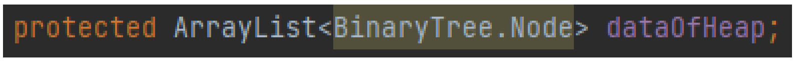
GTU Department of Computer Engineering
CSE 222/505 - SPRING 2022
HOMEWORK 5 REPORT

BURAK ÇİÇEK
1901042260

System Requirements

For Q3:

In this question I implemented a Ternary Heap as question said. I'm keeping that all Nodes in a Nodes ArrayList.



```
protected ArrayList<BinaryTree.Node> dataOfHeap;
```

There is a getters constructors as usual but I won't add them to this report

```
public boolean mergeHeaps(TernaryHeap<E> input)
public boolean incrementKeyValue(int index, E newValue)
public boolean removeEelement(BinaryTree.Node item)
```

Remove of given element from heap also Node's binary arrayList. Create a new Ternary Heap object and add all of items from this current Node ArrayList to sort again after that copy elements of copy's Ternary Heaps to current Heap.

```
public boolean incrementKeyValue(int index, E newValue)
```

Find the given index's item and increment the Key value according to given value.

```
public void add( BinaryTree.Node item )
```

Add a new node to ArrayList than sort using heapify Up Method.

```
protected void heapifyUp()
```

HeapifyUp method for sorting after add.

```
protected Node < E > left;
protected Node < E > middle;
protected Node < E > right;
```

For Q4:

```
public E find(E target)
```

Finding method to given target.

```
public boolean contains(E target)
```

is It contains or not?

```
public E delete(E target)
```

delete method to target object if couldn't find return null

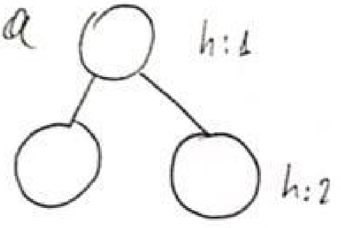
```
public boolean remove(E target)
```

remove method for given target parameter.

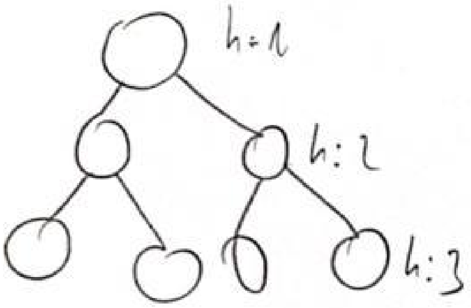
Problem Solution Approach

Frankly, the biggest problem in doing this assignment was common to both classes. When we remove a node, sorting the remaining structure again was the trickiest part of this assignment, I think. I had some difficulty in this matter, but then I found a solution that was not very effective but would work for me. I created a different object to add the remaining elements to the Binary Search Tree or Heap that I just created. After adding my elements to the new object, all my elements were sorted due to the add algorithm of that object. Then I took back those sorted elements in the same order. Thus, I was able to do almost all the homework with a single algorithm without having to design a new algorithm. Of course, this was not a very good solution, but I can say that I was happy when I found this design because I had time constraints and I had to finish the homework.

1-) a



depth -> $2.2+1=5$



depth -> $3.4+2.2+1=17$

Depth of h height binary tree.

$D(h) = 2^{h-1} \cdot h + D(h-1)$

$D(1) = \underline{1}$

, or , $h \cdot 2^{h-1} + h-1 \cdot 2^{h-2} + \ldots$

b-) Av. Comparison Count : Total Comparison count / Total Node Number.

or

words on the other $N(X)$ is location of the X of the BST than px is selectability probability.

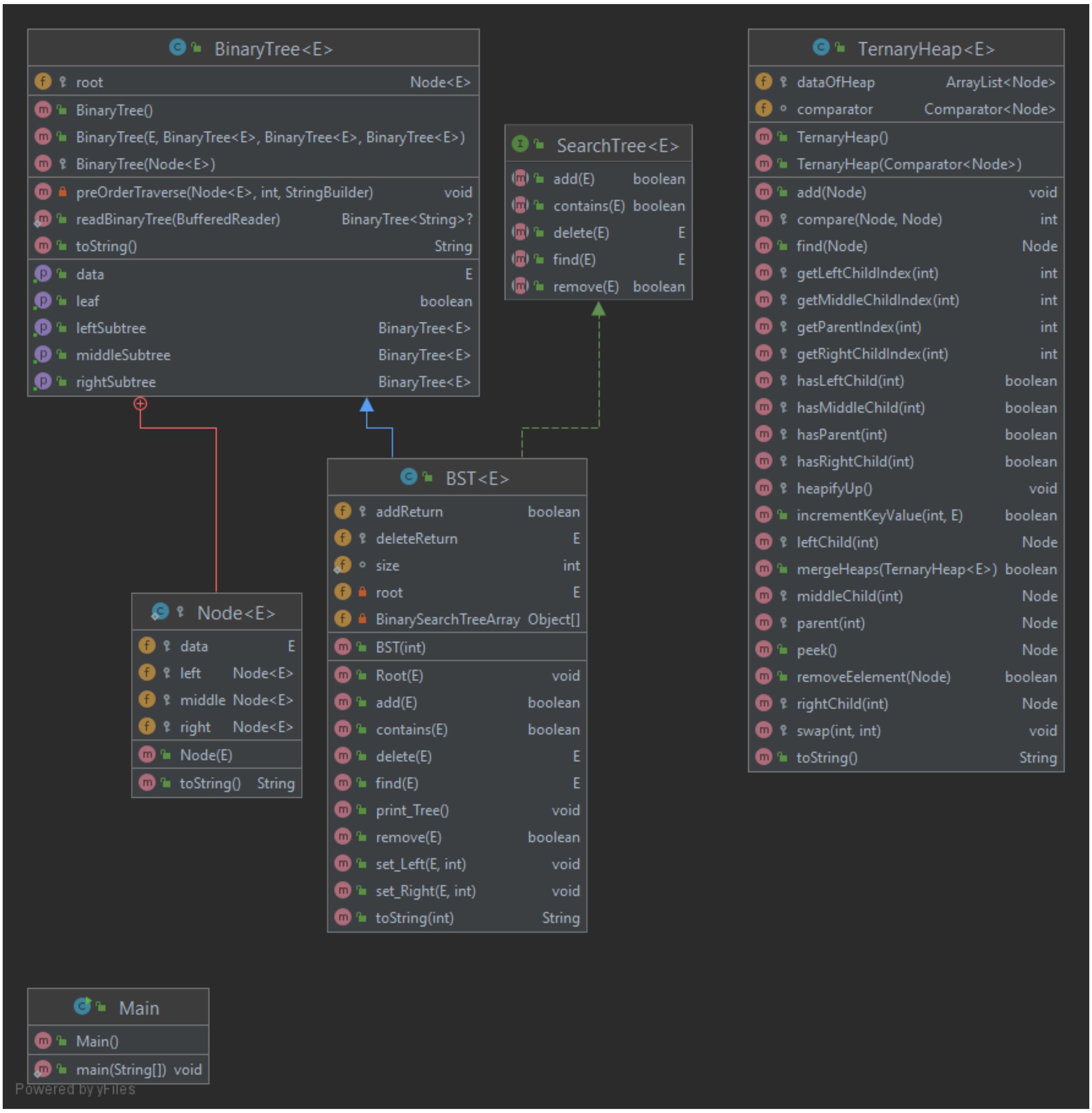
$$\sum_{\text{Iterations}} px \cdot C(x) \quad / \text{ All selectability probability}$$

C-) No, there is no restriction/(s) in there

N : total nodes, the number of internal node is $I = (N-1)/2$

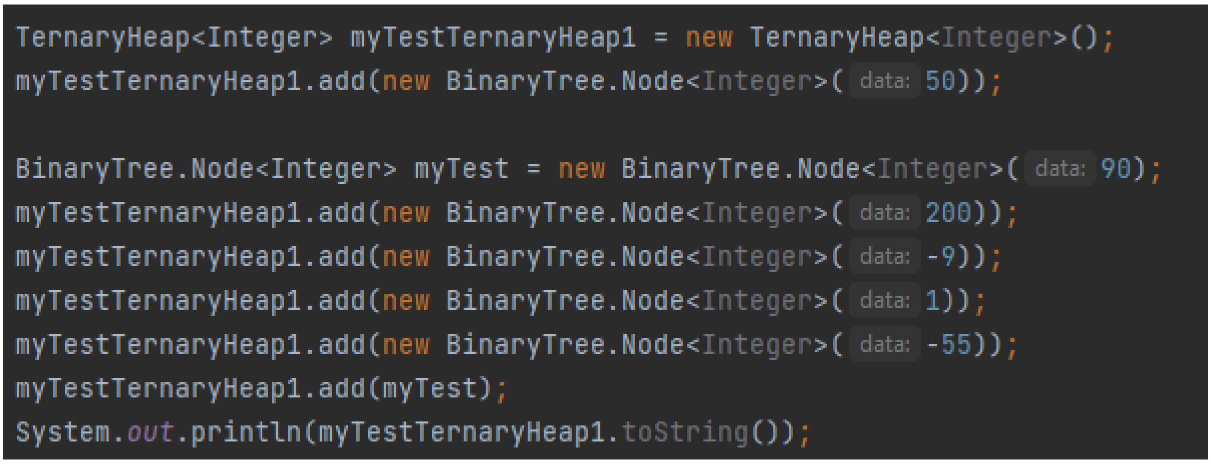
the number of leaves is $L = (N+1)/2$.

Class Diagrams

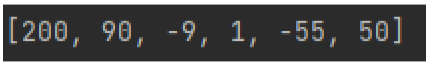


Test Cases

Q3 -

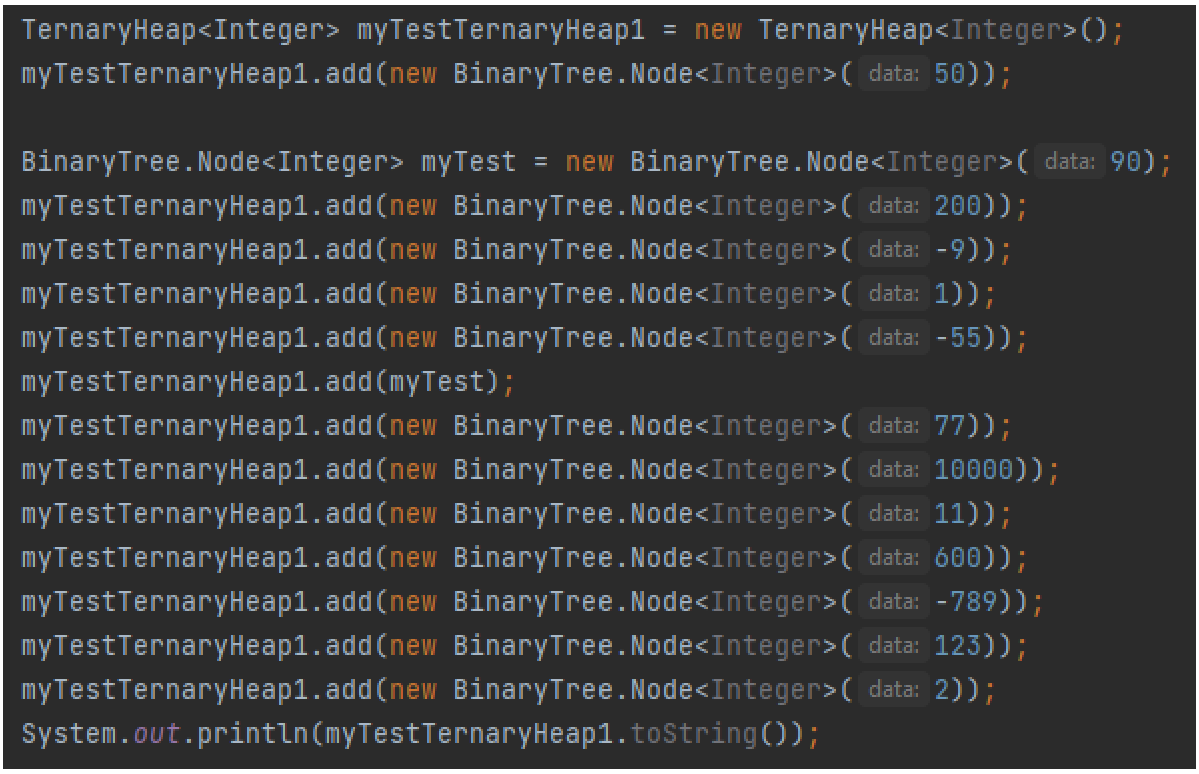


```
TernaryHeap<Integer> myTestTernaryHeap1 = new TernaryHeap<Integer>();
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: 50));

BinaryTree.Node<Integer> myTest = new BinaryTree.Node<Integer>( data: 90);
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: 200));
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: -9));
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: 1));
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: -55));
myTestTernaryHeap1.add(myTest);
System.out.println(myTestTernaryHeap1.toString());
```

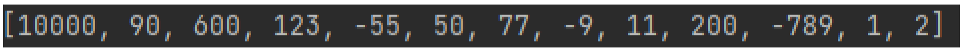


```
[200, 90, -9, 1, -55, 50]
```

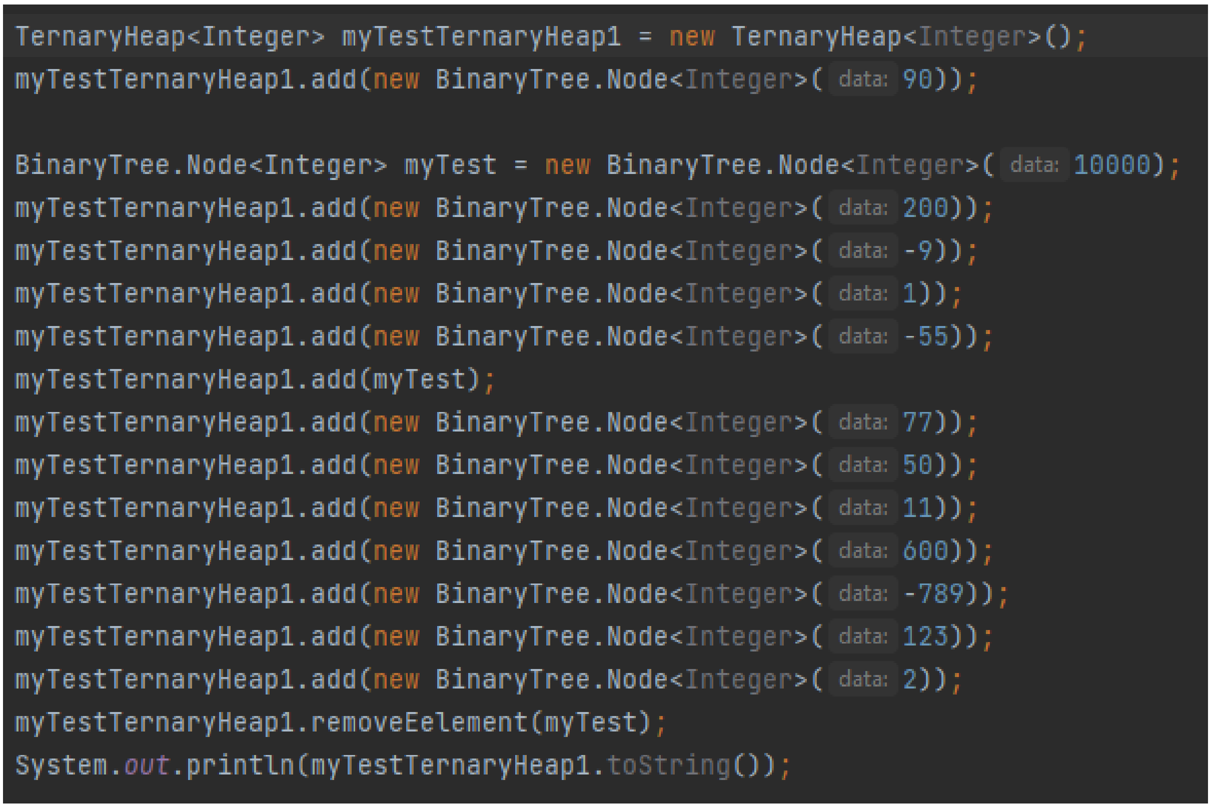
ORDER:
(root-leftchild-middlechild-rightchild-left’sleft-left’smiddle-left’sright)



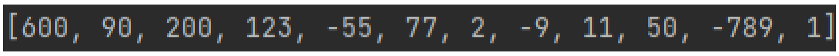
```
TernaryHeap<Integer> myTestTernaryHeap1 = new TernaryHeap<Integer>();
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: 50));

BinaryTree.Node<Integer> myTest = new BinaryTree.Node<Integer>( data: 90);
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: 200));
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: -9));
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: 1));
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: -55));
myTestTernaryHeap1.add(myTest);
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: 77));
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: 10000));
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: 11));
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: 600));
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: -789));
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: 123));
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: 2));
System.out.println(myTestTernaryHeap1.toString());
```



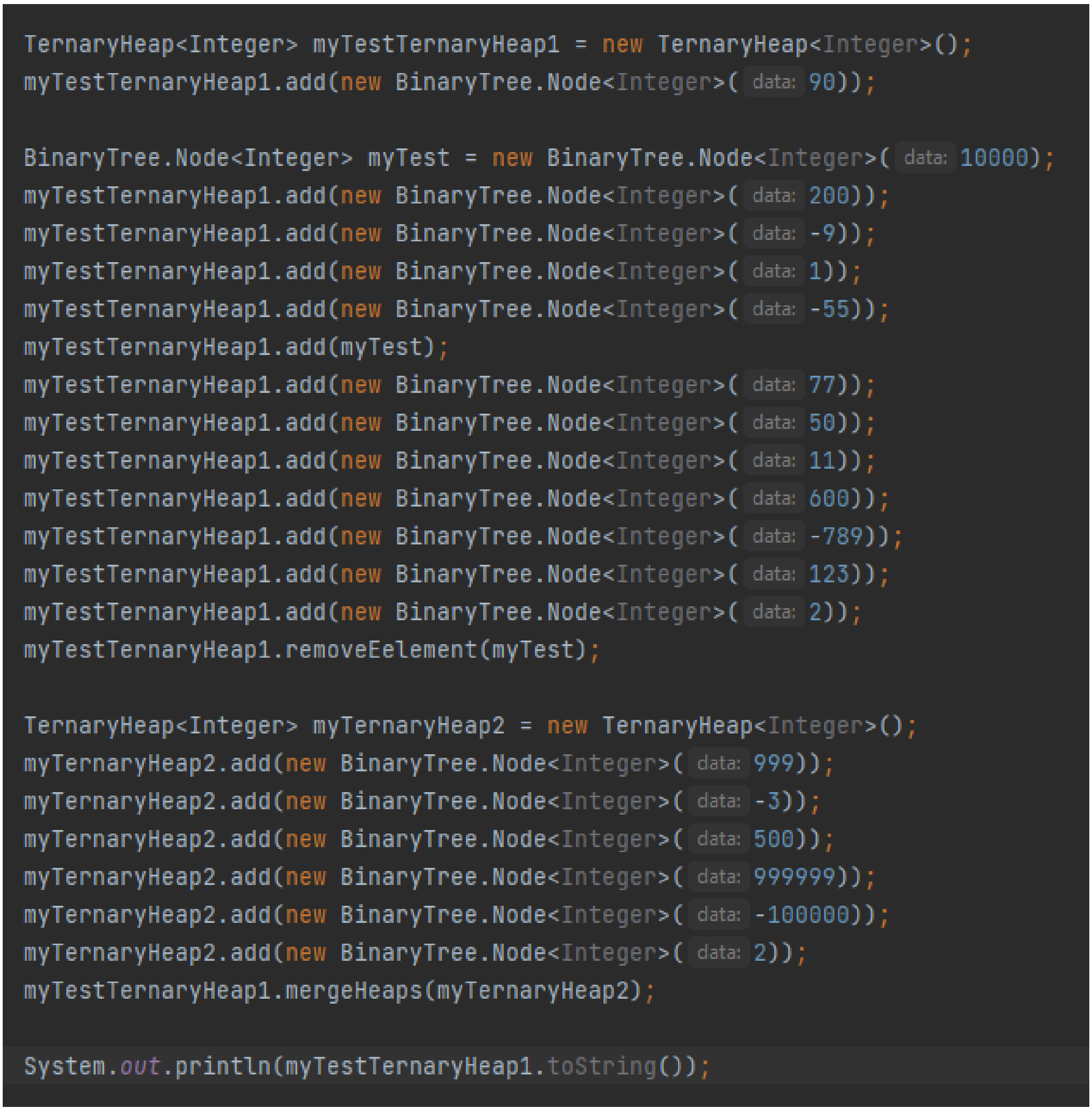
```
[10000, 90, 600, 123, -55, 50, 77, -9, 11, 200, -789, 1, 2]
```



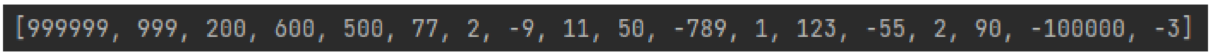
```
TernaryHeap<Integer> myTestTernaryHeap1 = new TernaryHeap<Integer>();
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: 90));

BinaryTree.Node<Integer> myTest = new BinaryTree.Node<Integer>( data: 10000);
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: 200));
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: -9));
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: 1));
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: -55));
myTestTernaryHeap1.add(myTest);
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: 77));
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: 50));
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: 11));
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: 600));
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: -789));
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: 123));
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: 2));
myTestTernaryHeap1.removeEelement(myTest);
System.out.println(myTestTernaryHeap1.toString());
```



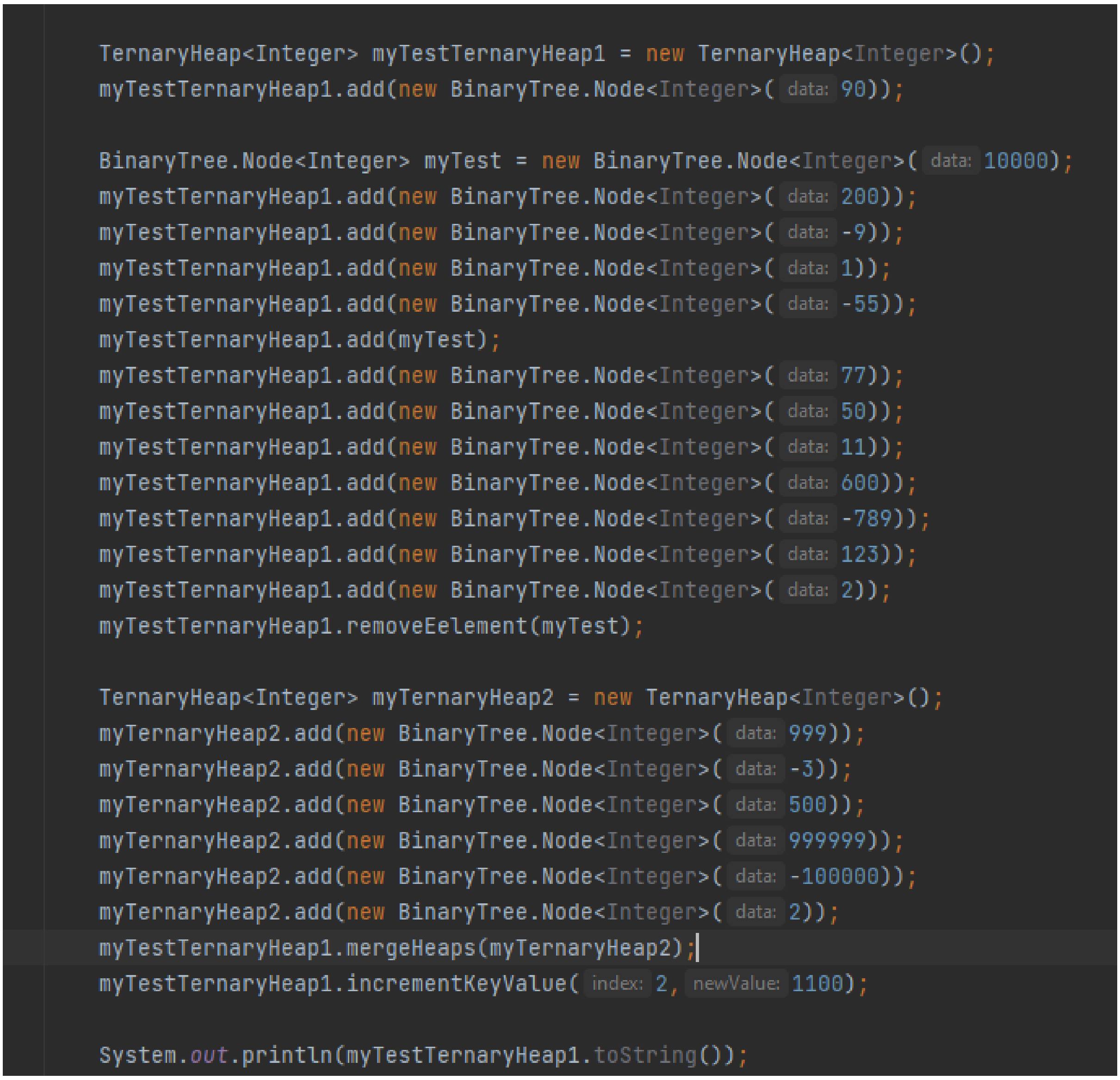
```
[600, 90, 200, 123, -55, 77, 2, -9, 11, 50, -789, 1]
```



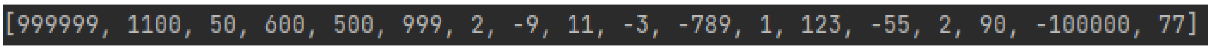
```
TernaryHeap<Integer> myTestTernaryHeap1 = new TernaryHeap<Integer>();
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: 90));

BinaryTree.Node<Integer> myTest = new BinaryTree.Node<Integer>( data: 10000);
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: 200));
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: -9));
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: 1));
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: -55));
myTestTernaryHeap1.add(myTest);
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: 77));
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: 50));
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: 11));
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: 600));
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: -789));
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: 123));
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: 2));
myTestTernaryHeap1.removeEelement(myTest);

TernaryHeap<Integer> myTernaryHeap2 = new TernaryHeap<Integer>();
myTernaryHeap2.add(new BinaryTree.Node<Integer>( data: 999));
myTernaryHeap2.add(new BinaryTree.Node<Integer>( data: -3));
myTernaryHeap2.add(new BinaryTree.Node<Integer>( data: 500));
myTernaryHeap2.add(new BinaryTree.Node<Integer>( data: 999999));
myTernaryHeap2.add(new BinaryTree.Node<Integer>( data: -100000));
myTernaryHeap2.add(new BinaryTree.Node<Integer>( data: 2));
myTestTernaryHeap1.mergeHeaps(myTernaryHeap2);

System.out.println(myTestTernaryHeap1.toString());
```



```
[999999, 999, 200, 600, 500, 77, 2, -9, 11, 50, -789, 1, 123, -55, 2, 90, -100000, -3]
```

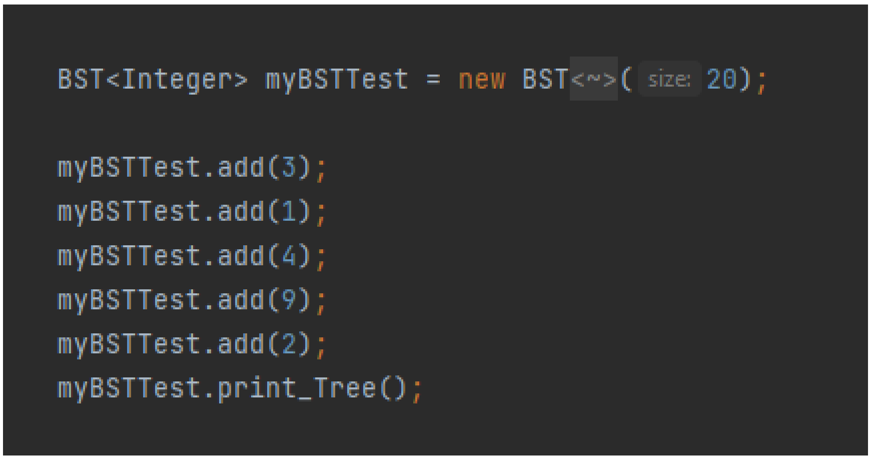


```
TernaryHeap<Integer> myTestTernaryHeap1 = new TernaryHeap<Integer>();
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: 90));

BinaryTree.Node<Integer> myTest = new BinaryTree.Node<Integer>( data: 10000);
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: 200));
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: -9));
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: 1));
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: -55));
myTestTernaryHeap1.add(myTest);
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: 77));
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: 50));
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: 11));
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: 600));
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: -789));
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: 123));
myTestTernaryHeap1.add(new BinaryTree.Node<Integer>( data: 2));
myTestTernaryHeap1.removeEelement(myTest);

TernaryHeap<Integer> myTernaryHeap2 = new TernaryHeap<Integer>();
myTernaryHeap2.add(new BinaryTree.Node<Integer>( data: 999));
myTernaryHeap2.add(new BinaryTree.Node<Integer>( data: -3));
myTernaryHeap2.add(new BinaryTree.Node<Integer>( data: 500));
myTernaryHeap2.add(new BinaryTree.Node<Integer>( data: 999999));
myTernaryHeap2.add(new BinaryTree.Node<Integer>( data: -100000));
myTernaryHeap2.add(new BinaryTree.Node<Integer>( data: 2));
myTestTernaryHeap1.mergeHeaps(myTernaryHeap2);
myTestTernaryHeap1.incrementKeyValue( index: 2, newValue: 1100);

System.out.println(myTestTernaryHeap1.toString());
```

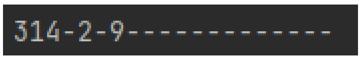


```
[999999, 1100, 50, 600, 500, 999, 2, -9, 11, -3, -789, 1, 123, -55, 2, 90, -100000, 77]
```

Q4 -

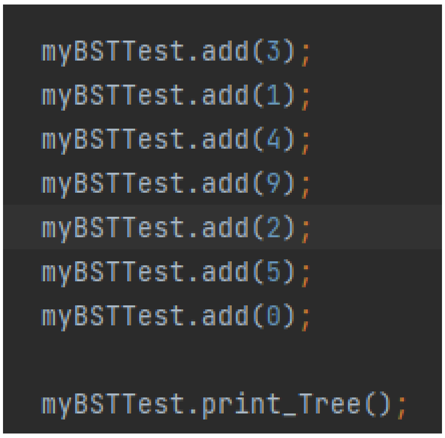


```
BST<Integer> myBSTTest = new BST<~>( size: 20);

myBSTTest.add(3);
myBSTTest.add(1);
myBSTTest.add(4);
myBSTTest.add(9);
myBSTTest.add(2);
myBSTTest.print_Tree();
```

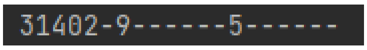


```
314-2-9-------------
```

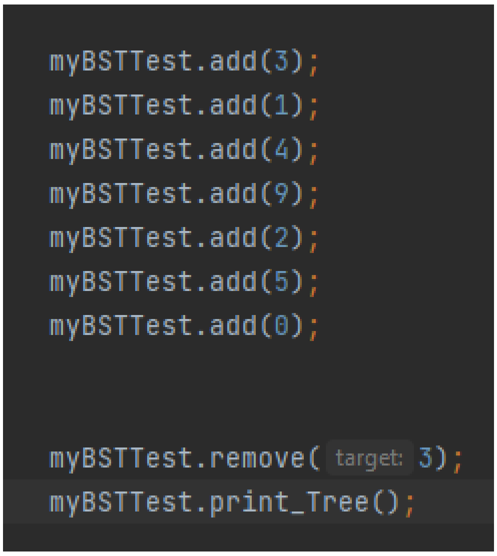
root-leftchild-rightchild-left’sleftchild-left’strightchilt-right’sleftchild-right’srightchild



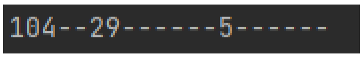
```
myBSTTest.add(3);
myBSTTest.add(1);
myBSTTest.add(4);
myBSTTest.add(9);
myBSTTest.add(2);
myBSTTest.add(5);
myBSTTest.add(0);

myBSTTest.print_Tree();
```



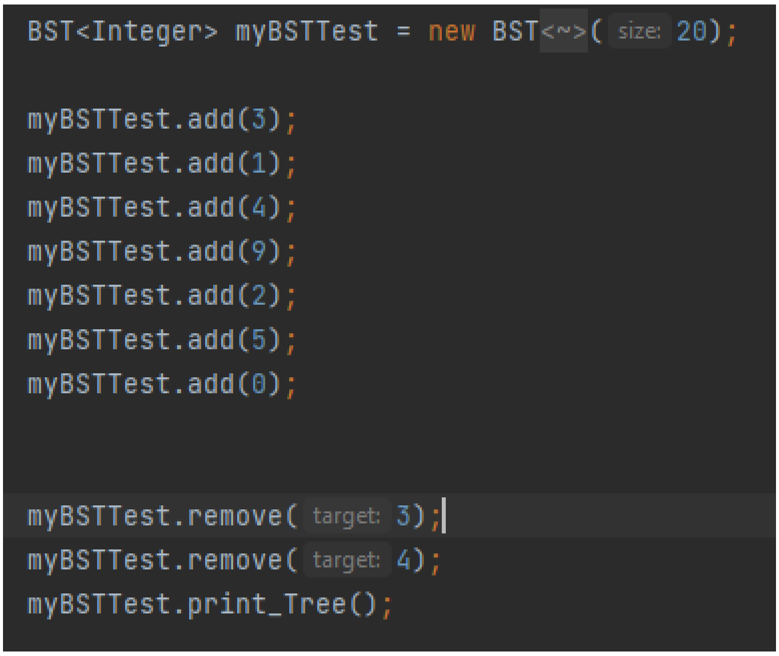
```
31402-9------5------
```



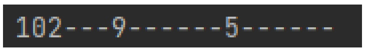
```
myBSTTest.add(3);
myBSTTest.add(1);
myBSTTest.add(4);
myBSTTest.add(9);
myBSTTest.add(2);
myBSTTest.add(5);
myBSTTest.add(0);


myBSTTest.remove( target: 3);
myBSTTest.print_Tree();
```



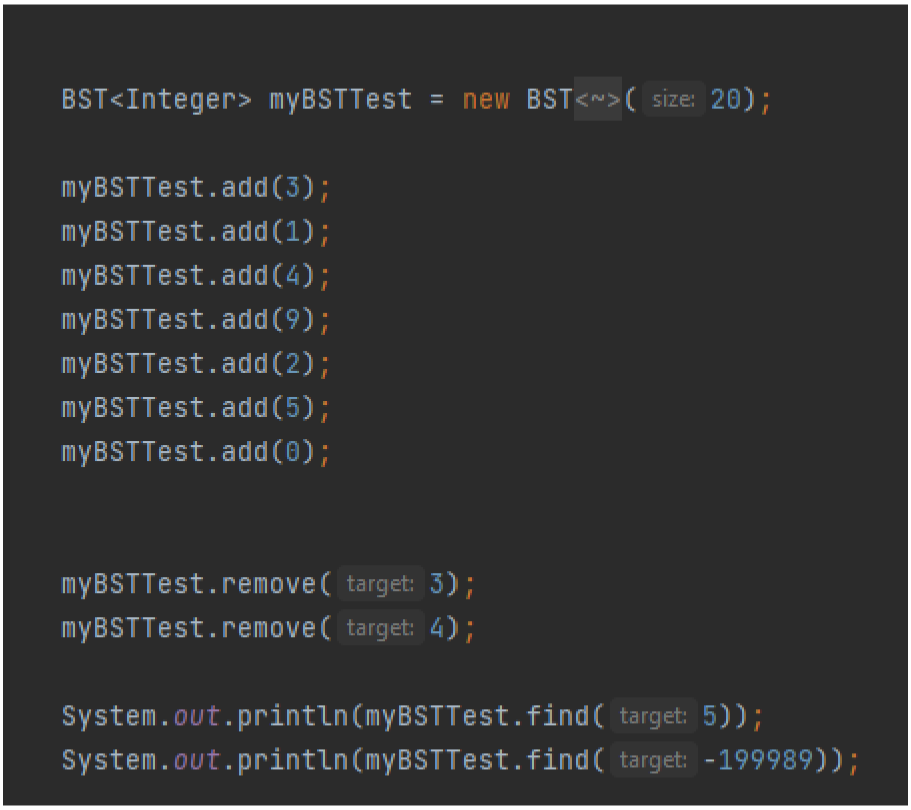
```
104--29------5------
```



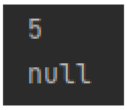
```
BST<Integer> myBSTTest = new BST<~>( size: 20);

myBSTTest.add(3);
myBSTTest.add(1);
myBSTTest.add(4);
myBSTTest.add(9);
myBSTTest.add(2);
myBSTTest.add(5);
myBSTTest.add(0);


myBSTTest.remove( target: 3);
myBSTTest.remove( target: 4);
myBSTTest.print_Tree();
```



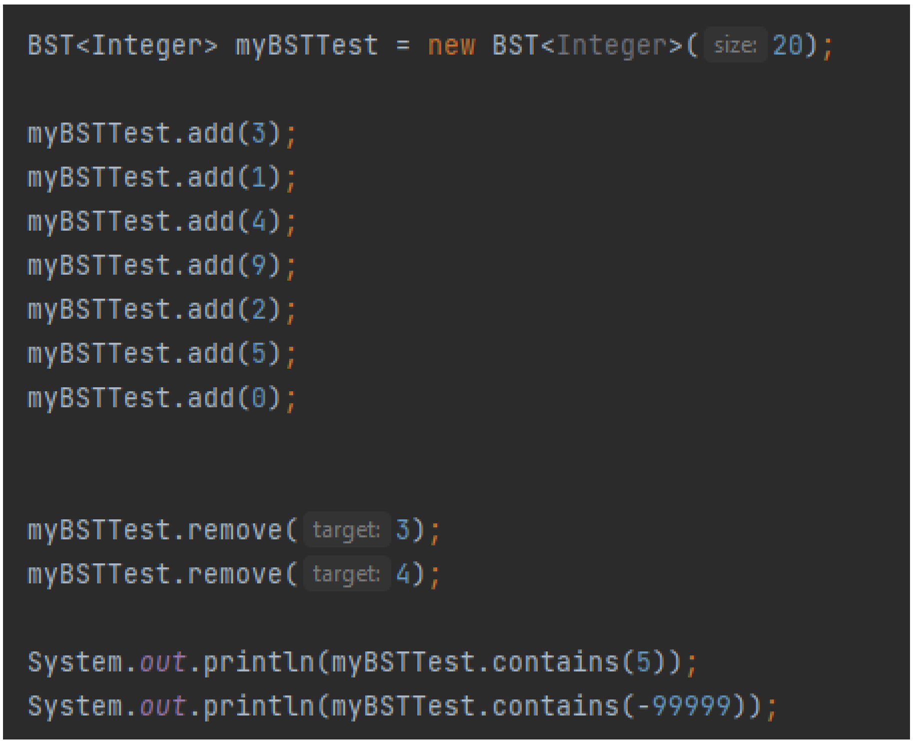
```
102---9------5------
```



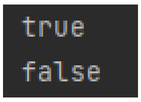
```
BST<Integer> myBSTTest = new BST<~>( size: 20);

myBSTTest.add(3);
myBSTTest.add(1);
myBSTTest.add(4);
myBSTTest.add(9);
myBSTTest.add(2);
myBSTTest.add(5);
myBSTTest.add(0);


myBSTTest.remove( target: 3);
myBSTTest.remove( target: 4);

System.out.println(myBSTTest.find( target: 5));
System.out.println(myBSTTest.find( target: -199989));
```



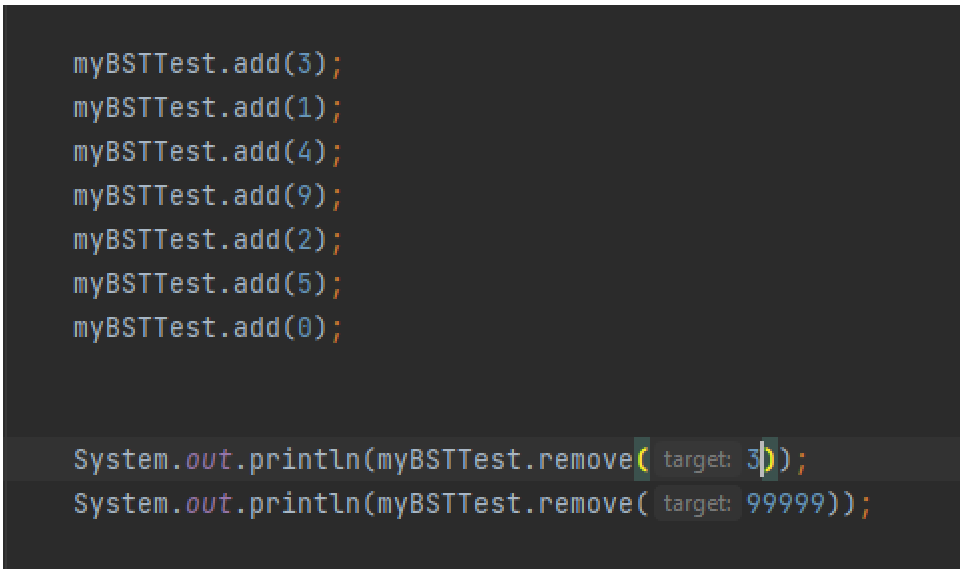
```
5
null
```



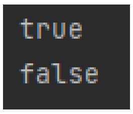
```
BST<Integer> myBSTTest = new BST<Integer>( size: 20);

myBSTTest.add(3);
myBSTTest.add(1);
myBSTTest.add(4);
myBSTTest.add(9);
myBSTTest.add(2);
myBSTTest.add(5);
myBSTTest.add(0);


myBSTTest.remove( target: 3);
myBSTTest.remove( target: 4);

System.out.println(myBSTTest.contains(5));
System.out.println(myBSTTest.contains(-99999));
```



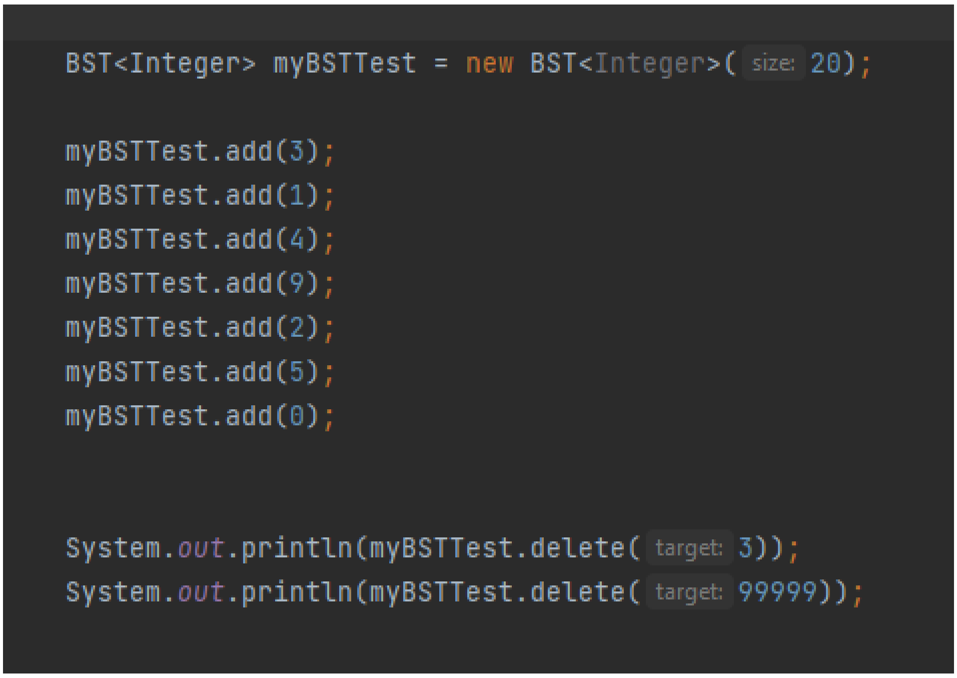
```
true
false
```



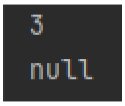
```
    myBSTTest.add(3);
    myBSTTest.add(1);
    myBSTTest.add(4);
    myBSTTest.add(9);
    myBSTTest.add(2);
    myBSTTest.add(5);
    myBSTTest.add(0);


    System.out.println(myBSTTest.remove( target: 3));
    System.out.println(myBSTTest.remove( target: 99999));
```



```
true
false
```

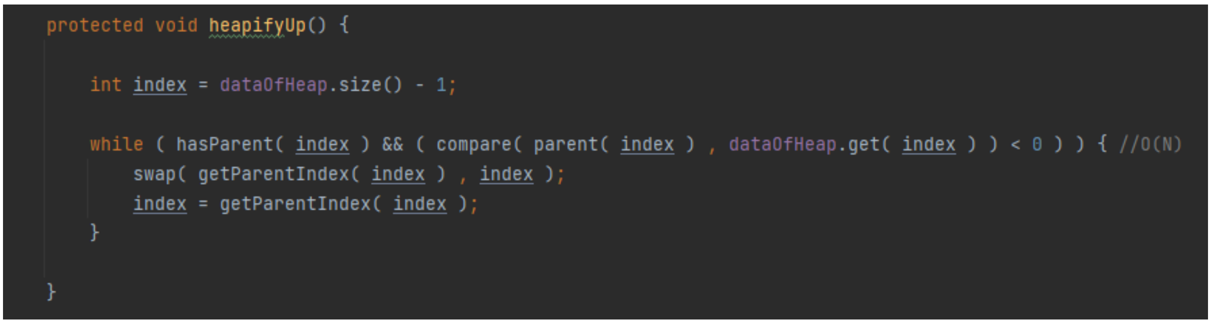


```java
BST<Integer> myBSTTest = new BST<Integer>( size: 20);

myBSTTest.add(3);
myBSTTest.add(1);
myBSTTest.add(4);
myBSTTest.add(9);
myBSTTest.add(2);
myBSTTest.add(5);
myBSTTest.add(0);


System.out.println(myBSTTest.delete( target: 3));
System.out.println(myBSTTest.delete( target: 99999));
```

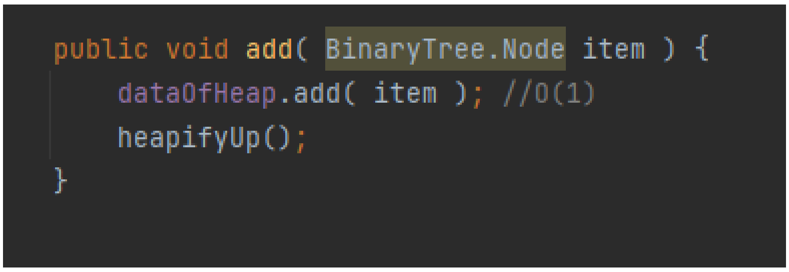


```
3
null
```

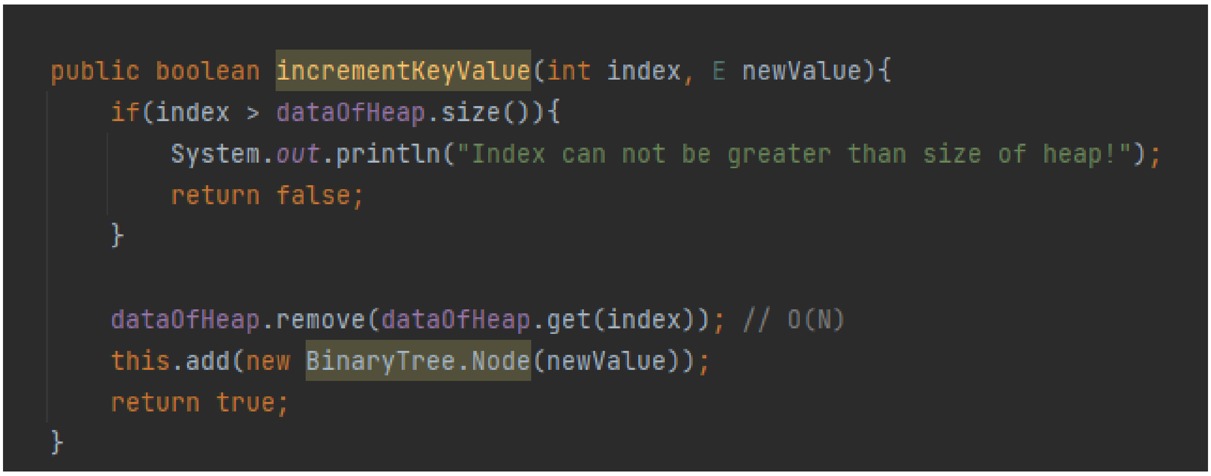
Analysis



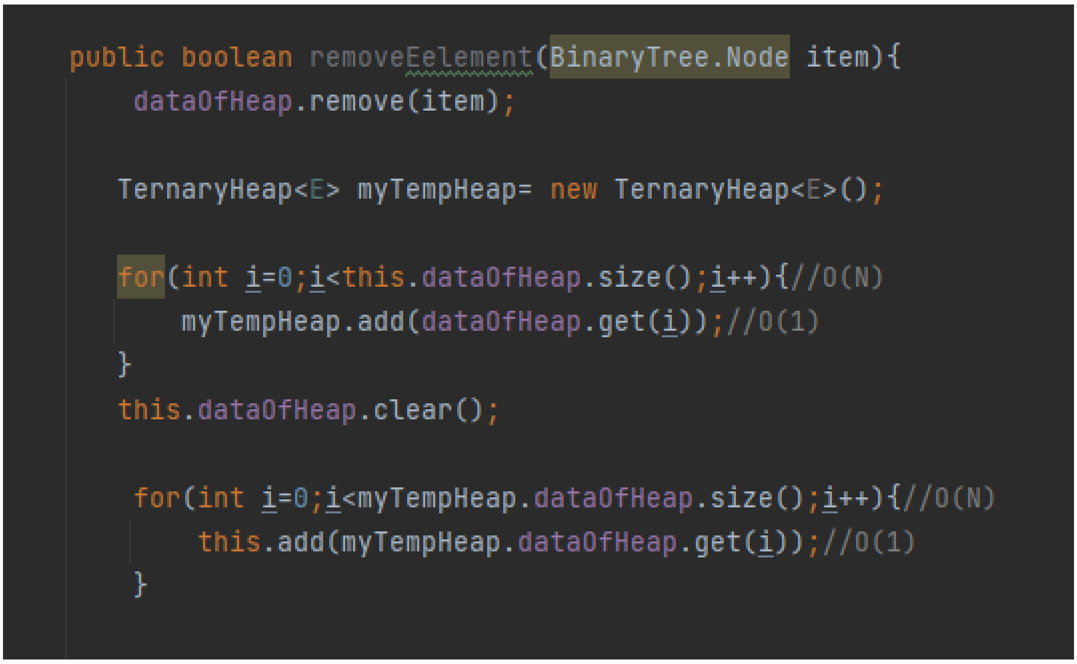
```java
protected void heapifyUp() {

    int index = dataOfHeap.size() - 1;

    while ( hasParent( index ) && ( compare( parent( index ) , dataOfHeap.get( index ) ) < 0 ) ) { //O(N)
        swap( getParentIndex( index ) , index );
        index = getParentIndex( index );
    }

}
```



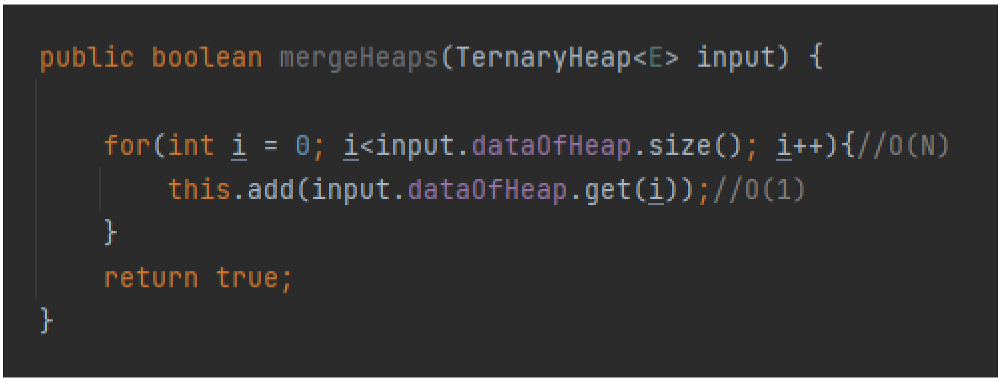
```java
public void add( BinaryTree.Node item ) {
    dataOfHeap.add( item ); //O(1)
    heapifyUp();
}
```



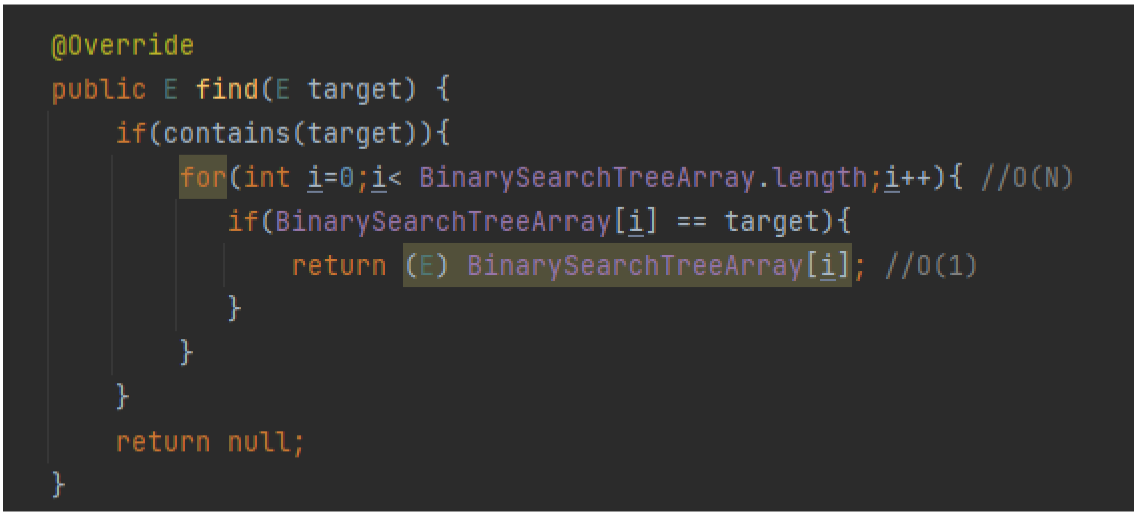
```java
public boolean incrementKeyValue(int index, E newValue){
    if(index > dataOfHeap.size()){
        System.out.println("Index can not be greater than size of heap!");
        return false;
    }

    dataOfHeap.remove(dataOfHeap.get(index)); // O(N)
    this.add(new BinaryTree.Node(newValue));
    return true;
}
```



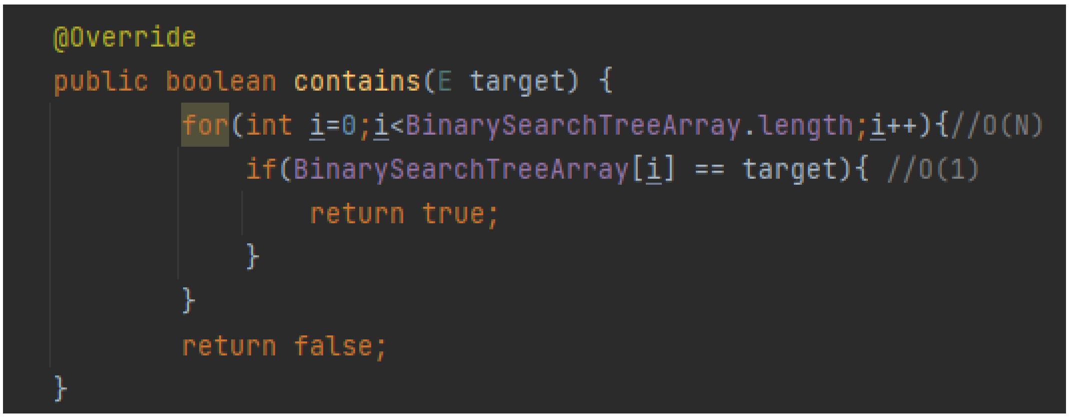
```java
public boolean removeEelement(BinaryTree.Node item){
     dataOfHeap.remove(item);

   TernaryHeap<E> myTempHeap= new TernaryHeap<E>();

   for(int i=0;i<this.dataOfHeap.size();i++){//O(N)
       myTempHeap.add(dataOfHeap.get(i));//O(1)
   }
   this.dataOfHeap.clear();

    for(int i=0;i<myTempHeap.dataOfHeap.size();i++){//O(N)
        this.add(myTempHeap.dataOfHeap.get(i));//O(1)
    }
```



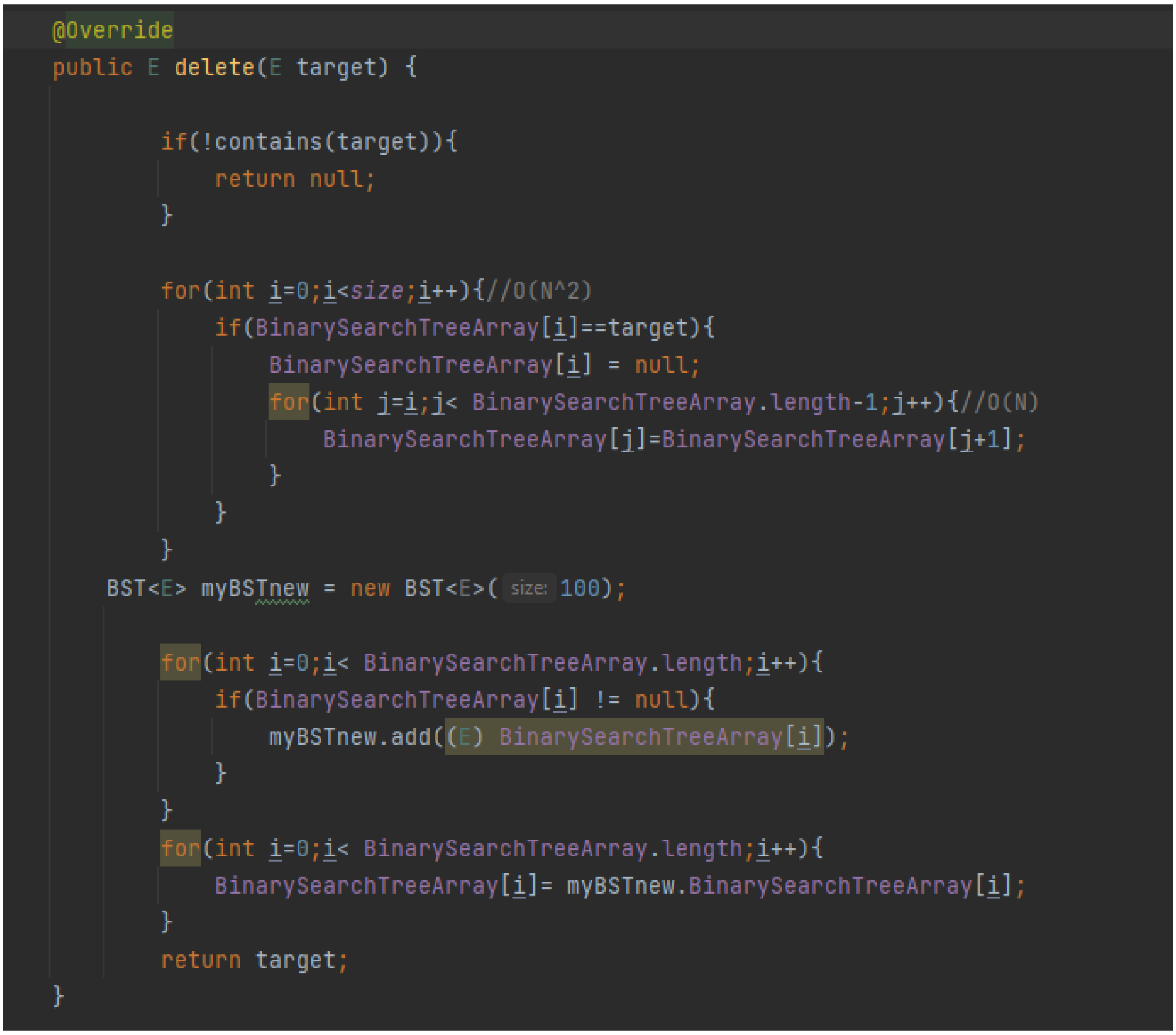
```java
public boolean mergeHeaps(TernaryHeap<E> input) {

    for(int i = 0; i<input.dataOfHeap.size(); i++){//O(N)
        this.add(input.dataOfHeap.get(i));//O(1)
    }
    return true;
}
```



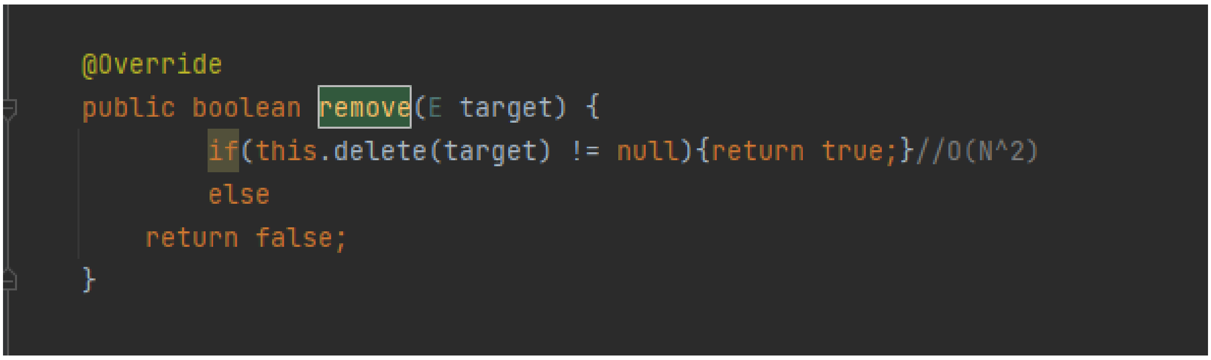
```
@Override
public E find(E target) {
    if(contains(target)){
        for(int i=0;i< BinarySearchTreeArray.length;i++){ //O(N)
           if(BinarySearchTreeArray[i] == target){
               return (E) BinarySearchTreeArray[i]; //O(1)
           }
        }
    }
    return null;
}
```



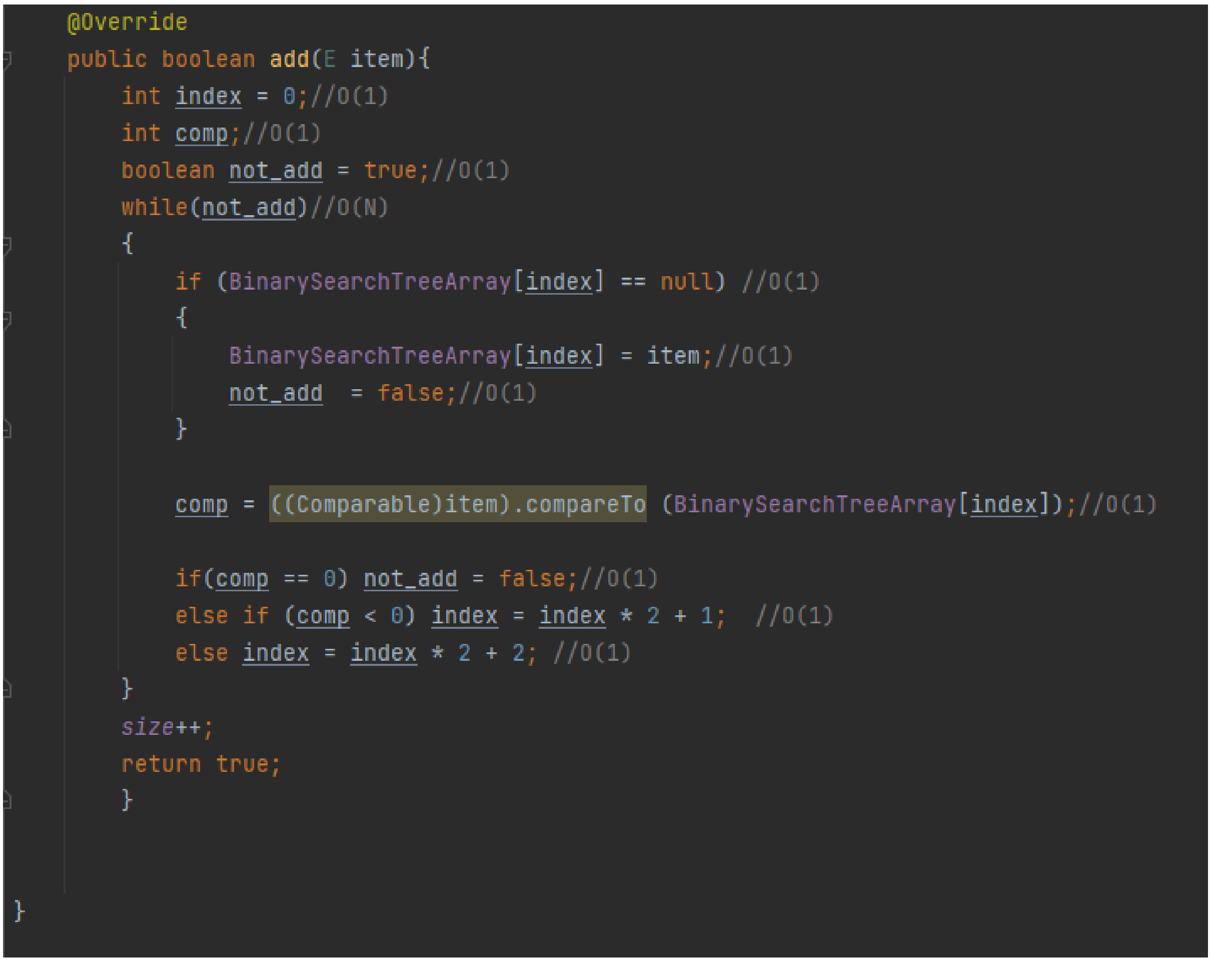
```
@Override
public boolean contains(E target) {
        for(int i=0;i<BinarySearchTreeArray.length;i++){//O(N)
            if(BinarySearchTreeArray[i] == target){ //O(1)
                return true;
            }
        }
        return false;
}
```



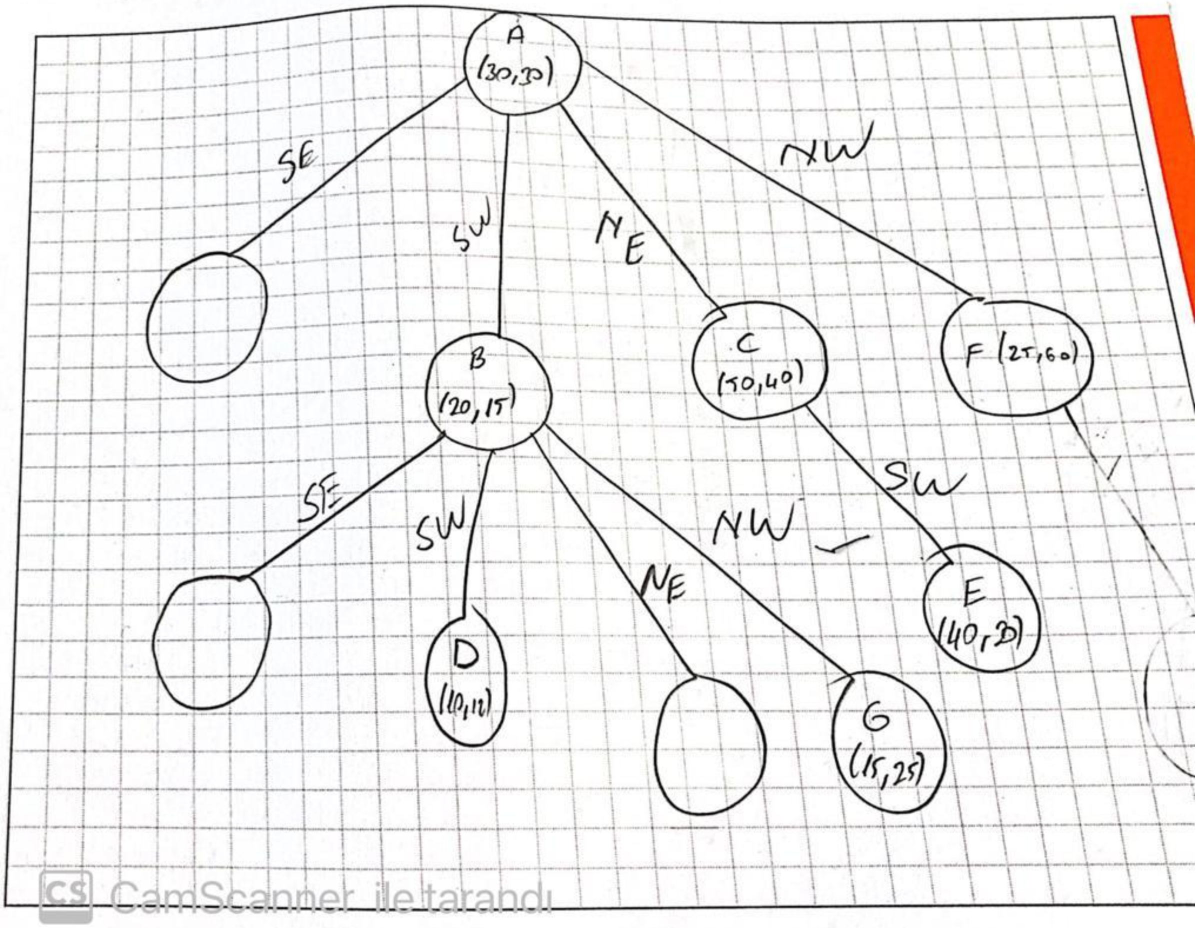
```
@Override
public E delete(E target) {

        if(!contains(target)){
            return null;
        }

        for(int i=0;i<size;i++){//O(N^2)
            if(BinarySearchTreeArray[i]==target){
                BinarySearchTreeArray[i] = null;
                for(int j=i;j< BinarySearchTreeArray.length-1;j++){//O(N)
                    BinarySearchTreeArray[j]=BinarySearchTreeArray[j+1];
                }
            }
        }
    BST<E> myBSTnew = new BST<E>( size: 100);

        for(int i=0;i< BinarySearchTreeArray.length;i++){
            if(BinarySearchTreeArray[i] != null){
                myBSTnew.add((E) BinarySearchTreeArray[i]);
            }
        }
        for(int i=0;i< BinarySearchTreeArray.length;i++){
            BinarySearchTreeArray[i]= myBSTnew.BinarySearchTreeArray[i];
        }
        return target;
}
```

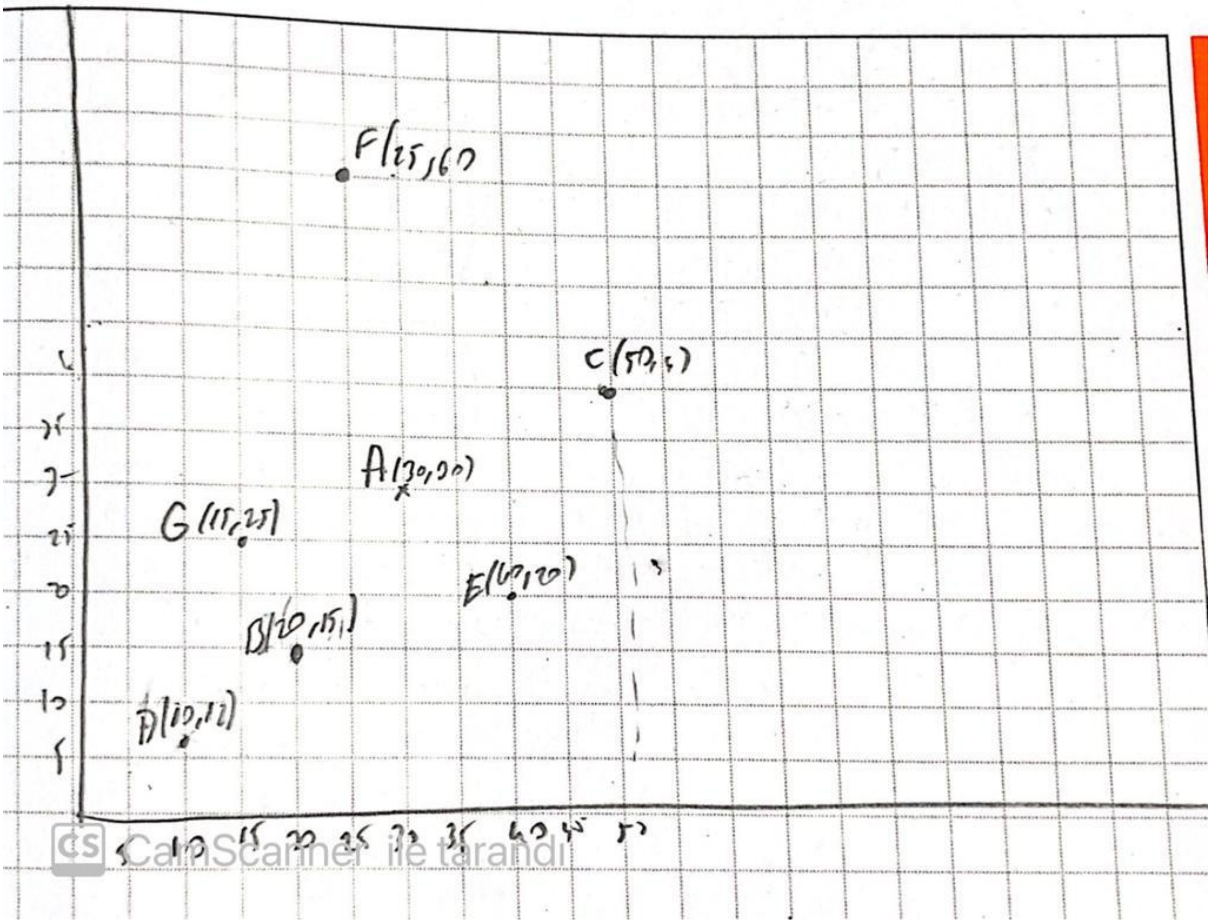


```
@Override
public boolean remove(E target) {
        if(this.delete(target) != null){return true;}//O(N^2)
        else
    return false;
}
```



```
    @Override
    public boolean add(E item){
        int index = 0;//O(1)
        int comp;//O(1)
        boolean not_add = true;//O(1)
        while(not_add)//O(N)
        {
            if (BinarySearchTreeArray[index] == null) //O(1)
            {
                BinarySearchTreeArray[index] = item;//O(1)
                not_add  = false;//O(1)
            }

            comp = ((Comparable)item).compareTo (BinarySearchTreeArray[index]);//O(1)

            if(comp == 0) not_add = false;//O(1)
            else if (comp < 0) index = index * 2 + 1;  //O(1)
            else index = index * 2 + 2; //O(1)
        }
        size++;
        return true;
        }


}
```


A
(30,30)
SE
SW
NE
NW
B
(20,15)
C
(50,40)
F (25,60)
SE
SW
NE
NW
SW
D
(10,12)
E
(40,20)
G
(15,25)


F(25,60)
C(50,40)
A(30,30)
G(15,25)
E(40,20)
B(20,15)
D(10,10)
40
35
30
25
20
15
10
5
5
10
15
20
25
30
35
40
45
50