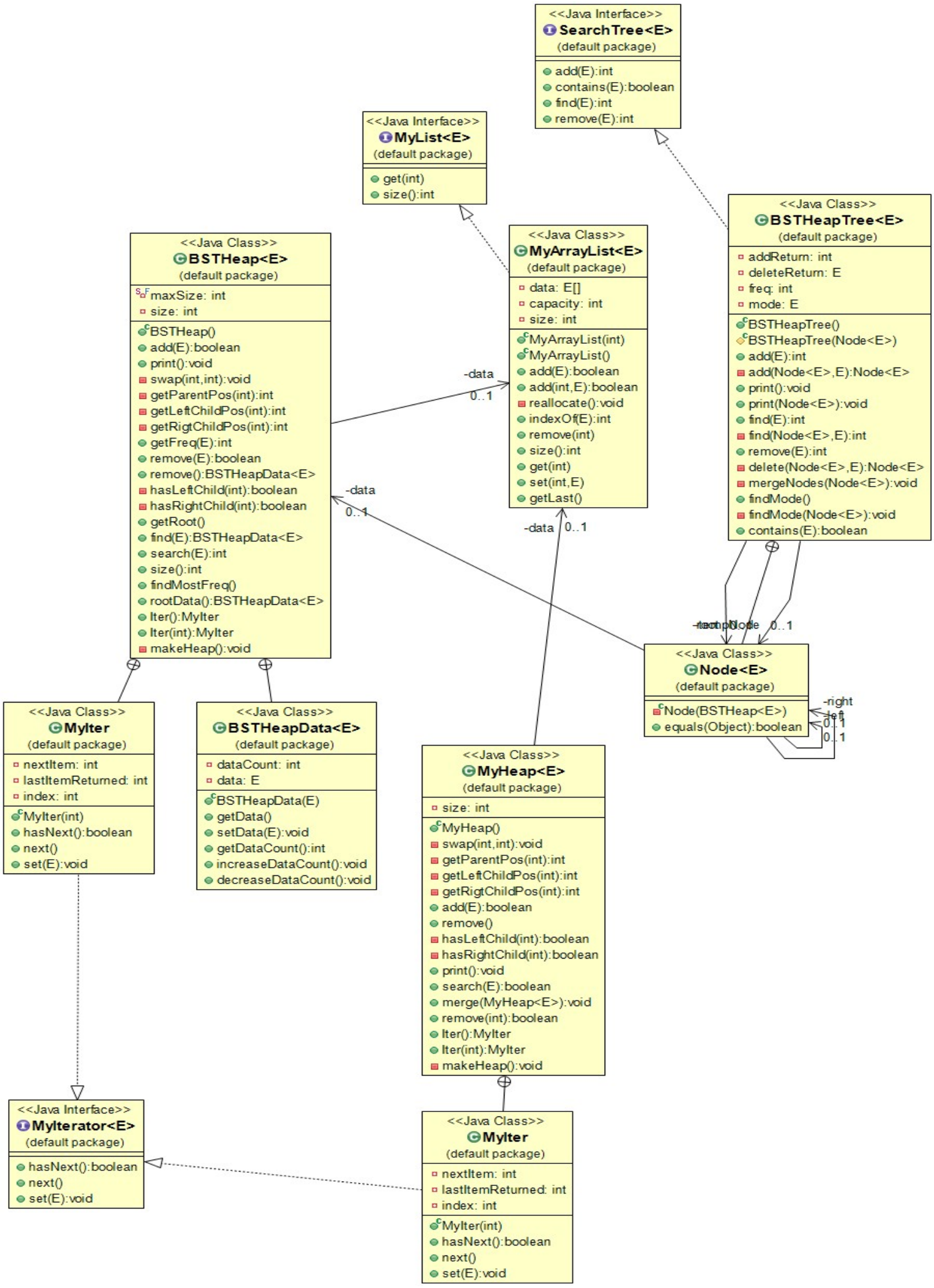
GIT Department of Computer Engineering
CSE 222/505 - Spring 2021
Homework #4 Report

Atakan ALTIN
1801042668

1. SYSTEM REQUIREMENTS

2. USE CASE AND CLASS DIAGRAMS



4. PROBLEM SOLUTION APPROACH

Heap data structure is a Complete Binary Tree. Heap is typically represented as an array because it is much more efficient than the Tree-Node representation. In order to use Iterator class with set operation custom Iterator interface is created and implemented. In order to build BSTHeapTree class some customizations are made to the Heap class such as adding a frequency to the elements of Heap structure. If the heap in the BST node is full new BST node created according to the root of the heap. Remove process is handled with two steps. First, which heap the given element is in is found. Then the root which founded heap is in is extracted. After this operation two root nodes are obtained. Lastly these roots are merged by adding elements in the second root into the first root by using add function of BSTHeapTree. As a result, tree-heap structure is preserved in this solution approach.

5. TEST CASES

--PART I--

1-Add an element into the empty heap.
2-Remove an element from the heap.
3-Create a new heap and merge them with each other.
4-Find an element in the array.
5-Find an element which is not in the array.
6-Find nth largest element.
7-Find nth largest element when the n is larger than heap size.
8-Set the element last returned by iterator.
9-Set the element when iterator has just created.

--PART II--

1-Add an element into the empty BSTHeapTree.
2-Remove an element from BSTHeapTree.
3-Remove an element from empty BSTHeapTree.
4-Add element to see if the tree structure is preserved or not after the remove process.
5-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.
6-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.
7-Find the mode of the BSTHeapTree. Check whether the mode value is correct.
8-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.

6. RUNNING AND RESULTS

```
Unit test for PART I
Removing element from empty heap.
Remove process cannot be done.
Adding some elements to the heap.
 PARENT : 89 LEFT CHILD : 74 RIGHT CHILD :76
 PARENT : 74 LEFT CHILD : 39 RIGHT CHILD :66
 PARENT : 76 LEFT CHILD : 28 RIGHT CHILD :8
 PARENT : 39 LEFT CHILD : 20 RIGHT CHILD :37
 PARENT : 66 LEFT CHILD : 29 RIGHT CHILD :32
 PARENT : 28 LEFT CHILD : 18 RIGHT CHILD :26
```

```
Removing some elements from heap
 PARENT : 39 LEFT CHILD : 37 RIGHT CHILD :28
 PARENT : 37 LEFT CHILD : 29 RIGHT CHILD :32
 PARENT : 28 LEFT CHILD : 26 RIGHT CHILD :8
 PARENT : 29 LEFT CHILD : 20 RIGHT CHILD :18
```

```
Creating Another heap:
 PARENT : 15 LEFT CHILD : 9 RIGHT CHILD :10
 PARENT : 9 LEFT CHILD : 2
Merging two heaps:
 PARENT : 39 LEFT CHILD : 37 RIGHT CHILD :28
 PARENT : 37 LEFT CHILD : 29 RIGHT CHILD :32
 PARENT : 28 LEFT CHILD : 26 RIGHT CHILD :8
 PARENT : 29 LEFT CHILD : 20 RIGHT CHILD :18
 PARENT : 32 LEFT CHILD : 15 RIGHT CHILD :9
 PARENT : 26 LEFT CHILD : 10 RIGHT CHILD :2
```

```
Trying to reach out of bounds.
Index out of bounds. Exception are handled.
Removing 5th largest element from the heap. It should be 28
 PARENT : 39 LEFT CHILD : 32 RIGHT CHILD :37
 PARENT : 32 LEFT CHILD : 20 RIGHT CHILD :29
 PARENT : 37 LEFT CHILD : 18 RIGHT CHILD :15
 PARENT : 20 LEFT CHILD : 2 RIGHT CHILD :10
 PARENT : 29 LEFT CHILD : 9 RIGHT CHILD :26
 PARENT : 18 LEFT CHILD : 8
```

```
Is there 120 in de heap?
false
Is there 15 in de heap?
true
```

```
Attempting to set an element without next() call in the iterator
Please call next() method first.
Printing elements using iterator
39,32,37,20,29,18,15,2,10,9,26,8,
```

```
Set with iterator the last element : 8 to 65
 PARENT : 65 LEFT CHILD : 32 RIGHT CHILD :37
 PARENT : 32 LEFT CHILD : 20 RIGHT CHILD :29
 PARENT : 37 LEFT CHILD : 18 RIGHT CHILD :15
 PARENT : 20 LEFT CHILD : 2 RIGHT CHILD :10
 PARENT : 29 LEFT CHILD : 9 RIGHT CHILD :26
 PARENT : 18 LEFT CHILD : 39
```

```
Unit test for PART II
BSTHeap Mode: 1781
Array Mode: 100
```

```
 PARENT : 37,1 LEFT CHILD : 23,1 RIGHT CHILD :10,2
 PARENT : 23,1 LEFT CHILD : 16,1 RIGHT CHILD :19,1
 PARENT : 10,2 LEFT CHILD : 9,3 RIGHT CHILD :3,1

LEFT
 PARENT : 31,1 LEFT CHILD : 15,1 RIGHT CHILD :29,1
 PARENT : 15,1 LEFT CHILD : 13,4

RIGHT
 PARENT : 124,1 LEFT CHILD : 52,1 RIGHT CHILD :98,1
 PARENT : 52,1 LEFT CHILD : 51,3 RIGHT CHILD :38,1
 PARENT : 98,1 LEFT CHILD : 87,1 RIGHT CHILD :80,1

LEFT
 PARENT : 60,1 LEFT CHILD : 57,1 RIGHT CHILD :43,3
 PARENT : 57,1 LEFT CHILD : 54,1 RIGHT CHILD :39,2
```

PARENT

PARENT-LEFT

PARENT-RIGHT

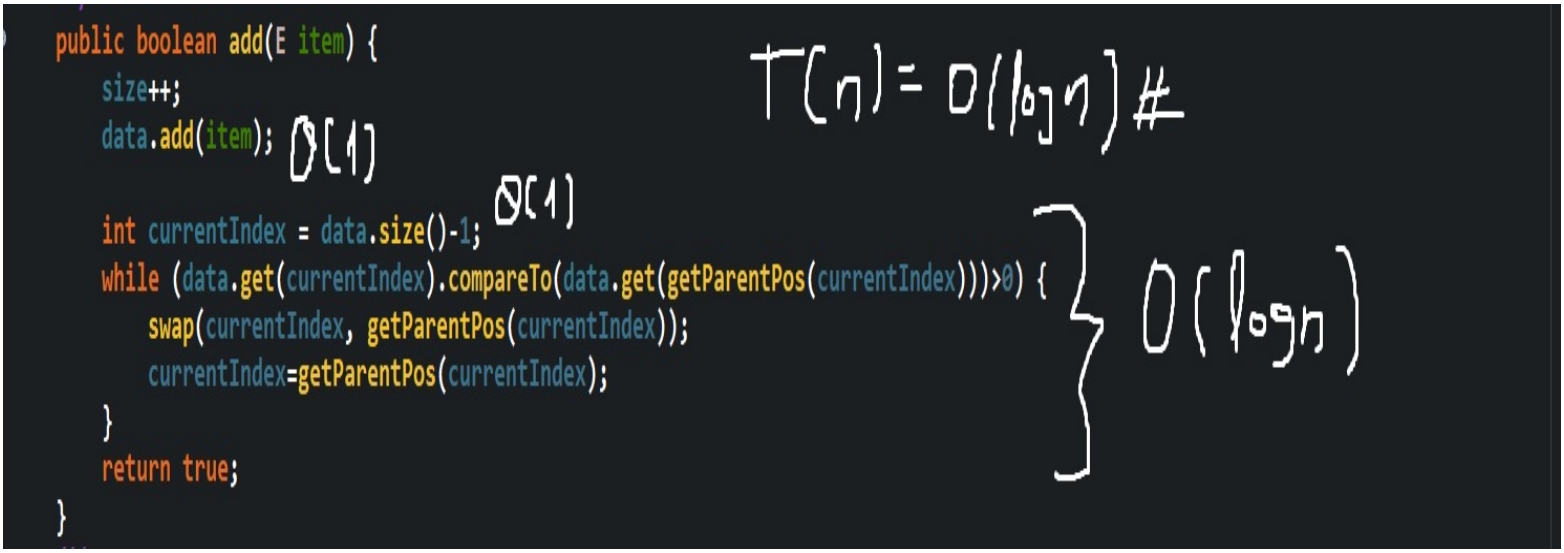
PARENT-RIGHT-LEFT

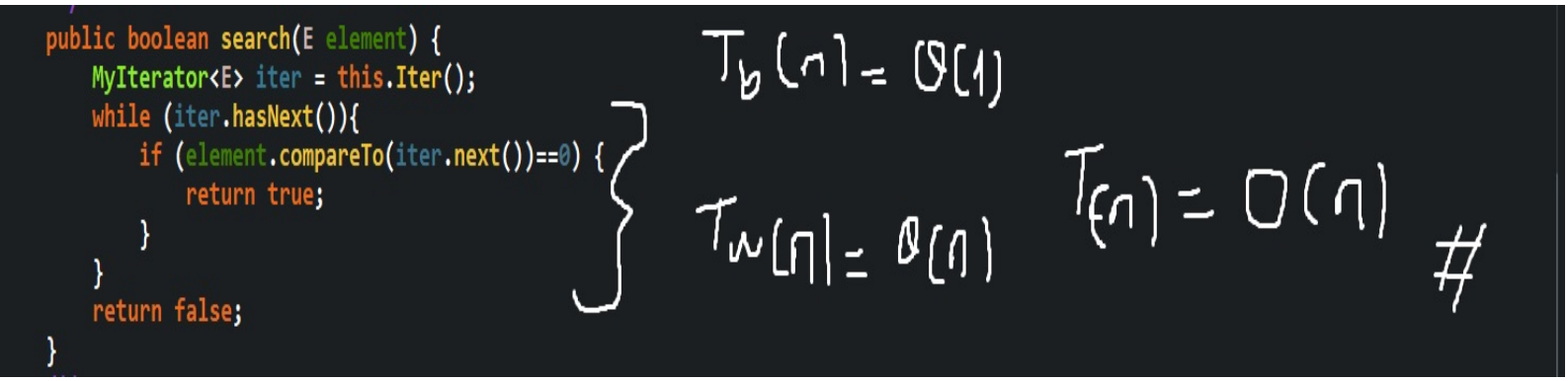
```
-------------------------
 PARENT : 31,1 LEFT CHILD : 19,1 RIGHT CHILD :23,1
 PARENT : 19,1 LEFT CHILD : 16,1 RIGHT CHILD :3,1
 PARENT : 23,1 LEFT CHILD : 9,3 RIGHT CHILD :10,2

LEFT
 PARENT : 29,1 LEFT CHILD : 15,1 RIGHT CHILD :13,4

RIGHT
 PARENT : 124,1 LEFT CHILD : 52,1 RIGHT CHILD :98,1
 PARENT : 52,1 LEFT CHILD : 51,3 RIGHT CHILD :38,1
 PARENT : 98,1 LEFT CHILD : 87,1 RIGHT CHILD :80,1

LEFT
 PARENT : 60,1 LEFT CHILD : 57,1 RIGHT CHILD :43,3
 PARENT : 57,1 LEFT CHILD : 54,1 RIGHT CHILD :39,2
```

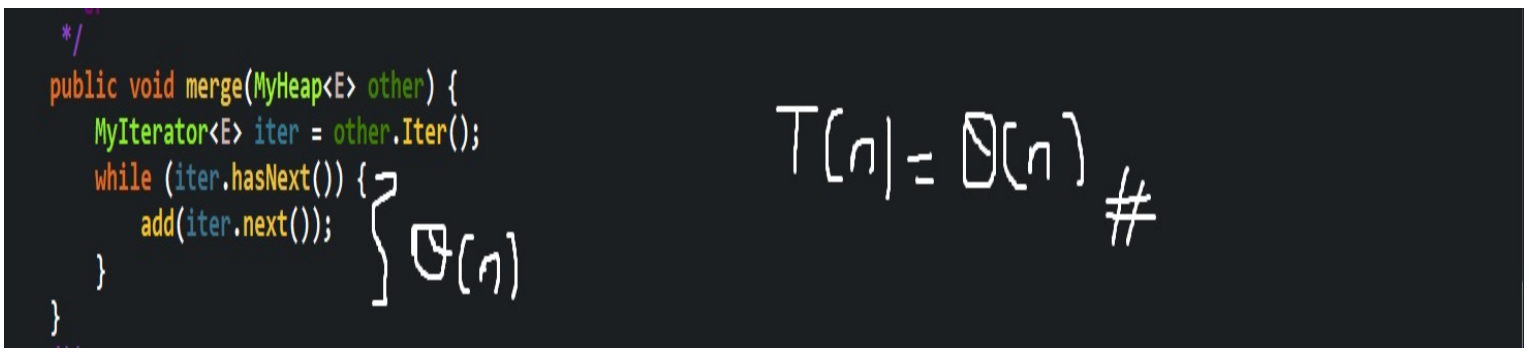
AFTER REMOVİNG ROOT

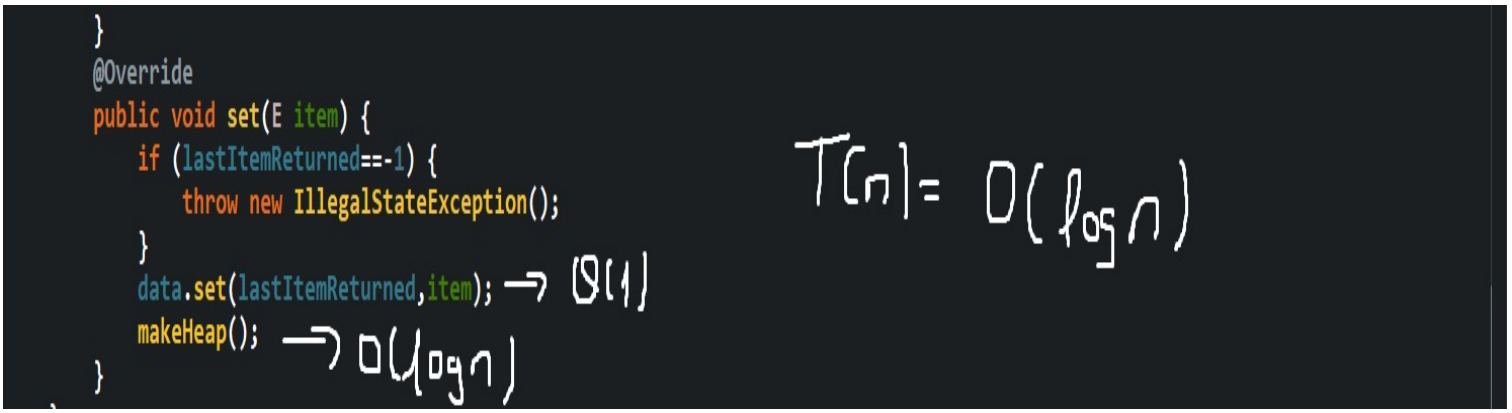
```
 Searched Item: 3479 BST Frequency: 1 Array Frequency: 1
 Searched Item: 350 BST Frequency: 2 Array Frequency: 2
 Searched Item: 2526 BST Frequency: 1 Array Frequency: 1
 Searched Item: 1662 BST Frequency: 1 Array Frequency: 1
 Searched Item: 3905 BST Frequency: 1 Array Frequency: 1
 Searched Item: 3089 BST Frequency: 2 Array Frequency: 2
 Searched Item: 3747 BST Frequency: 1 Array Frequency: 1
 Searched Item: 2930 BST Frequency: 1 Array Frequency: 1
 Searched Item: 1840 BST Frequency: 3 Array Frequency: 3
 Searched Item: 1338 BST Frequency: 2 Array Frequency: 2
 Searched Item: 4518 BST Frequency: 1 Array Frequency: 1
 Searched Item: 125 BST Frequency: 1 Array Frequency: 1
 Searched Item: 3129 BST Frequency: 2 Array Frequency: 2
 Searched Item: 3001 BST Frequency: 1 Array Frequency: 1
 Searched Item: 3746 BST Frequency: 3 Array Frequency: 3
 Searched Item: 2620 BST Frequency: 1 Array Frequency: 1
 Searched Item: 84 BST Frequency: 1 Array Frequency: 1
 Searched Item: 1581 BST Frequency: 1 Array Frequency: 1
 Searched Item: 4584 BST Frequency: 1 Array Frequency: 1
 Searched Item: 757 BST Frequency: 1 Array Frequency: 1
 Searched Item: 4005 BST Frequency: 2 Array Frequency: 2
 Searched Item: 1303 BST Frequency: 2 Array Frequency: 2
 Searched Item: 2046 BST Frequency: 1 Array Frequency: 1
 Searched Item: 4405 BST Frequency: 2 Array Frequency: 2
```

7. Time Complexity Analyzes









```java
private Node<E> add (Node<E> tempRoot,E element) {
    if (tempRoot==null) {
        BSTHeap<E> tempHeap = new BSTHeap<>();
        tempHeap.add(element);
        return new Node<E>(tempHeap);
    }
    if (tempRoot.data.add(element)) {
        addReturn = tempRoot.data.find(element).getDataCount();
        return tempRoot;
    }else {
        if ((element.compareTo(tempRoot.data.getRoot()))<0) {
            tempRoot.left=add(tempRoot.left, element);
            return tempRoot;
        }else {
            tempRoot.right = add(tempRoot.right, element);
            return tempRoot;
        }
    }
}
```

$O(\log n)$

$O(n)$

$T(n) = O(\log n(n + \log n))\#$

```java
private int find(Node<E> tempRoot, E element) {
    if (tempRoot==null) {
        throw new NoSuchElementException();
    }
    if (tempRoot.data.find(element)==null && element.compareTo(tempRoot.data.getRoot())<0) {
        return find(tempRoot.left,element);
    }
    else if (tempRoot.data.find(element)==null && element.compareTo(tempRoot.data.getRoot())>0 ) {
        return find(tempRoot.right,element);
    }
    else {
        return tempRoot.data.find(element).getDataCount();
    }
}
```

$T(n) = O(\log n)\#$

```java
private Node<E> delete(Node<E> tempRoot,E element) {
    if (tempRoot==null) {
        return null;
    }
    if (tempRoot.left.data.remove(element)) {
        tempNode = tempRoot.left;
        tempRoot.left=null;
        return tempRoot;
    }
    else if (tempRoot.right.data.remove(element)) {
        tempNode = tempRoot.right;
        tempRoot.right=null;
        return tempRoot;
    }
    else if (tempRoot.data.remove(element)) {
        tempNode=tempRoot;
        return tempRoot;
    }
    else {
        if (element.compareTo(tempRoot.data.getRoot())<0) {
            return delete(tempRoot.left,element);
        }
        else {
            return delete(tempRoot.right,element);
        }
    }
}
```

$O(\log n)$

$T(n) = O(\log^2 n)\#$