Gebze Technical University

Computer Engineering

CSE222-2021-SPRING

Homework-4-Report

Part2

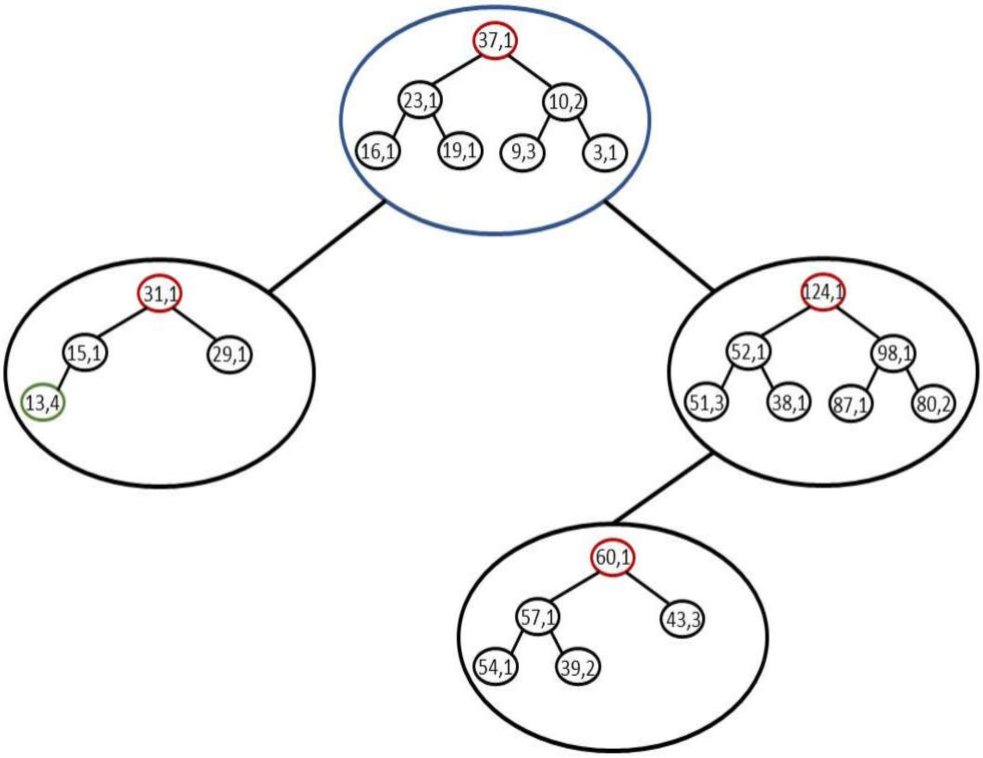
Ferdi Sönmez

161044046

1)INTRODUCTİON

1.1)Problem Definition

We have been asked to make a structure that holds the Binary Search Tree and a MAXHEAP structure in its nodes.



1.2- System Requirements

The necessity of a MAXHEAP structure and a BST structure that includes it emerges from the problem definition. Combining all these components creates a BSTHeapTree structure.

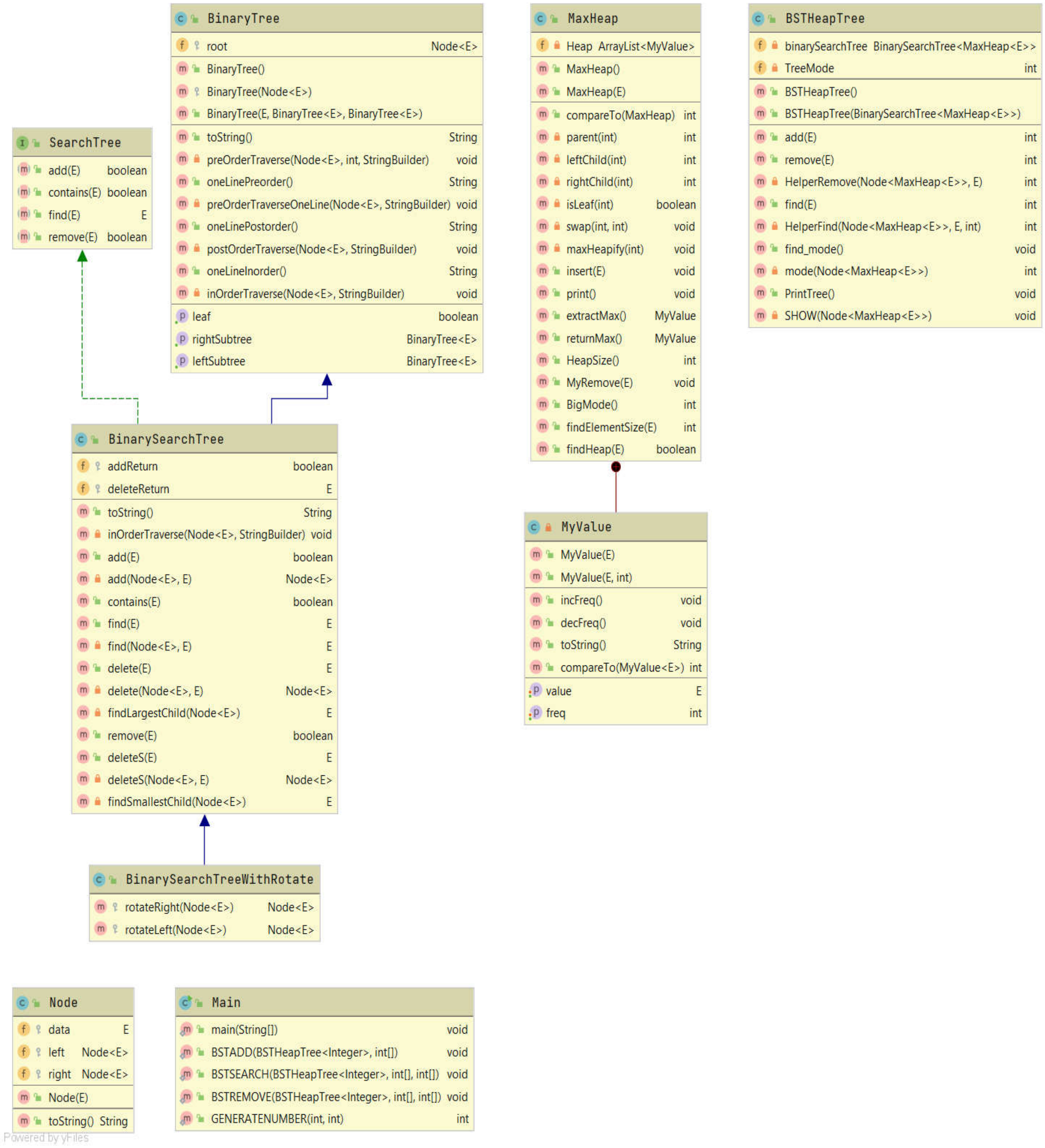
1.2.1-Users of the System

MAXHEAP structure is stored inside the BST structure and all necessary operations are performed using this MAXHEAP structure.

1.2.2-Solution Of The Problem

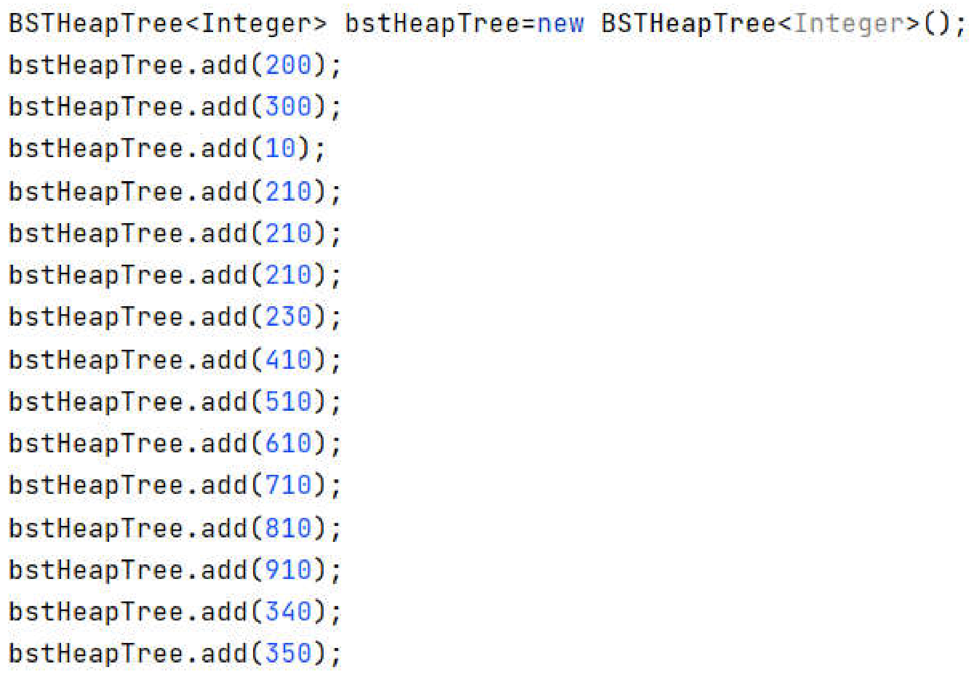
Keeping the MAXHEAP structure in BSTHEAPTREE, the desired functions are solved in this class.

2) Class Diagram



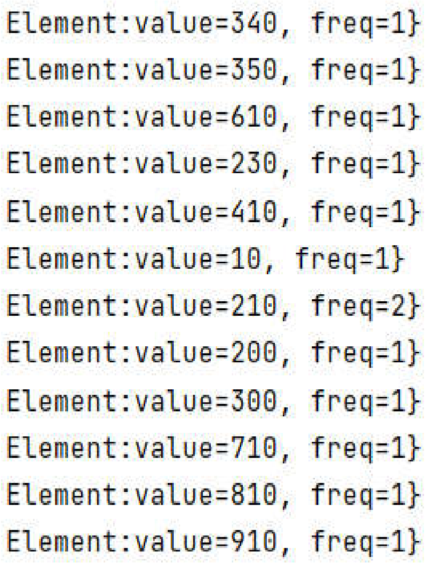
3)Test Case

a) Many numbers have been added to the system.



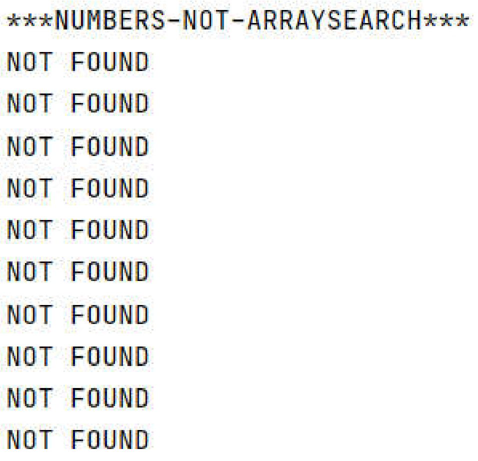
```
BSTHeapTree<Integer> bstHeapTree=new BSTHeapTree<Integer>();
bstHeapTree.add(200);
bstHeapTree.add(300);
bstHeapTree.add(10);
bstHeapTree.add(210);
bstHeapTree.add(210);
bstHeapTree.add(210);
bstHeapTree.add(230);
bstHeapTree.add(410);
bstHeapTree.add(510);
bstHeapTree.add(610);
bstHeapTree.add(710);
bstHeapTree.add(810);
bstHeapTree.add(910);
bstHeapTree.add(340);
bstHeapTree.add(350);
```

Result:



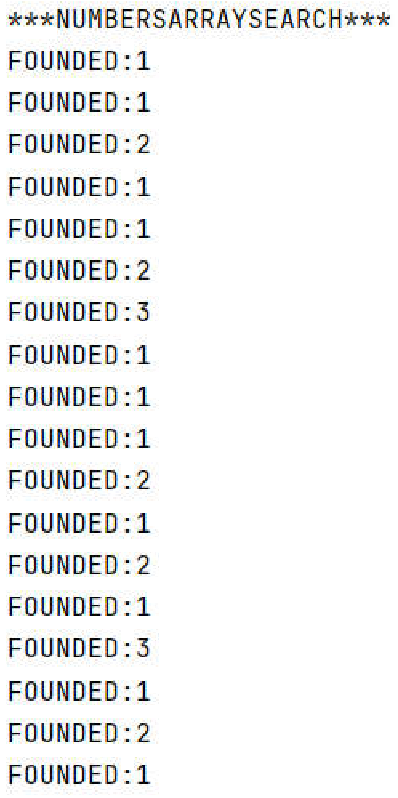
```
Element:value=340, freq=1}
Element:value=350, freq=1}
Element:value=610, freq=1}
Element:value=230, freq=1}
Element:value=410, freq=1}
Element:value=10, freq=1}
Element:value=210, freq=2}
Element:value=200, freq=1}
Element:value=300, freq=1}
Element:value=710, freq=1}
Element:value=810, freq=1}
Element:value=910, freq=1}
```

b) It was attempted to delete the elements that are not in the structure.



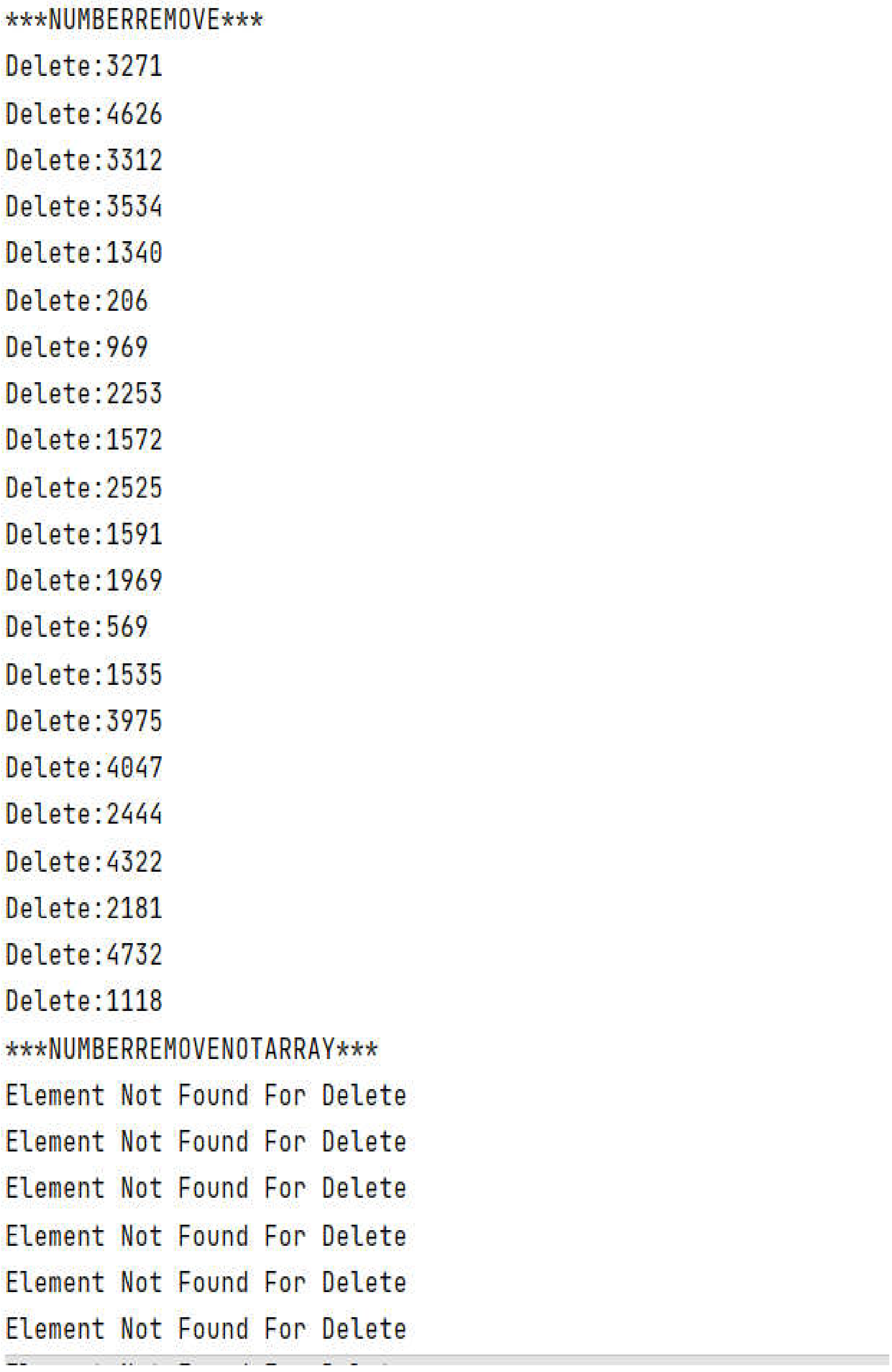
```
***NUMBERS-NOT-ARRAYSEARCH***
NOT FOUND
NOT FOUND
NOT FOUND
NOT FOUND
NOT FOUND
NOT FOUND
NOT FOUND
NOT FOUND
NOT FOUND
NOT FOUND
```

c) An element search has been done in BSTHeapTree.



```
***NUMBERSARRAYSEARCH***
FOUNDED:1
FOUNDED:1
FOUNDED:2
FOUNDED:1
FOUNDED:1
FOUNDED:2
FOUNDED:3
FOUNDED:1
FOUNDED:1
FOUNDED:1
FOUNDED:2
FOUNDED:1
FOUNDED:2
FOUNDED:1
FOUNDED:3
FOUNDED:1
FOUNDED:2
FOUNDED:1
```

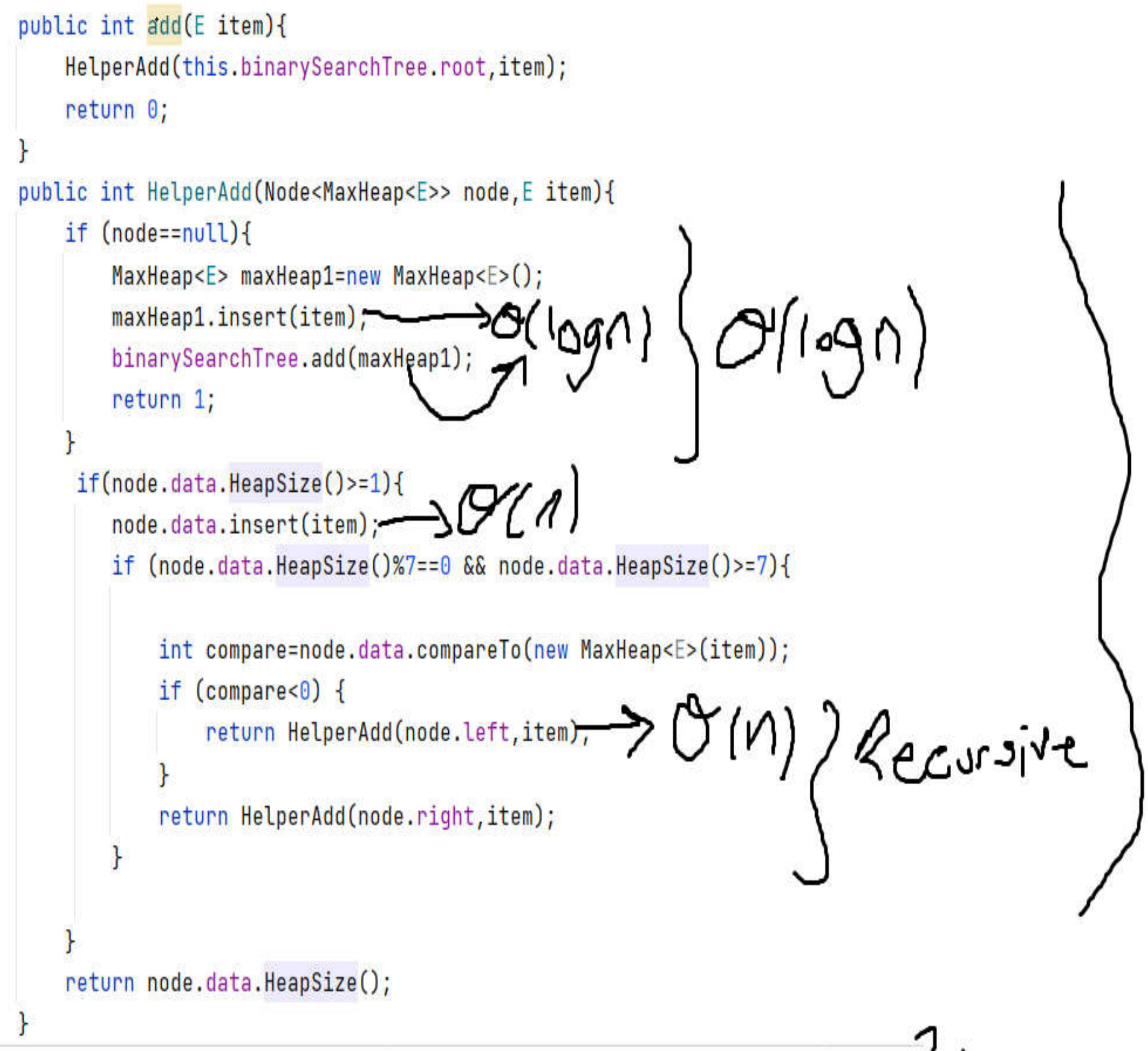
d) Delete any element in the structure



```
***NUMBERREMOVE***
Delete:3271
Delete:4626
Delete:3312
Delete:3534
Delete:1340
Delete:206
Delete:969
Delete:2253
Delete:1572
Delete:2525
Delete:1591
Delete:1969
Delete:569
Delete:1535
Delete:3975
Delete:4047
Delete:2444
Delete:4322
Delete:2181
Delete:4732
Delete:1118
***NUMBERREMOVENOTARRAY***
Element Not Found For Delete
Element Not Found For Delete
Element Not Found For Delete
Element Not Found For Delete
Element Not Found For Delete
Element Not Found For Delete
```

Time Complexity:

1)



```
public int add(E item){
    HelperAdd(this.binarySearchTree.root,item);
    return 0;
}
public int HelperAdd(Node<MaxHeap<E>> node,E item){
    if (node==null){
        MaxHeap<E> maxHeap1=new MaxHeap<E>();
        maxHeap1.insert(item);
        binarySearchTree.add(maxHeap1);
        return 1;
    }
     if(node.data.HeapSize()>=1){
        node.data.insert(item);
        if (node.data.HeapSize()%7==0 && node.data.HeapSize()>=7){

            int compare=node.data.compareTo(new MaxHeap<E>(item));
            if (compare<0) {
                return HelperAdd(node.left,item);
            }
            return HelperAdd(node.right,item);
        }

    }
    return node.data.HeapSize();
}
```

$\to O(\log n)$

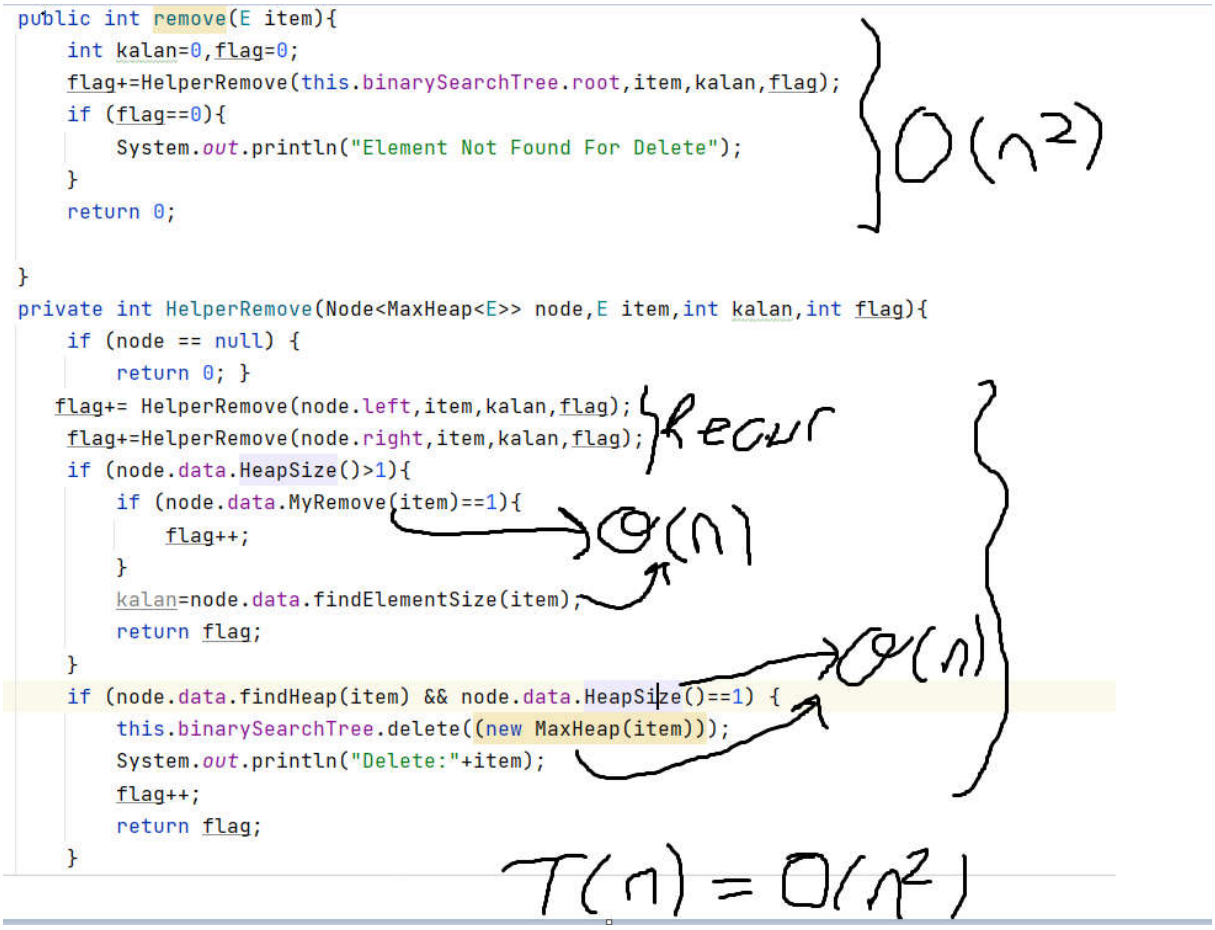
$\} O(\log n)$

$\to O(n)$

$\to O(n) \}$ Recursive

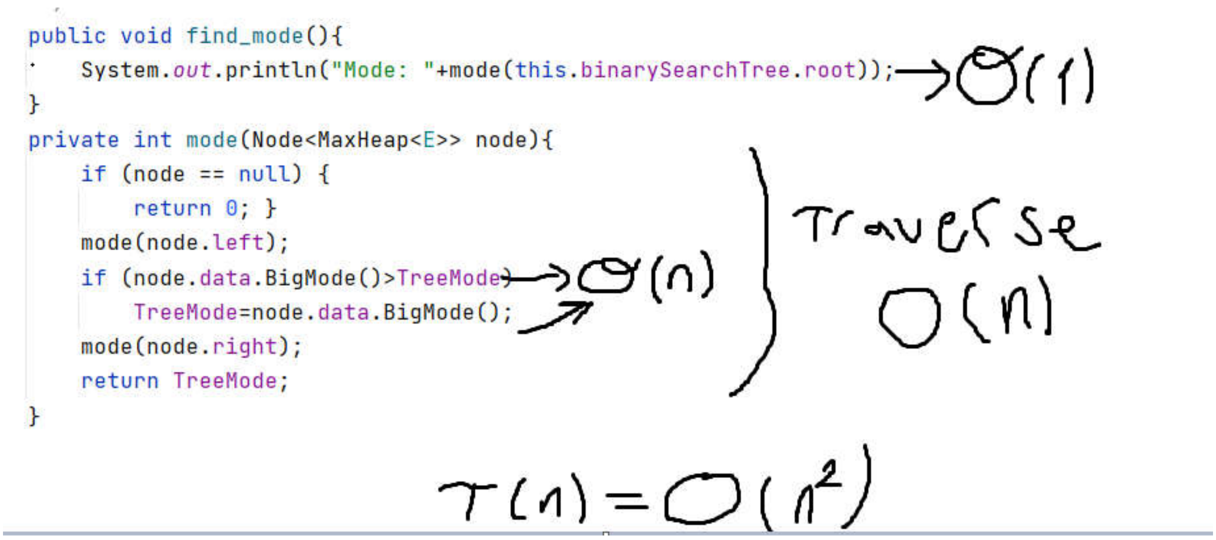
$$T(n)=O(n^2)$$

2)



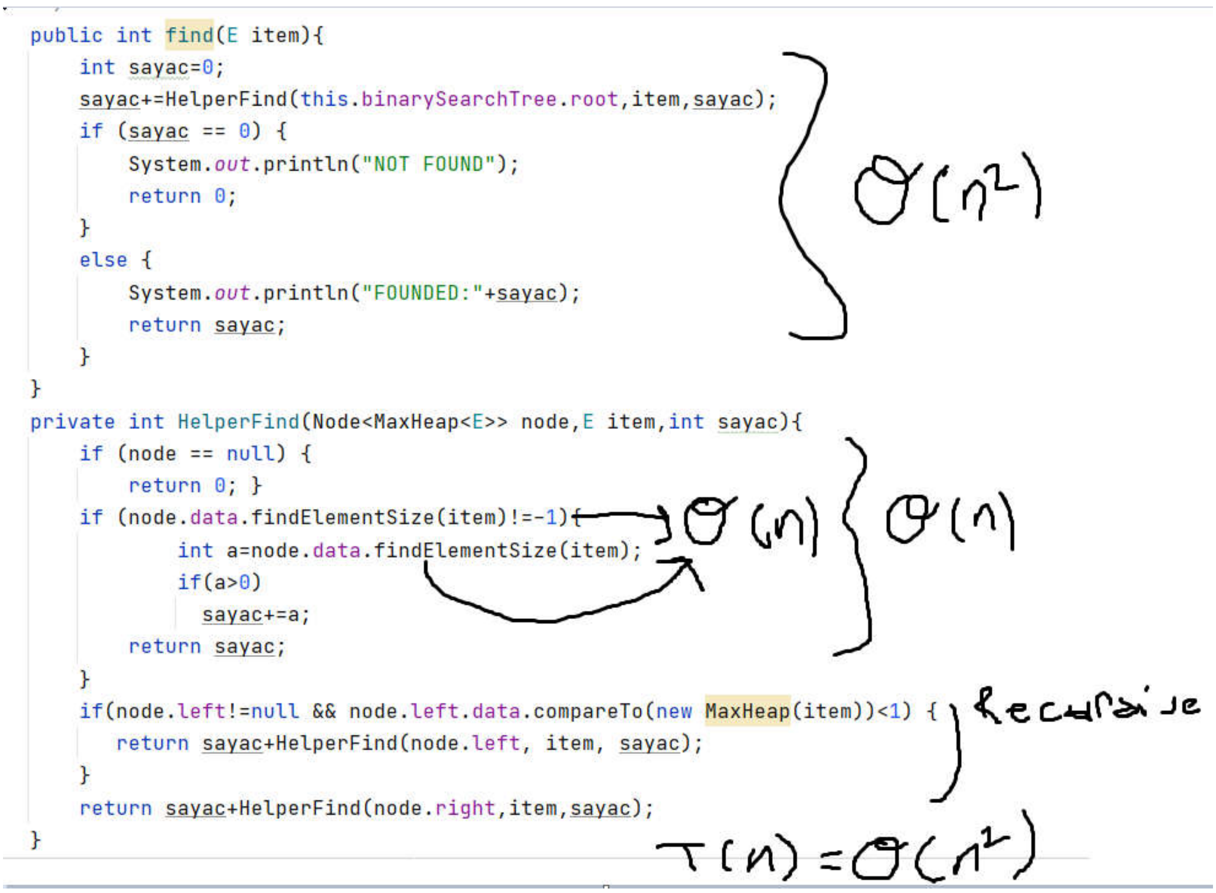
```
public int remove(E item){
    int kalan=0,flag=0;
    flag+=HelperRemove(this.binarySearchTree.root,item,kalan,flag);
    if (flag==0){
        System.out.println("Element Not Found For Delete");
    }
    return 0;

}
private int HelperRemove(Node<MaxHeap<E>> node,E item,int kalan,int flag){
    if (node == null) {
        return 0; }
   flag+= HelperRemove(node.left,item,kalan,flag);
    flag+=HelperRemove(node.right,item,kalan,flag);
    if (node.data.HeapSize()>1){
        if (node.data.MyRemove(item)==1){
            flag++;
        }
        kalan=node.data.findElementSize(item);
        return flag;
    }
    if (node.data.findHeap(item) && node.data.HeapSize()==1) {
        this.binarySearchTree.delete((new MaxHeap(item)));
        System.out.println("Delete:"+item);
        flag++;
        return flag;
    }
```

3)



```
public void find_mode(){
    System.out.println("Mode: "+mode(this.binarySearchTree.root));
}
private int mode(Node<MaxHeap<E>> node){
    if (node == null) {
        return 0; }
    mode(node.left);
    if (node.data.BigMode()>TreeMode)
        TreeMode=node.data.BigMode();
    mode(node.right);
    return TreeMode;
}
```

4)



```java
public int find(E item){
    int sayac=0;
    sayac+=HelperFind(this.binarySearchTree.root,item,sayac);
    if (sayac == 0) {
        System.out.println("NOT FOUND");
        return 0;
    }
    else {
        System.out.println("FOUNDED:"+sayac);
        return sayac;
    }
}
private int HelperFind(Node<MaxHeap<E>> node,E item,int sayac){
    if (node == null) {
        return 0; }
    if (node.data.findElementSize(item)!=-1){
            int a=node.data.findElementSize(item);
            if(a>0)
              sayac+=a;
        return sayac;
    }
    if(node.left!=null && node.left.data.compareTo(new MaxHeap(item))<1) {
       return sayac+HelperFind(node.left, item, sayac);
    }
    return sayac+HelperFind(node.right,item,sayac);
}
```

$\theta(n^2)$

$\theta(n)$

$\theta(n)$

Recursive

$T(n)=\theta(n^2)$