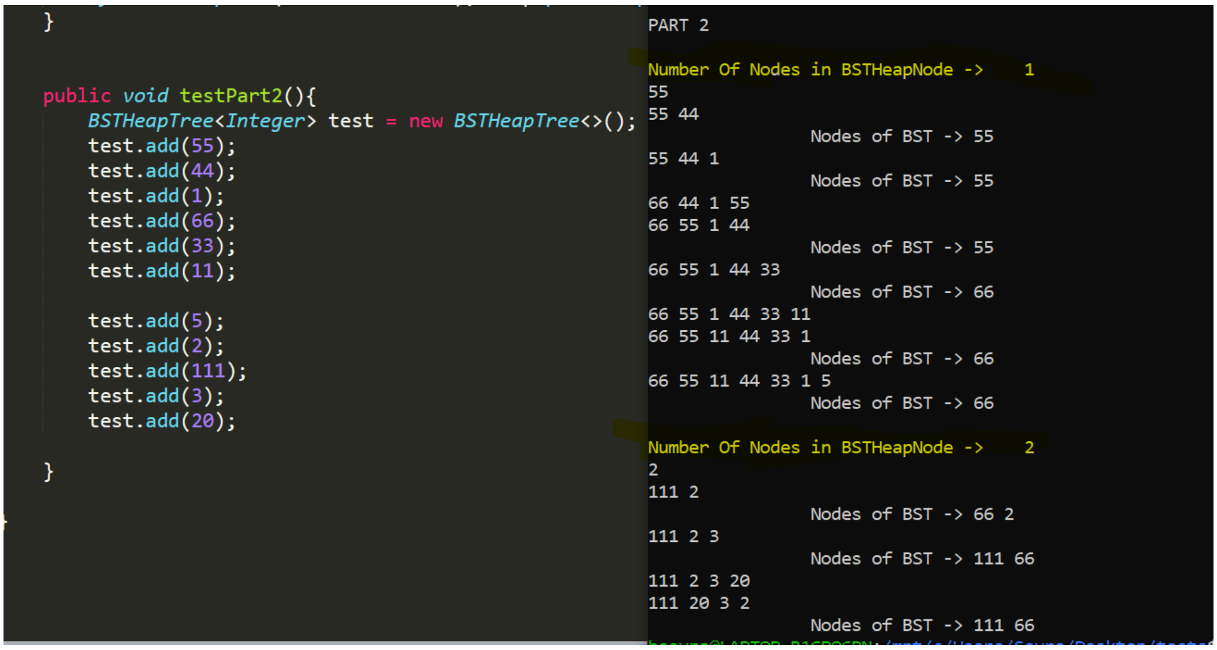

Comparable
compareTo(T) int
BinarySearchTree
find(E) E
find(Node<E>, E) E
add(E) boolean
add(Node<E>, E) Node<E>
delete(E) E
delete(Node<E>, E) Node<E>
remove(E) boolean
contains(E) boolean
findLargestChild(Node<E>) E
inOrder(Node<E>) void
inOrder() void
Node
toString() String
Powered by yFiles


Comparable
compareTo(T) int
BSTHeapTree
add(E) int
add(Node<MaxHeap<E>>, E) int
remove(int) int
find(int) int
find_mode(int) int
prt() void
Powered by yFiles

4. Test Cases and Running Time Results

Program can store

- elements in the maxheap.
- maxheaps in the BinarySearchTree.
- BinarySearchTee in the BSTHeapTree.



```
}


public void testPart2(){
    BSTHeapTree<Integer> test = new BSTHeapTree<>();
    test.add(55);
    test.add(44);
    test.add(1);
    test.add(66);
    test.add(33);
    test.add(11);

    test.add(5);
    test.add(2);
    test.add(111);
    test.add(3);
    test.add(20);

}
```

```
PART 2

Number Of Nodes in BSTHeapNode ->    1
55
55 44
                Nodes of BST -> 55
55 44 1
                Nodes of BST -> 55
66 44 1 55
66 55 1 44
                Nodes of BST -> 55
66 55 1 44 33
                Nodes of BST -> 66
66 55 1 44 33 11
66 55 11 44 33 1
                Nodes of BST -> 66
66 55 11 44 33 1 5
                Nodes of BST -> 66

Number Of Nodes in BSTHeapNode ->    2
2
111 2
                Nodes of BST -> 66 2
111 2 3
                Nodes of BST -> 111 66
111 2 3 20
111 20 3 2
                Nodes of BST -> 111 66
```

Note:
I had no time to complete 2nd part of this project. I am so sorry for that. But I wanted to share my works up to now.
Best Regards.

D) ANALYSIS OF PART2