CSE222/505 Spring 2021
HW4

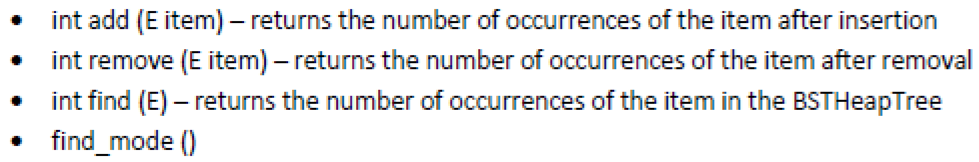
Oğulcan Kalafatoğlu
1801042613

1.System Requirements

From problem definitions we see that, in part 1 we need to implement Heap structure that has at least functions for:

1-Searching
2-Merging
3-Removing ith biggest element
4-Iterator with next and set methods

And in part 2 we need to implement BST tree that holds Heap and Occurence as data in its nodes, and needs at least these functions:

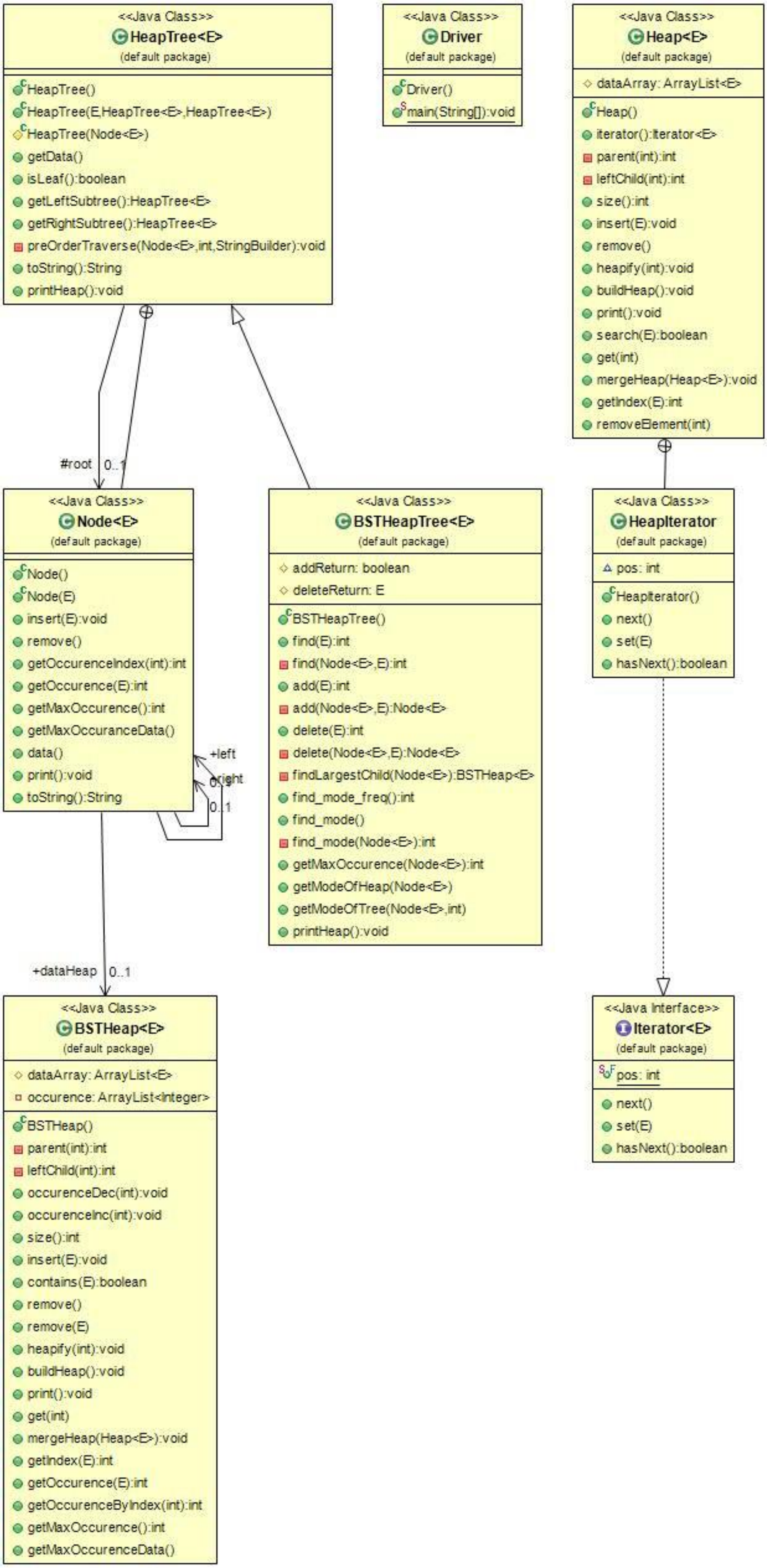


- int add (E item) – returns the number of occurrences of the item after insertion
- int remove (E item) – returns the number of occurrences of the item after removal
- int find (E) – returns the number of occurrences of the item in the BSTHeapTree
- find_mode ()

In part1, I created Heap and Iterator classes using textbook's pseudocode

In part2, I created BSTHeap, HeapTree and BSTHeapTree classes with help from textbook for some methods.

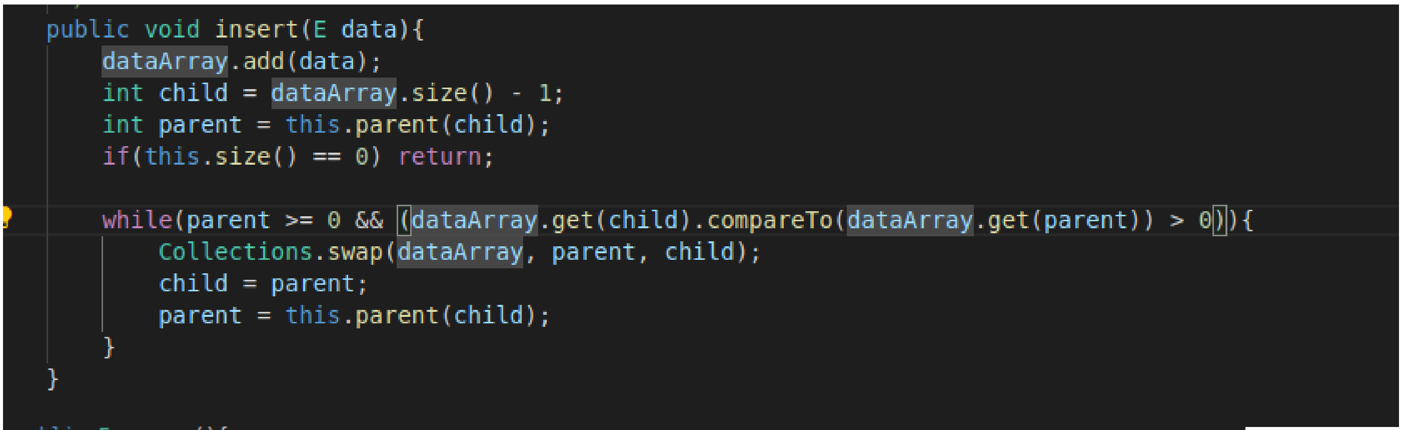
2. Class Diagrams



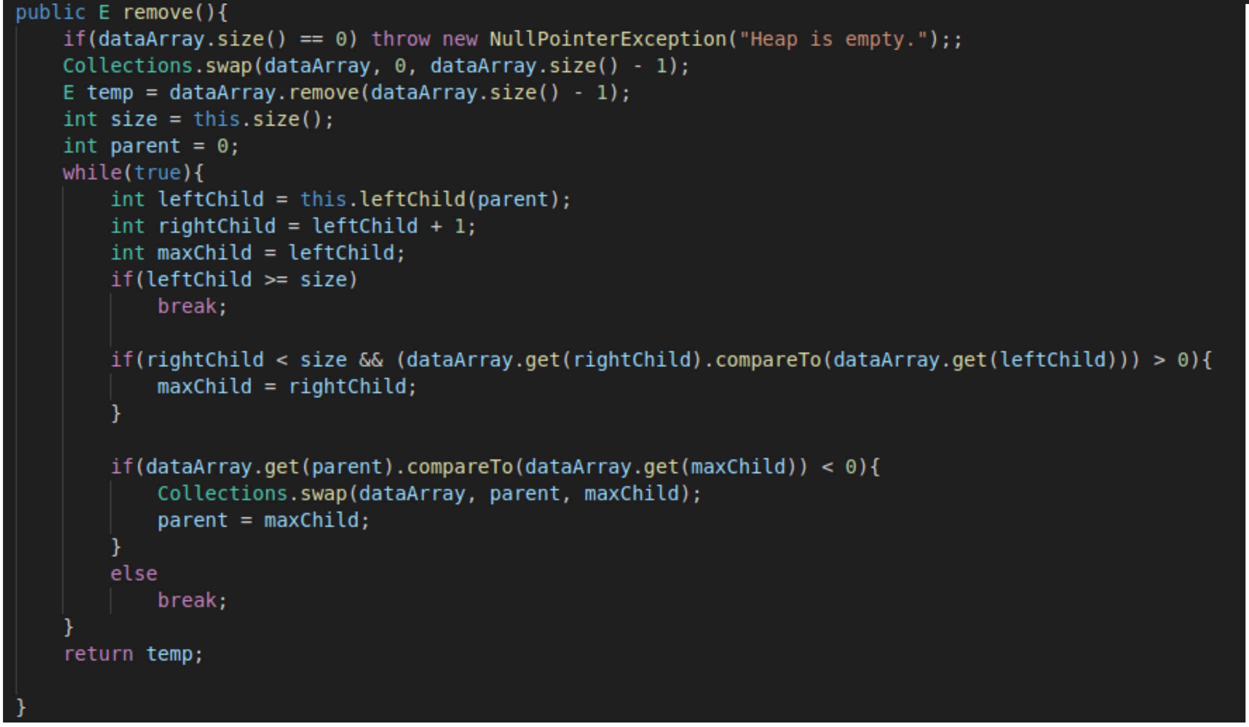
3. Problem Solutions approach

For part1:

I implemented heap using arrayList and used pseudocode of heap insert and heap remove from textbook

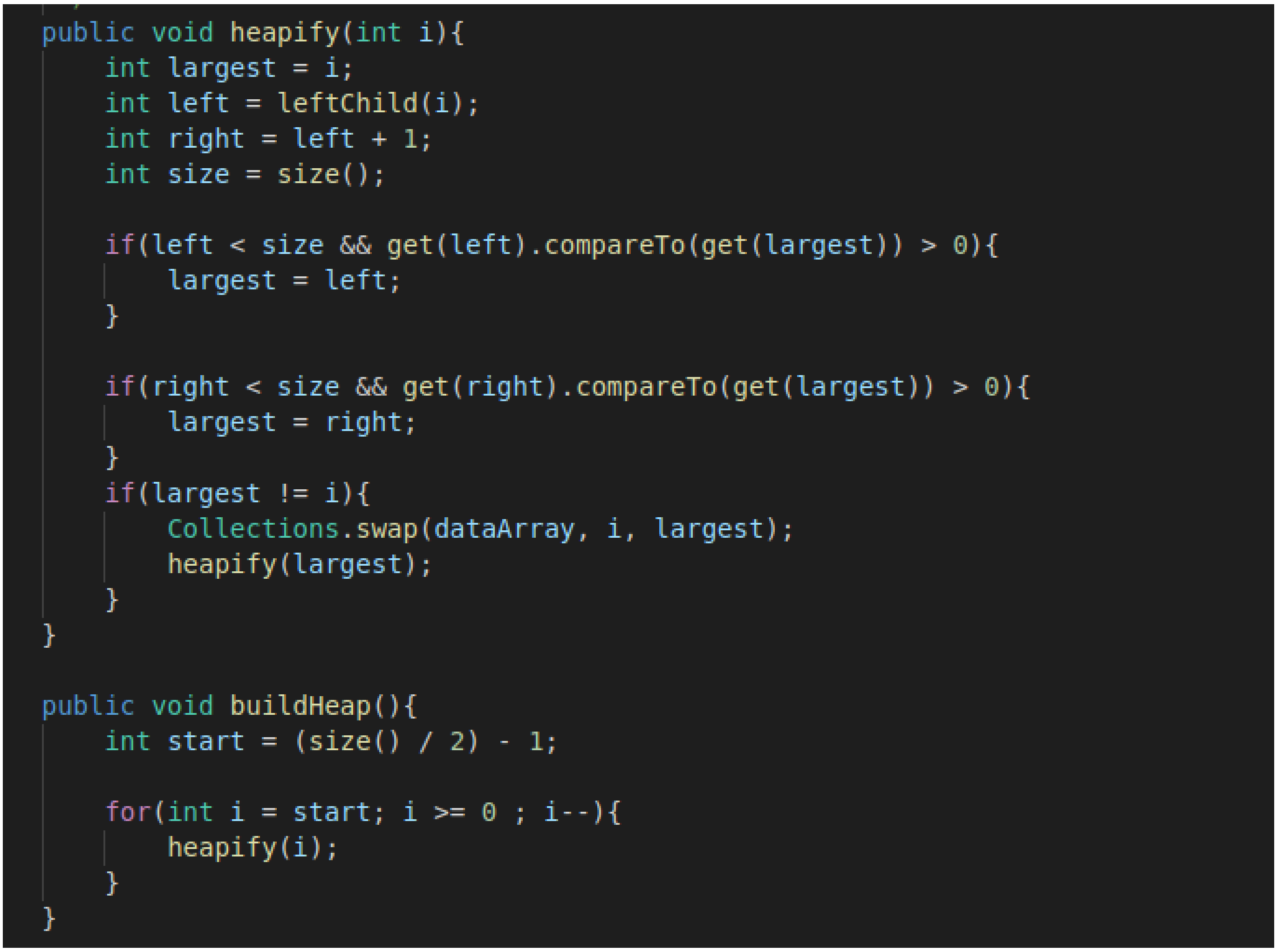


```
    public void insert(E data){
        dataArray.add(data);
        int child = dataArray.size() - 1;
        int parent = this.parent(child);
        if(this.size() == 0) return;

        while(parent >= 0 && (dataArray.get(child).compareTo(dataArray.get(parent)) > 0)){
            Collections.swap(dataArray, parent, child);
            child = parent;
            parent = this.parent(child);
        }
    }
```



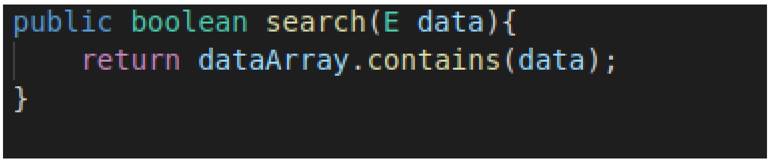
```
public E remove(){
    if(dataArray.size() == 0) throw new NullPointerException("Heap is empty.");;
    Collections.swap(dataArray, 0, dataArray.size() - 1);
    E temp = dataArray.remove(dataArray.size() - 1);
    int size = this.size();
    int parent = 0;
    while(true){
        int leftChild = this.leftChild(parent);
        int rightChild = leftChild + 1;
        int maxChild = leftChild;
        if(leftChild >= size)
            break;

        if(rightChild < size && (dataArray.get(rightChild).compareTo(dataArray.get(leftChild))) > 0){
            maxChild = rightChild;
        }

        if(dataArray.get(parent).compareTo(dataArray.get(maxChild)) < 0){
            Collections.swap(dataArray, parent, maxChild);
            parent = maxChild;
        }
        else
            break;
    }
    return temp;

}
```

But since I couldn’t find a use for remove() function instead used ArrayList’s remove function and wrote a function for building heap from unordered array so I could rebuild the heap after removing an element from the heap.



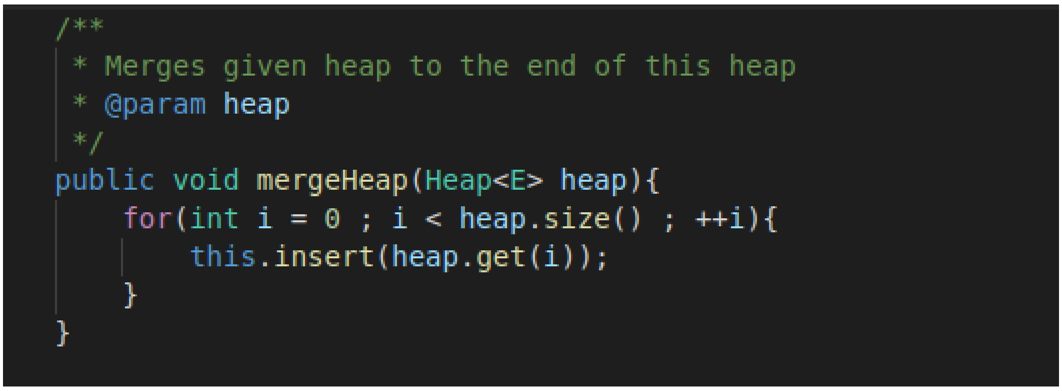
```
public void heapify(int i){
    int largest = i;
    int left = leftChild(i);
    int right = left + 1;
    int size = size();

    if(left < size && get(left).compareTo(get(largest)) > 0){
        largest = left;
    }

    if(right < size && get(right).compareTo(get(largest)) > 0){
        largest = right;
    }
    if(largest != i){
        Collections.swap(dataArray, i, largest);
        heapify(largest);
    }
}

public void buildHeap(){
    int start = (size() / 2) - 1;

    for(int i = start; i >= 0 ; i--){
        heapify(i);
    }
}
```

For search function, since heap was implemented using arrayList, used contains function of the arrayList for searching.



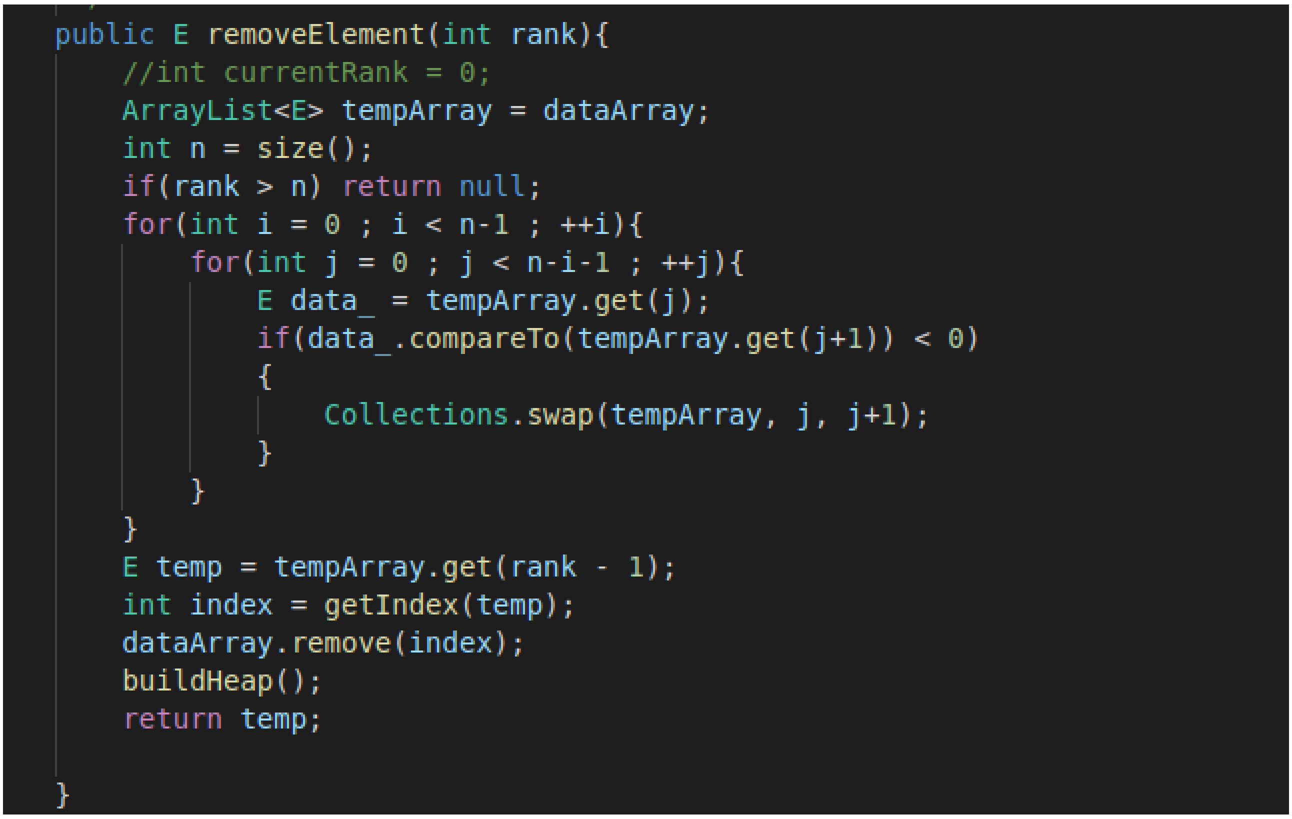
```
public boolean search(E data){
    return dataArray.contains(data);
}
```

For merging heaps, Inserting to the first heap was enough, since ordering of heap was done at insert.



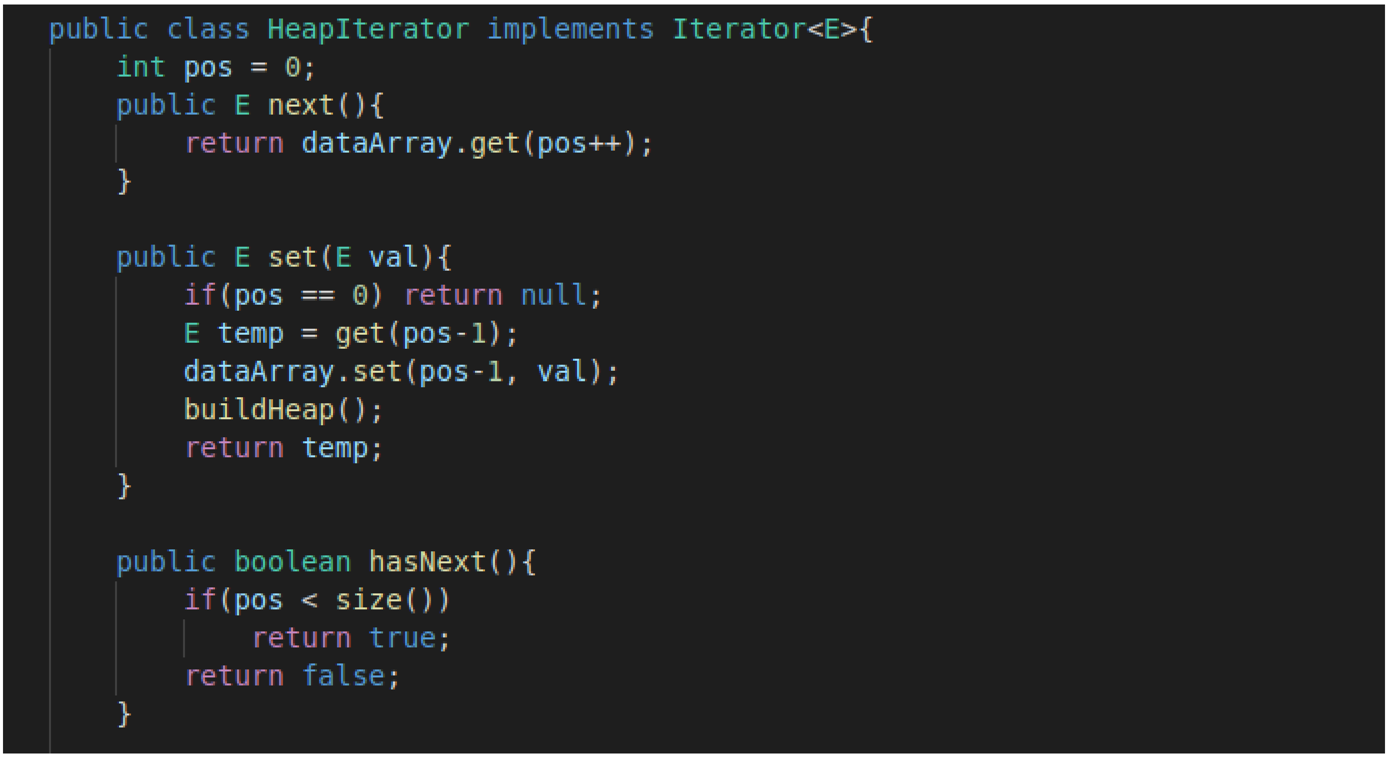
```java
/**
 * Merges given heap to the end of this heap
 * @param heap
 */
public void mergeHeap(Heap<E> heap){
    for(int i = 0 ; i < heap.size() ; ++i){
        this.insert(heap.get(i));
    }
}
```

For remove ith element function, created another temp array and copied heap to it, sorted the array and found wanted ith element, then removed it and rebuilt heap using buildHeap function.



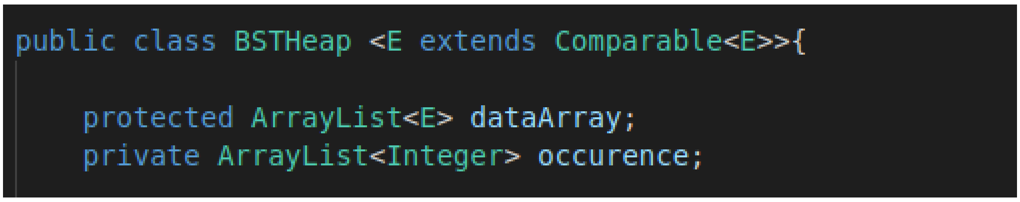
```java
public E removeElement(int rank){
    //int currentRank = 0;
    ArrayList<E> tempArray = dataArray;
    int n = size();
    if(rank > n) return null;
    for(int i = 0 ; i < n-1 ; ++i){
        for(int j = 0 ; j < n-i-1 ; ++j){
            E data_ = tempArray.get(j);
            if(data_.compareTo(tempArray.get(j+1)) < 0)
            {
                Collections.swap(tempArray, j, j+1);
            }
        }
    }
    E temp = tempArray.get(rank - 1);
    int index = getIndex(temp);
    dataArray.remove(index);
    buildHeap();
    return temp;

}
```

For iterator class, created a new generic Iterator interface, and implemented it in Heap class



```
    public class HeapIterator implements Iterator<E>{
        int pos = 0;
        public E next(){
            return dataArray.get(pos++);
        }

        public E set(E val){
            if(pos == 0) return null;
            E temp = get(pos-1);
            dataArray.set(pos-1, val);
            buildHeap();
            return temp;
        }

        public boolean hasNext(){
            if(pos < size())
                return true;
            return false;
        }
```

For part2 – Building BST and managing occurence of the heap elements was necessary, so i created new Heap class for node and created 2 classes for tree, as HeapTree and BSTHeapTree

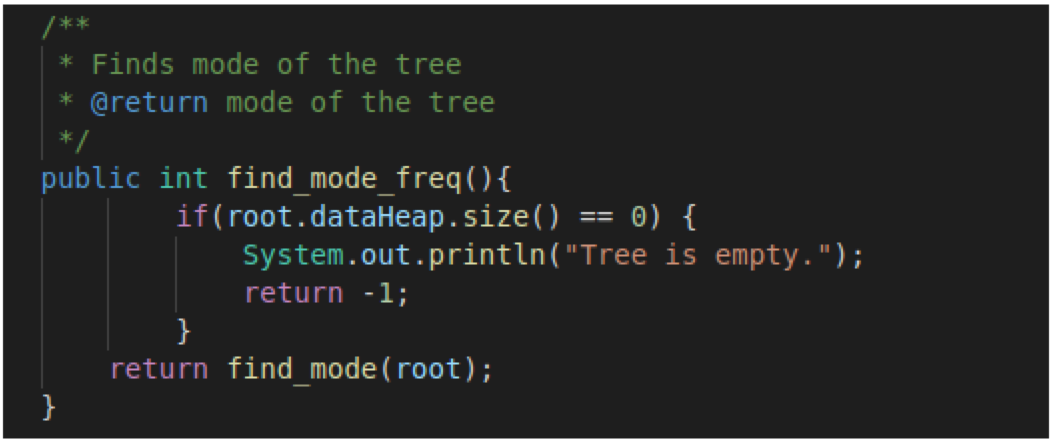


```
public class BSTHeap <E extends Comparable<E>>{

    protected ArrayList<E> dataArray;
    private ArrayList<Integer> occurence;
```

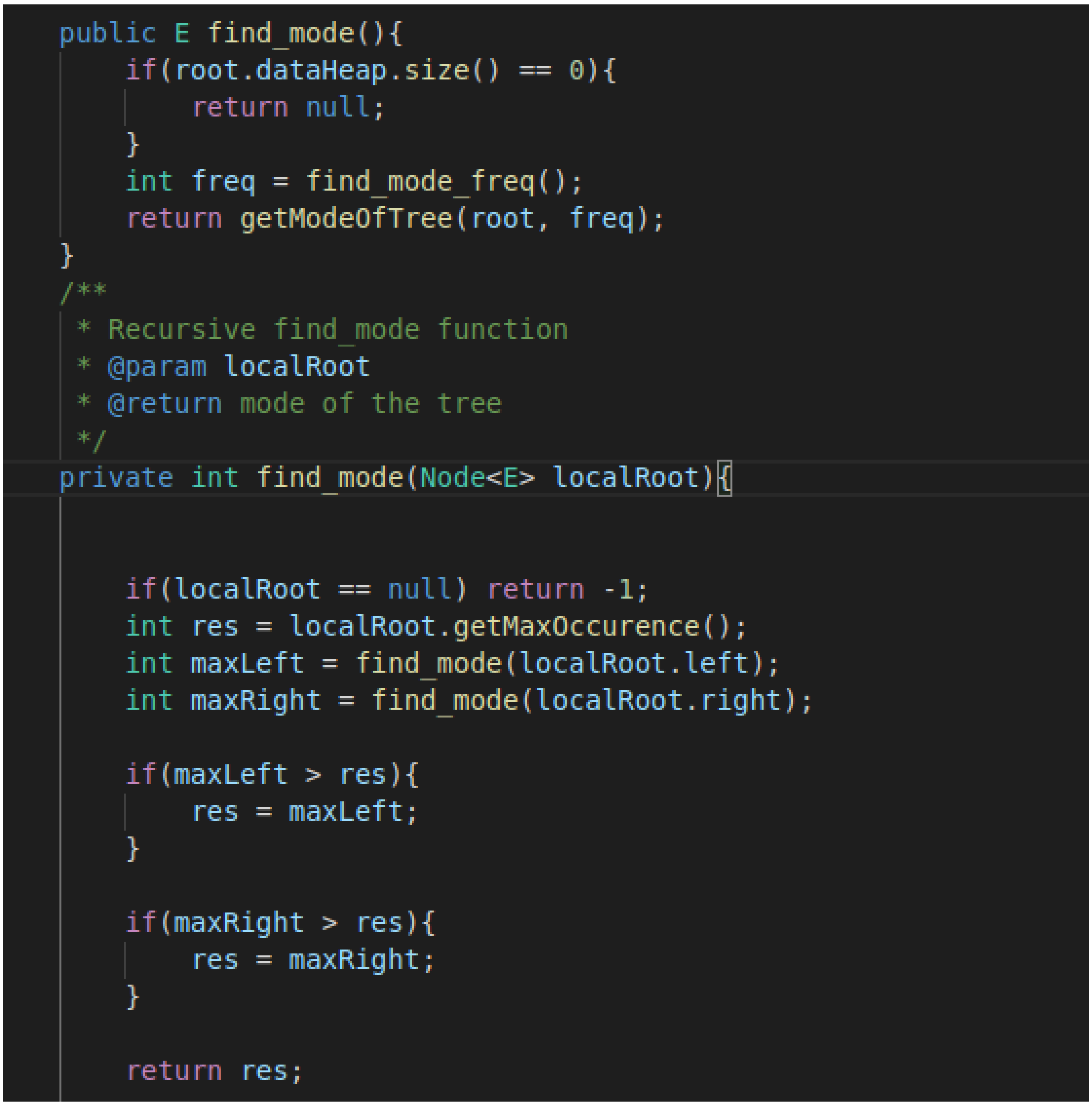
I managed occurence arrayList in all inserting and removing methods, and initialized occurence arrayList as 7 element in constructor.

Add,find and delete functions were taken from textbook and modified.

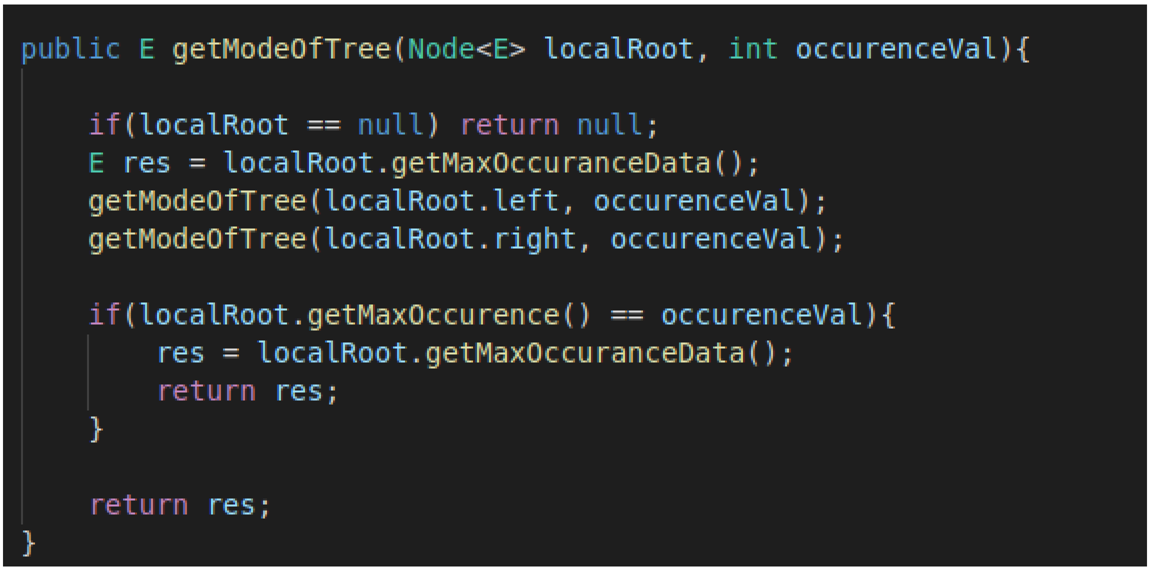
For finding mode, first most frequent value's frequency is found then value with that frequency is found with another function.



```
/**
 * Finds mode of the tree
 * @return mode of the tree
 */
public int find_mode_freq(){
        if(root.dataHeap.size() == 0) {
            System.out.println("Tree is empty.");
            return -1;
        }
    return find_mode(root);
}
```

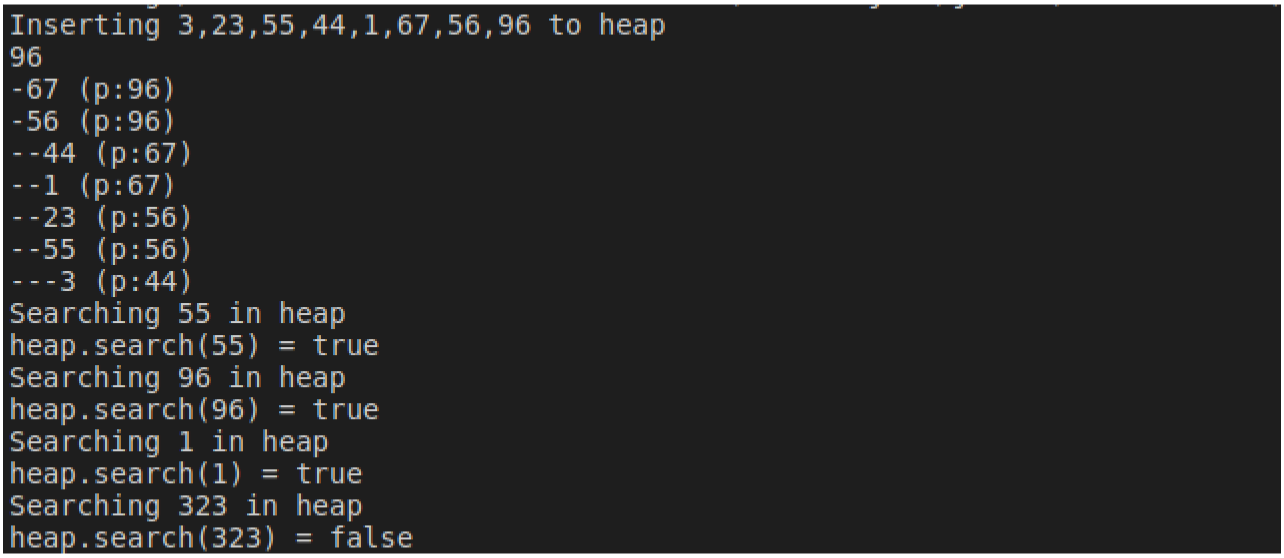


```
public E find_mode(){
    if(root.dataHeap.size() == 0){
        return null;
    }
    int freq = find_mode_freq();
    return getModeOfTree(root, freq);
}
/**
 * Recursive find_mode function
 * @param localRoot
 * @return mode of the tree
 */
private int find_mode(Node<E> localRoot){


    if(localRoot == null) return -1;
    int res = localRoot.getMaxOccurence();
    int maxLeft = find_mode(localRoot.left);
    int maxRight = find_mode(localRoot.right);

    if(maxLeft > res){
        res = maxLeft;
    }

    if(maxRight > res){
        res = maxRight;
    }

    return res;
```

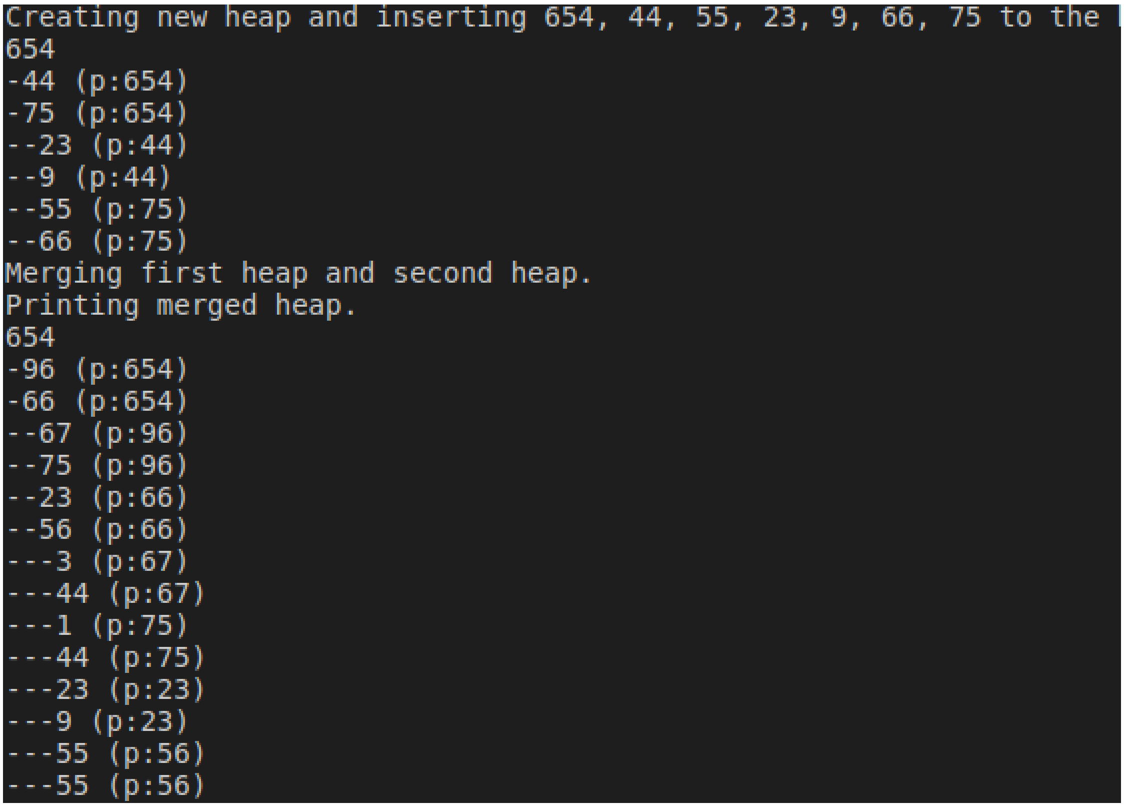


```
public E getModeOfTree(Node<E> localRoot, int occurenceVal){

    if(localRoot == null) return null;
    E res = localRoot.getMaxOccuranceData();
    getModeOfTree(localRoot.left, occurenceVal);
    getModeOfTree(localRoot.right, occurenceVal);

    if(localRoot.getMaxOccurence() == occurenceVal){
        res = localRoot.getMaxOccuranceData();
        return res;
    }

    return res;
}
```

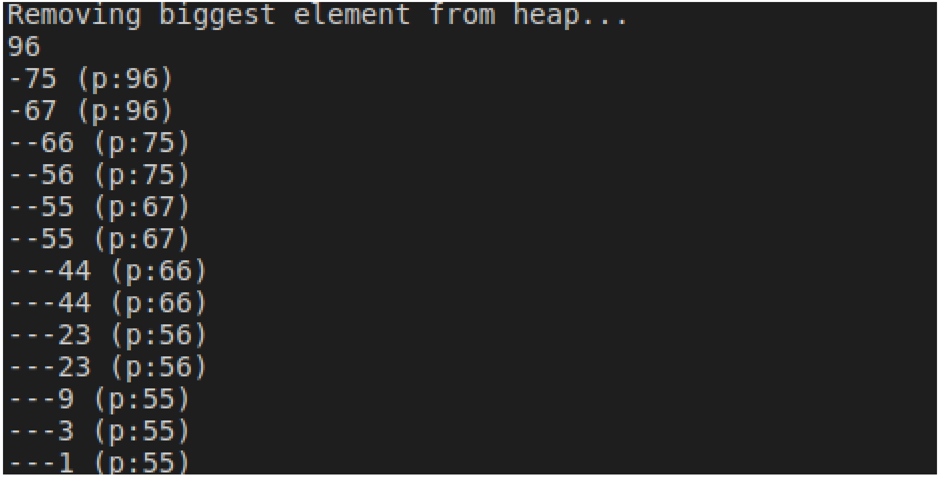
4. Test cases



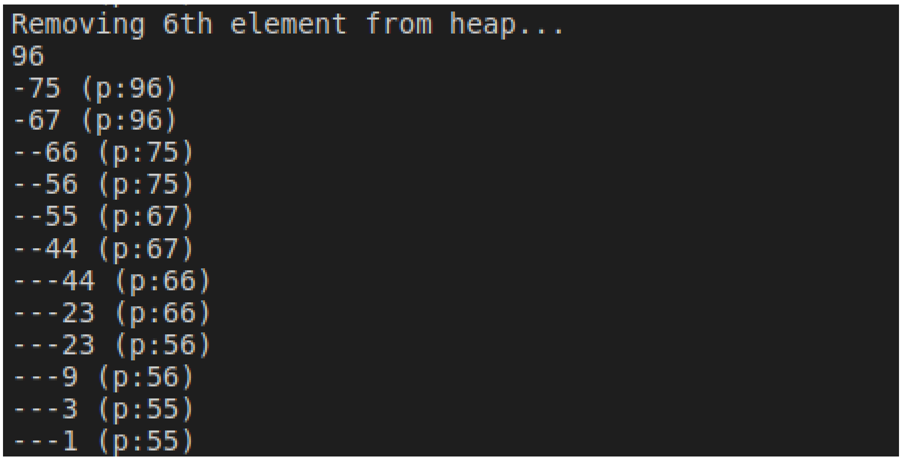
```
Inserting 3,23,55,44,1,67,56,96 to heap
96
-67 (p:96)
-56 (p:96)
--44 (p:67)
--1 (p:67)
--23 (p:56)
--55 (p:56)
---3 (p:44)
Searching 55 in heap
heap.search(55) = true
Searching 96 in heap
heap.search(96) = true
Searching 1 in heap
heap.search(1) = true
Searching 323 in heap
heap.search(323) = false
```



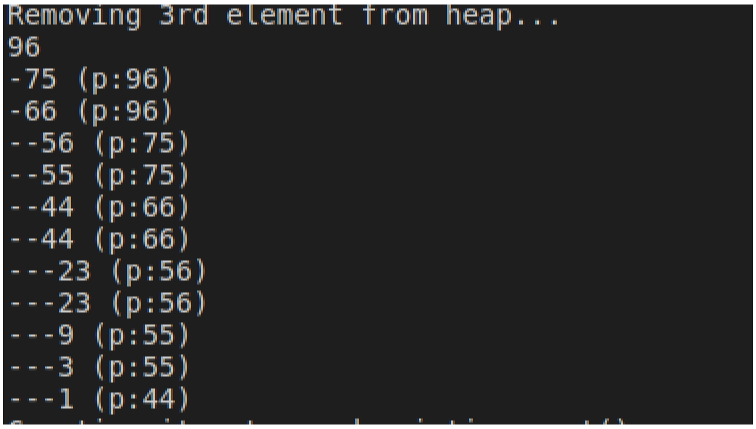
```
Creating new heap and inserting 654, 44, 55, 23, 9, 66, 75 to the
654
-44 (p:654)
-75 (p:654)
--23 (p:44)
--9 (p:44)
--55 (p:75)
--66 (p:75)
Merging first heap and second heap.
Printing merged heap.
654
-96 (p:654)
-66 (p:654)
--67 (p:96)
--75 (p:96)
--23 (p:66)
--56 (p:66)
---3 (p:67)
---44 (p:67)
---1 (p:75)
---44 (p:75)
---23 (p:23)
---9 (p:23)
---55 (p:56)
---55 (p:56)
```



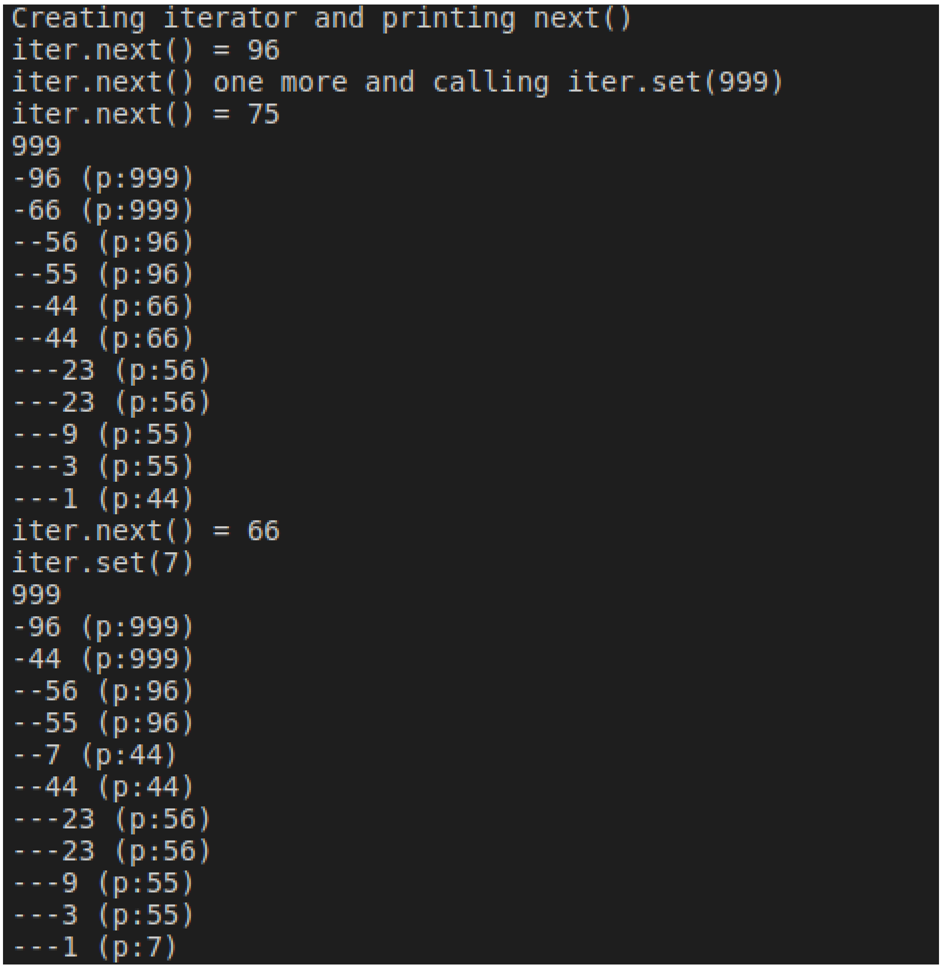
```
Removing biggest element from heap...
96
-75 (p:96)
-67 (p:96)
--66 (p:75)
--56 (p:75)
--55 (p:67)
--55 (p:67)
---44 (p:66)
---44 (p:66)
---23 (p:56)
---23 (p:56)
---9 (p:55)
---3 (p:55)
---1 (p:55)
```



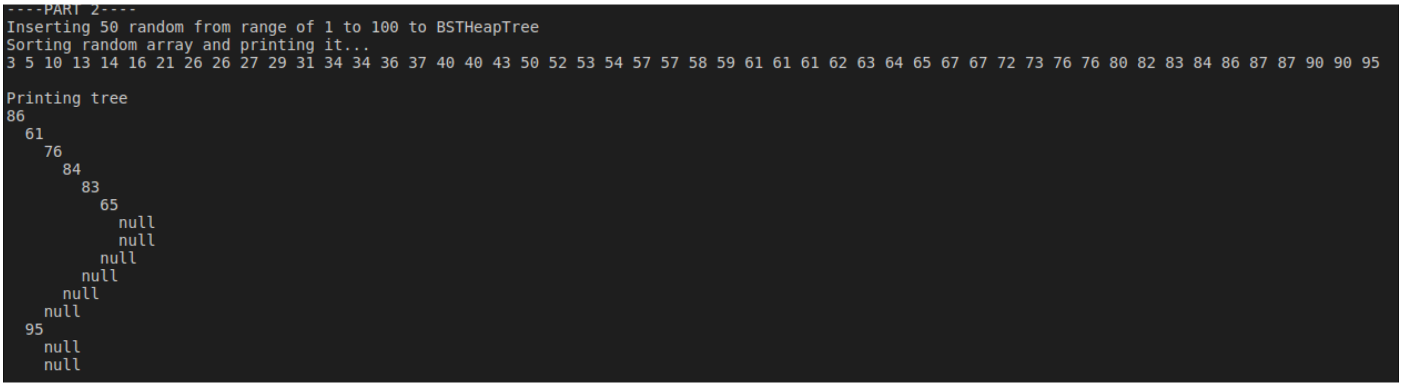
```
Removing 6th element from heap...
96
-75 (p:96)
-67 (p:96)
--66 (p:75)
--56 (p:75)
--55 (p:67)
--44 (p:67)
---44 (p:66)
---23 (p:66)
---23 (p:56)
---9 (p:56)
---3 (p:55)
---1 (p:55)
```



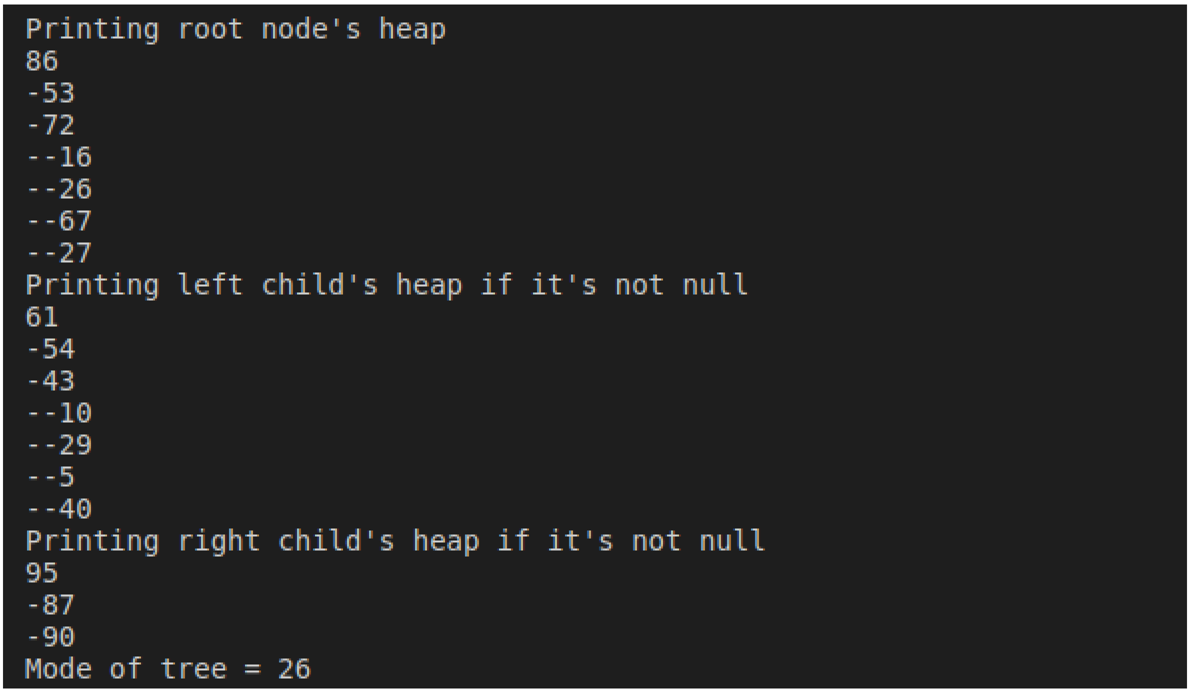
```
Removing 3rd element from heap...
96
-75 (p:96)
-66 (p:96)
--56 (p:75)
--55 (p:75)
--44 (p:66)
--44 (p:66)
---23 (p:56)
---23 (p:56)
---9 (p:55)
---3 (p:55)
---1 (p:44)
```



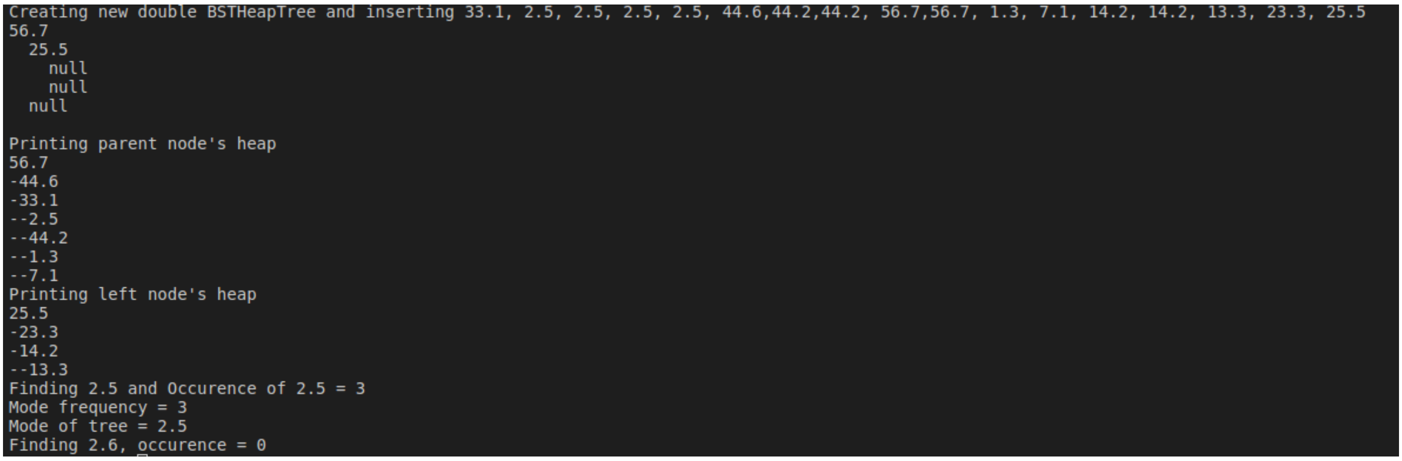
```
Creating iterator and printing next()
iter.next() = 96
iter.next() one more and calling iter.set(999)
iter.next() = 75
999
-96 (p:999)
-66 (p:999)
--56 (p:96)
--55 (p:96)
--44 (p:66)
--44 (p:66)
---23 (p:56)
---23 (p:56)
---9 (p:55)
---3 (p:55)
---1 (p:44)
iter.next() = 66
iter.set(7)
999
-96 (p:999)
-44 (p:999)
--56 (p:96)
--55 (p:96)
--7 (p:44)
--44 (p:44)
---23 (p:56)
---23 (p:56)
---9 (p:55)
---3 (p:55)
---1 (p:7)
```



```
----PART 2----
Inserting 50 random from range of 1 to 100 to BSTHeapTree
Sorting random array and printing it...
3 5 10 13 14 16 21 26 26 27 29 31 34 34 36 37 40 40 43 50 52 53 54 57 57 58 59 61 61 61 62 63 64 65 67 67 72 73 76 76 80 82 83 84 86 87 87 90 90 95

Printing tree
86
  61
    76
      84
        83
          65
            null
            null
          null
        null
      null
    null
  95
    null
    null
```

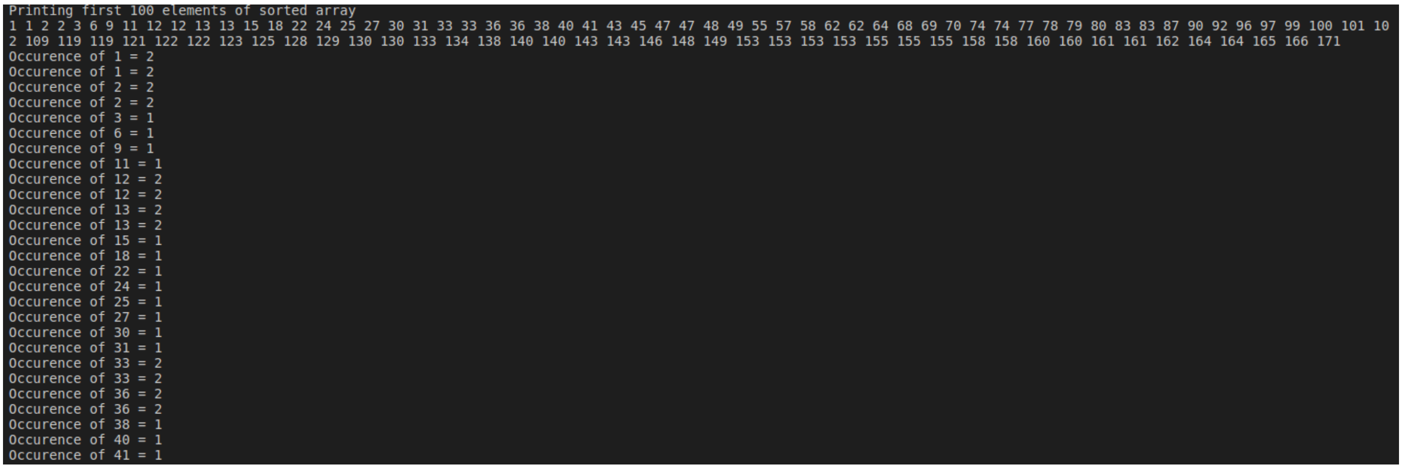


```
Printing root node's heap
86
-53
-72
--16
--26
--67
--27
Printing left child's heap if it's not null
61
-54
-43
--10
--29
--5
--40
Printing right child's heap if it's not null
95
-87
-90
Mode of tree = 26
```

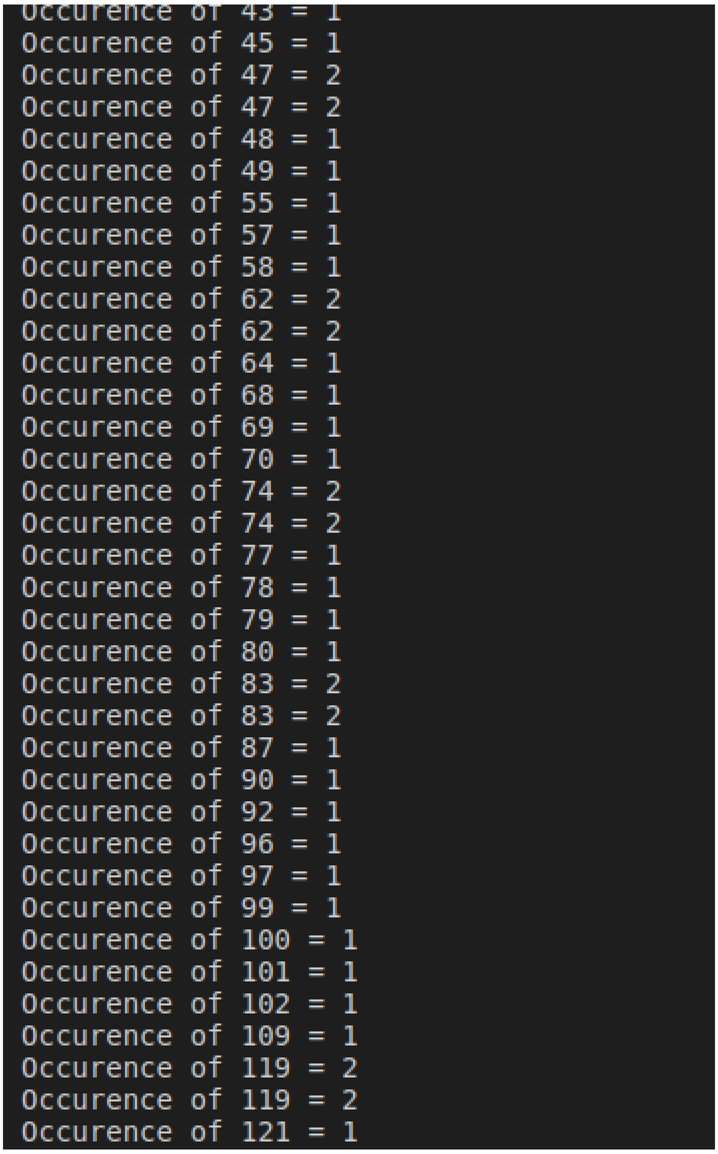


```
Creating new double BSTHeapTree and inserting 33.1, 2.5, 2.5, 2.5, 2.5, 44.6,44.2,44.2, 56.7,56.7, 1.3, 7.1, 14.2, 14.2, 13.3, 23.3, 25.5
56.7
  25.5
    null
    null
  null

Printing parent node's heap
56.7
-44.6
-33.1
--2.5
--44.2
--1.3
--7.1
Printing left node's heap
25.5
-23.3
-14.2
--13.3
Finding 2.5 and Occurence of 2.5 = 3
Mode frequency = 3
Mode of tree = 2.5
Finding 2.6, occurence = 0
```

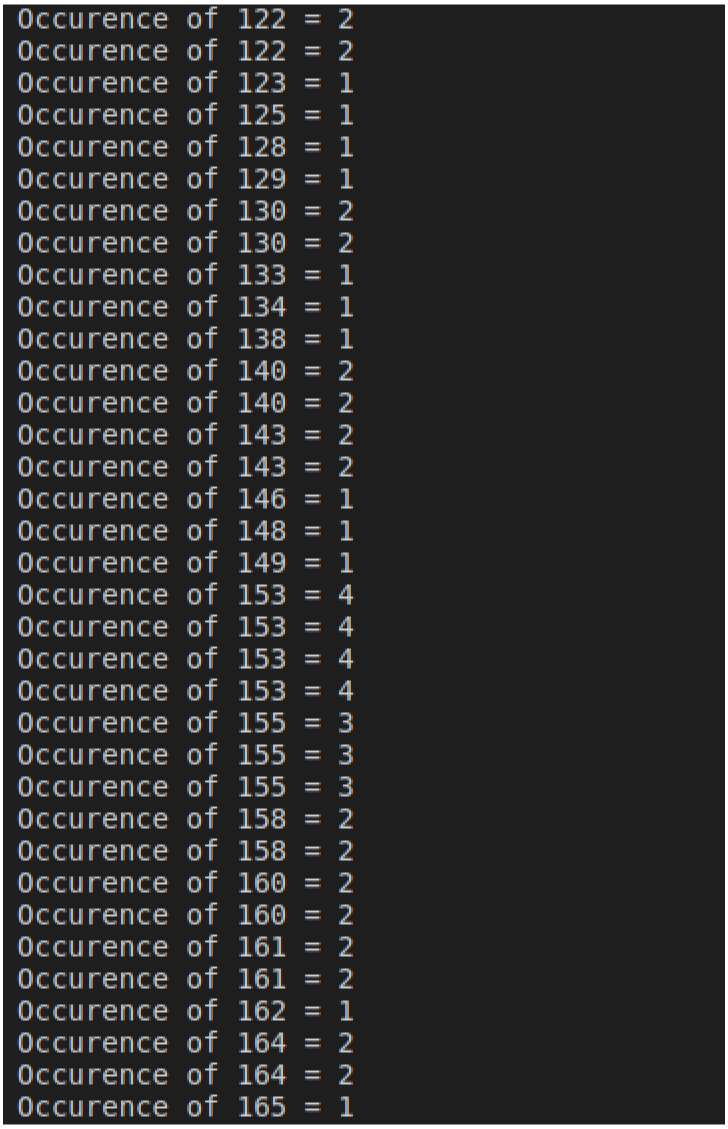
5.Running commands and results



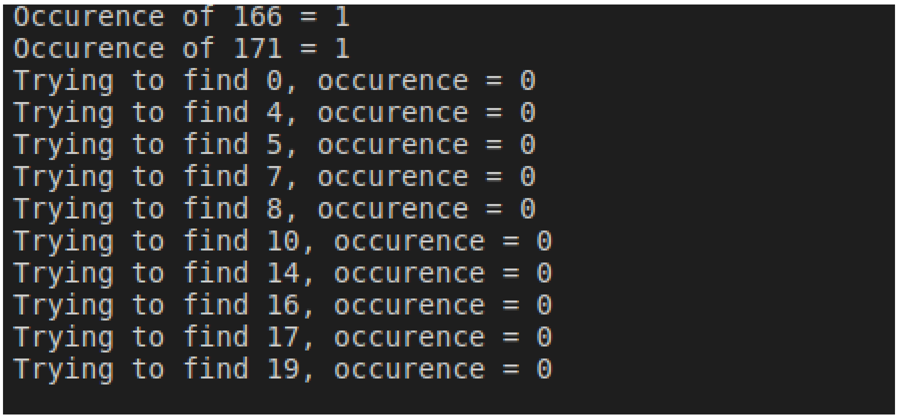
```
Printing first 100 elements of sorted array
1 1 2 2 3 6 9 11 12 12 13 13 15 18 22 24 25 27 30 31 33 33 36 36 38 40 41 43 45 47 47 48 49 55 57 58 62 62 64 68 69 70 74 74 77 78 79 80 83 83 87 90 92 96 97 99 100 101 10
2 109 119 119 121 122 122 123 125 128 129 130 130 133 134 138 140 140 143 143 146 148 149 153 153 153 153 155 155 155 158 158 160 160 161 161 162 164 164 165 166 171
Occurence of 1 = 2
Occurence of 1 = 2
Occurence of 2 = 2
Occurence of 2 = 2
Occurence of 3 = 1
Occurence of 6 = 1
Occurence of 9 = 1
Occurence of 11 = 1
Occurence of 12 = 2
Occurence of 12 = 2
Occurence of 13 = 2
Occurence of 13 = 2
Occurence of 15 = 1
Occurence of 18 = 1
Occurence of 22 = 1
Occurence of 24 = 1
Occurence of 25 = 1
Occurence of 27 = 1
Occurence of 30 = 1
Occurence of 31 = 1
Occurence of 33 = 2
Occurence of 33 = 2
Occurence of 36 = 2
Occurence of 36 = 2
Occurence of 38 = 1
Occurence of 40 = 1
Occurence of 41 = 1
```



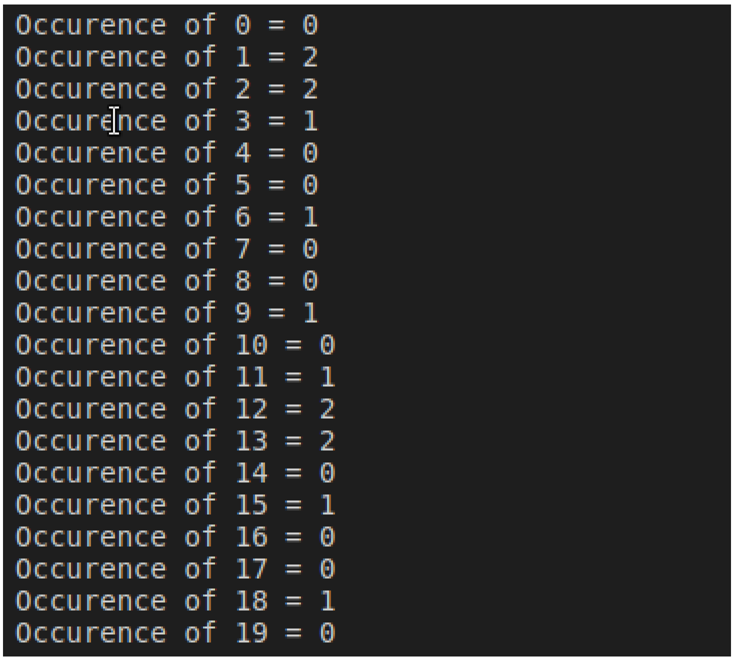
```
Occurence of 43 = 1
Occurence of 45 = 1
Occurence of 47 = 2
Occurence of 47 = 2
Occurence of 48 = 1
Occurence of 49 = 1
Occurence of 55 = 1
Occurence of 57 = 1
Occurence of 58 = 1
Occurence of 62 = 2
Occurence of 62 = 2
Occurence of 64 = 1
Occurence of 68 = 1
Occurence of 69 = 1
Occurence of 70 = 1
Occurence of 74 = 2
Occurence of 74 = 2
Occurence of 77 = 1
Occurence of 78 = 1
Occurence of 79 = 1
Occurence of 80 = 1
Occurence of 83 = 2
Occurence of 83 = 2
Occurence of 87 = 1
Occurence of 90 = 1
Occurence of 92 = 1
Occurence of 96 = 1
Occurence of 97 = 1
Occurence of 99 = 1
Occurence of 100 = 1
Occurence of 101 = 1
Occurence of 102 = 1
Occurence of 109 = 1
Occurence of 119 = 2
Occurence of 119 = 2
Occurence of 121 = 1
```



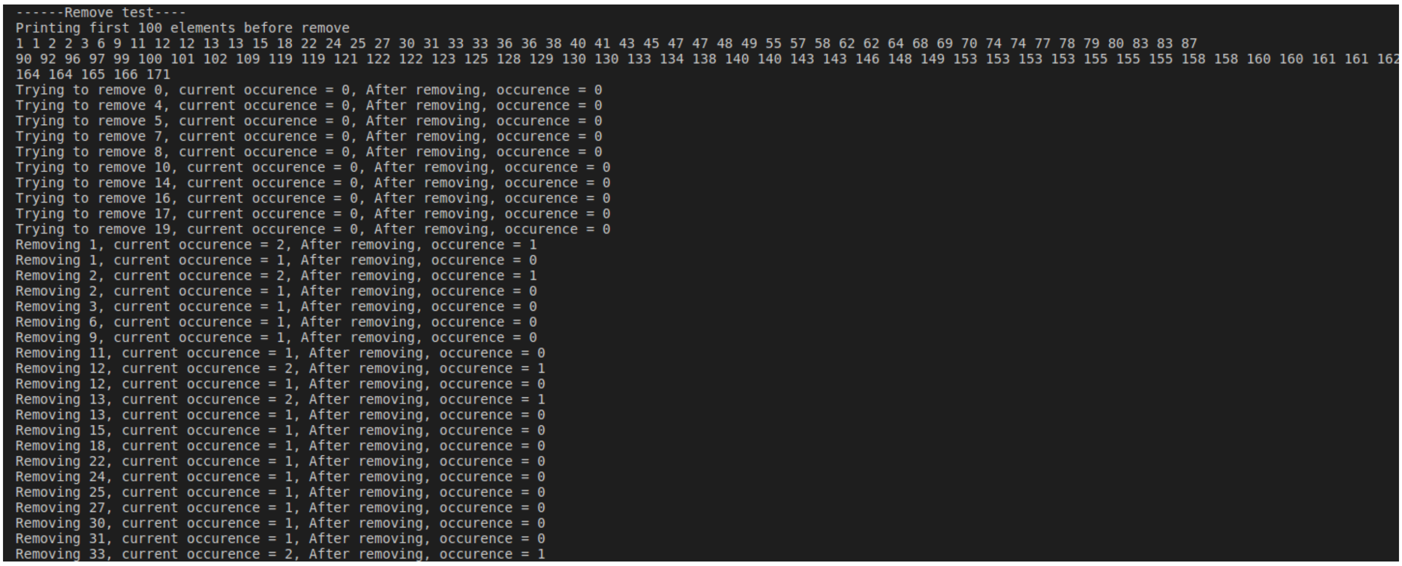
```
Occurence of 122 = 2
Occurence of 122 = 2
Occurence of 123 = 1
Occurence of 125 = 1
Occurence of 128 = 1
Occurence of 129 = 1
Occurence of 130 = 2
Occurence of 130 = 2
Occurence of 133 = 1
Occurence of 134 = 1
Occurence of 138 = 1
Occurence of 140 = 2
Occurence of 140 = 2
Occurence of 143 = 2
Occurence of 143 = 2
Occurence of 146 = 1
Occurence of 148 = 1
Occurence of 149 = 1
Occurence of 153 = 4
Occurence of 153 = 4
Occurence of 153 = 4
Occurence of 153 = 4
Occurence of 155 = 3
Occurence of 155 = 3
Occurence of 155 = 3
Occurence of 158 = 2
Occurence of 158 = 2
Occurence of 160 = 2
Occurence of 160 = 2
Occurence of 161 = 2
Occurence of 161 = 2
Occurence of 162 = 1
Occurence of 164 = 2
Occurence of 164 = 2
Occurence of 165 = 1
```



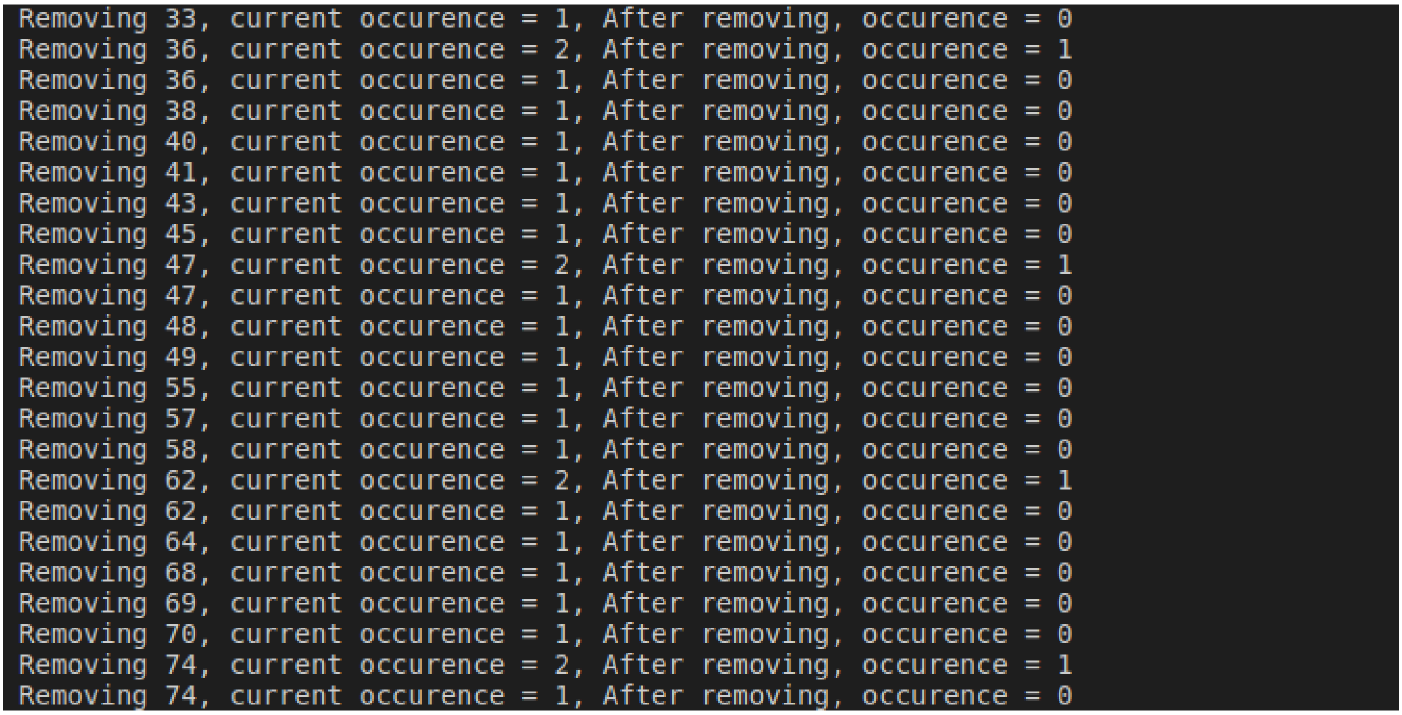
```
Occurence of 166 = 1
Occurence of 171 = 1
Trying to find 0, occurence = 0
Trying to find 4, occurence = 0
Trying to find 5, occurence = 0
Trying to find 7, occurence = 0
Trying to find 8, occurence = 0
Trying to find 10, occurence = 0
Trying to find 14, occurence = 0
Trying to find 16, occurence = 0
Trying to find 17, occurence = 0
Trying to find 19, occurence = 0
```



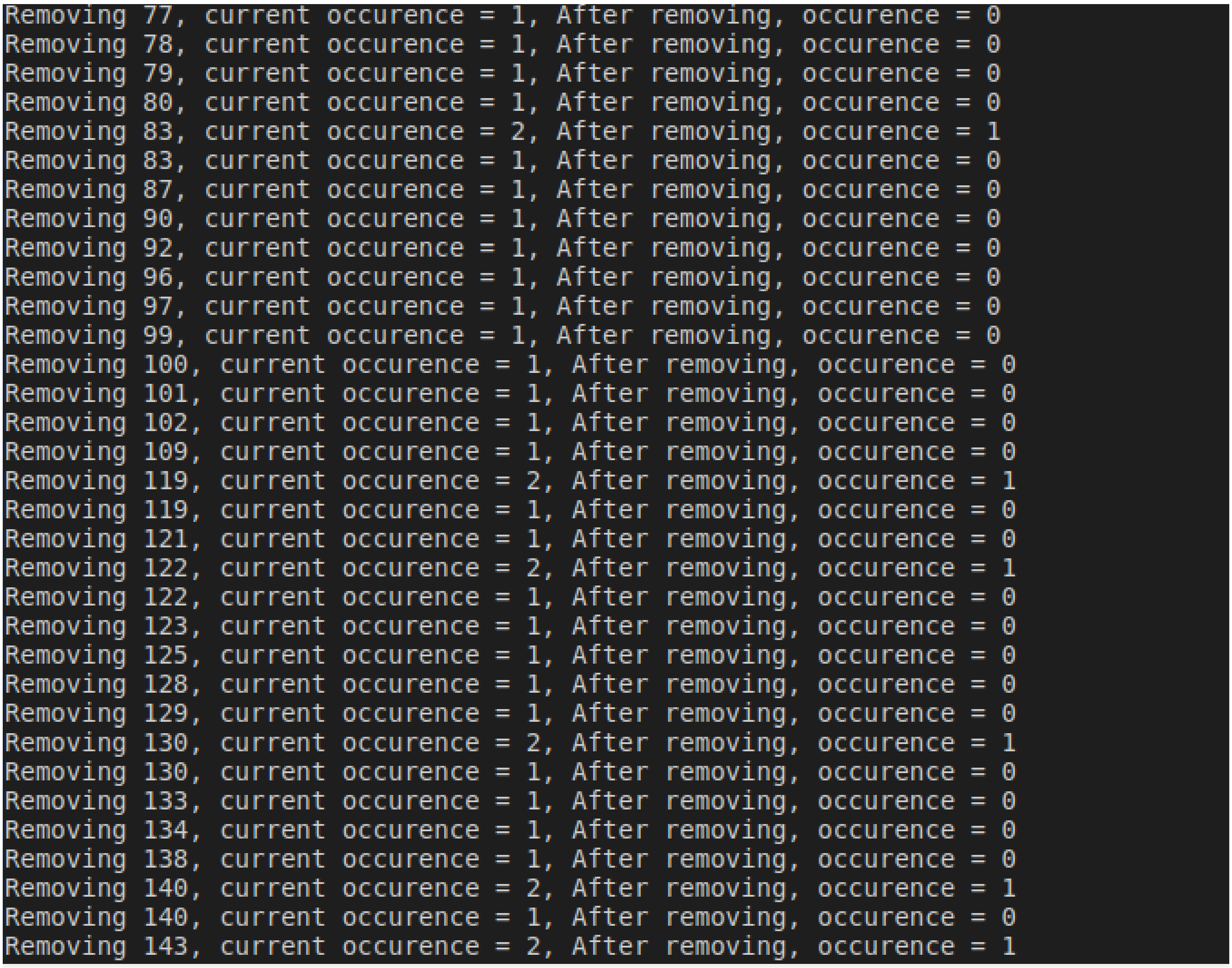
```
Occurence of 0 = 0
Occurence of 1 = 2
Occurence of 2 = 2
Occurence of 3 = 1
Occurence of 4 = 0
Occurence of 5 = 0
Occurence of 6 = 1
Occurence of 7 = 0
Occurence of 8 = 0
Occurence of 9 = 1
Occurence of 10 = 0
Occurence of 11 = 1
Occurence of 12 = 2
Occurence of 13 = 2
Occurence of 14 = 0
Occurence of 15 = 1
Occurence of 16 = 0
Occurence of 17 = 0
Occurence of 18 = 1
Occurence of 19 = 0
```



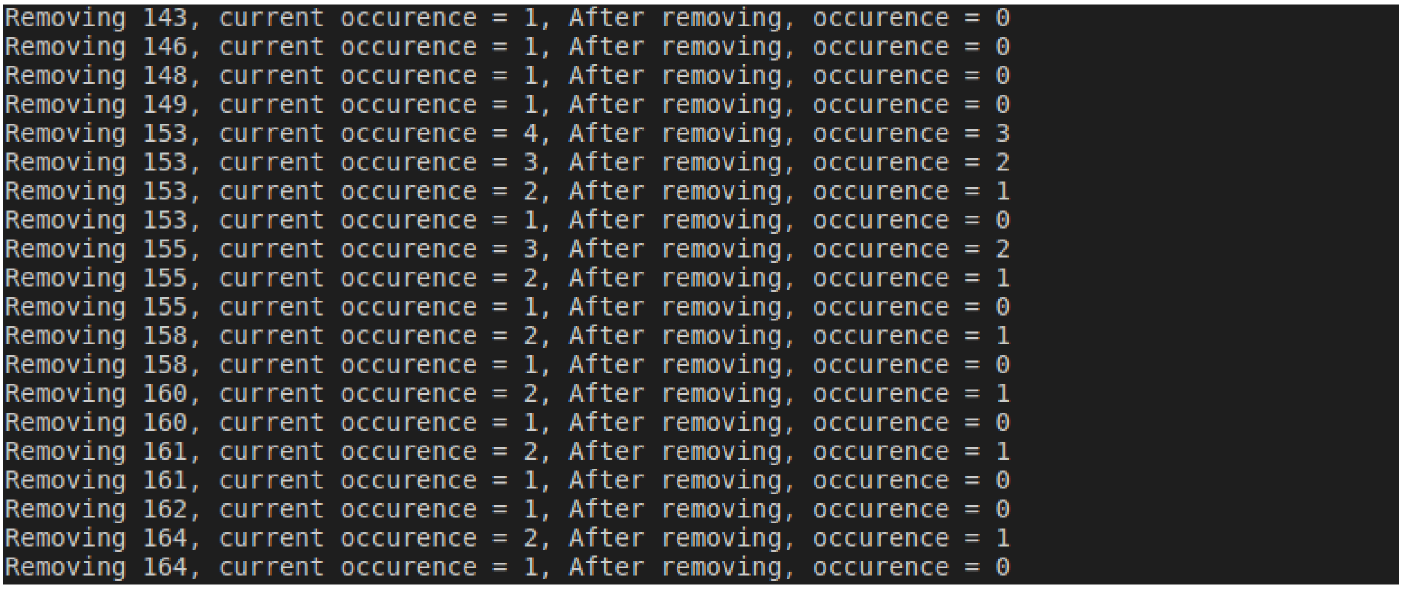
```
------Remove test----
Printing first 100 elements before remove
1 1 2 2 3 6 9 11 12 12 13 13 15 18 22 24 25 27 30 31 33 33 36 36 38 40 41 43 45 47 47 48 49 55 57 58 62 62 64 68 69 70 74 74 77 78 79 80 83 83 87
90 92 96 97 99 100 101 102 109 119 119 121 122 122 123 125 128 129 130 130 133 134 138 140 140 143 143 146 148 149 153 153 153 153 155 155 155 158 158 160 160 161 161 162
164 164 165 166 171
Trying to remove 0, current occurence = 0, After removing, occurence = 0
Trying to remove 4, current occurence = 0, After removing, occurence = 0
Trying to remove 5, current occurence = 0, After removing, occurence = 0
Trying to remove 7, current occurence = 0, After removing, occurence = 0
Trying to remove 8, current occurence = 0, After removing, occurence = 0
Trying to remove 10, current occurence = 0, After removing, occurence = 0
Trying to remove 14, current occurence = 0, After removing, occurence = 0
Trying to remove 16, current occurence = 0, After removing, occurence = 0
Trying to remove 17, current occurence = 0, After removing, occurence = 0
Trying to remove 19, current occurence = 0, After removing, occurence = 0
Removing 1, current occurence = 2, After removing, occurence = 1
Removing 1, current occurence = 1, After removing, occurence = 0
Removing 2, current occurence = 2, After removing, occurence = 1
Removing 2, current occurence = 1, After removing, occurence = 0
Removing 3, current occurence = 1, After removing, occurence = 0
Removing 6, current occurence = 1, After removing, occurence = 0
Removing 9, current occurence = 1, After removing, occurence = 0
Removing 11, current occurence = 1, After removing, occurence = 0
Removing 12, current occurence = 2, After removing, occurence = 1
Removing 12, current occurence = 1, After removing, occurence = 0
Removing 13, current occurence = 2, After removing, occurence = 1
Removing 13, current occurence = 1, After removing, occurence = 0
Removing 15, current occurence = 1, After removing, occurence = 0
Removing 18, current occurence = 1, After removing, occurence = 0
Removing 22, current occurence = 1, After removing, occurence = 0
Removing 24, current occurence = 1, After removing, occurence = 0
Removing 25, current occurence = 1, After removing, occurence = 0
Removing 27, current occurence = 1, After removing, occurence = 0
Removing 30, current occurence = 1, After removing, occurence = 0
Removing 31, current occurence = 1, After removing, occurence = 0
Removing 33, current occurence = 2, After removing, occurence = 1
```



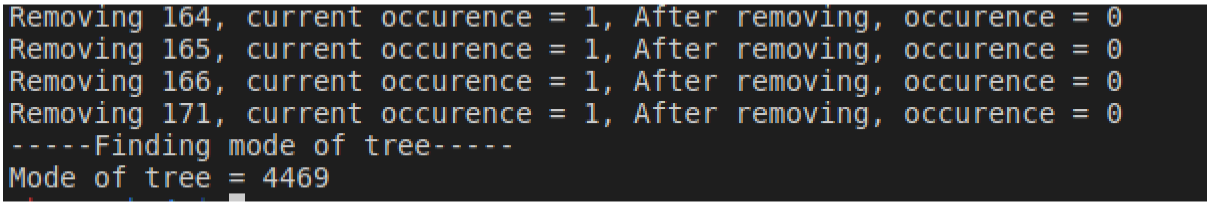
```
 Removing 33, current occurence = 1, After removing, occurence = 0
 Removing 36, current occurence = 2, After removing, occurence = 1
 Removing 36, current occurence = 1, After removing, occurence = 0
 Removing 38, current occurence = 1, After removing, occurence = 0
 Removing 40, current occurence = 1, After removing, occurence = 0
 Removing 41, current occurence = 1, After removing, occurence = 0
 Removing 43, current occurence = 1, After removing, occurence = 0
 Removing 45, current occurence = 1, After removing, occurence = 0
 Removing 47, current occurence = 2, After removing, occurence = 1
 Removing 47, current occurence = 1, After removing, occurence = 0
 Removing 48, current occurence = 1, After removing, occurence = 0
 Removing 49, current occurence = 1, After removing, occurence = 0
 Removing 55, current occurence = 1, After removing, occurence = 0
 Removing 57, current occurence = 1, After removing, occurence = 0
 Removing 58, current occurence = 1, After removing, occurence = 0
 Removing 62, current occurence = 2, After removing, occurence = 1
 Removing 62, current occurence = 1, After removing, occurence = 0
 Removing 64, current occurence = 1, After removing, occurence = 0
 Removing 68, current occurence = 1, After removing, occurence = 0
 Removing 69, current occurence = 1, After removing, occurence = 0
 Removing 70, current occurence = 1, After removing, occurence = 0
 Removing 74, current occurence = 2, After removing, occurence = 1
 Removing 74, current occurence = 1, After removing, occurence = 0
```



```
Removing 77, current occurence = 1, After removing, occurence = 0
Removing 78, current occurence = 1, After removing, occurence = 0
Removing 79, current occurence = 1, After removing, occurence = 0
Removing 80, current occurence = 1, After removing, occurence = 0
Removing 83, current occurence = 2, After removing, occurence = 1
Removing 83, current occurence = 1, After removing, occurence = 0
Removing 87, current occurence = 1, After removing, occurence = 0
Removing 90, current occurence = 1, After removing, occurence = 0
Removing 92, current occurence = 1, After removing, occurence = 0
Removing 96, current occurence = 1, After removing, occurence = 0
Removing 97, current occurence = 1, After removing, occurence = 0
Removing 99, current occurence = 1, After removing, occurence = 0
Removing 100, current occurence = 1, After removing, occurence = 0
Removing 101, current occurence = 1, After removing, occurence = 0
Removing 102, current occurence = 1, After removing, occurence = 0
Removing 109, current occurence = 1, After removing, occurence = 0
Removing 119, current occurence = 2, After removing, occurence = 1
Removing 119, current occurence = 1, After removing, occurence = 0
Removing 121, current occurence = 1, After removing, occurence = 0
Removing 122, current occurence = 2, After removing, occurence = 1
Removing 122, current occurence = 1, After removing, occurence = 0
Removing 123, current occurence = 1, After removing, occurence = 0
Removing 125, current occurence = 1, After removing, occurence = 0
Removing 128, current occurence = 1, After removing, occurence = 0
Removing 129, current occurence = 1, After removing, occurence = 0
Removing 130, current occurence = 2, After removing, occurence = 1
Removing 130, current occurence = 1, After removing, occurence = 0
Removing 133, current occurence = 1, After removing, occurence = 0
Removing 134, current occurence = 1, After removing, occurence = 0
Removing 138, current occurence = 1, After removing, occurence = 0
Removing 140, current occurence = 2, After removing, occurence = 1
Removing 140, current occurence = 1, After removing, occurence = 0
Removing 143, current occurence = 2, After removing, occurence = 1
```



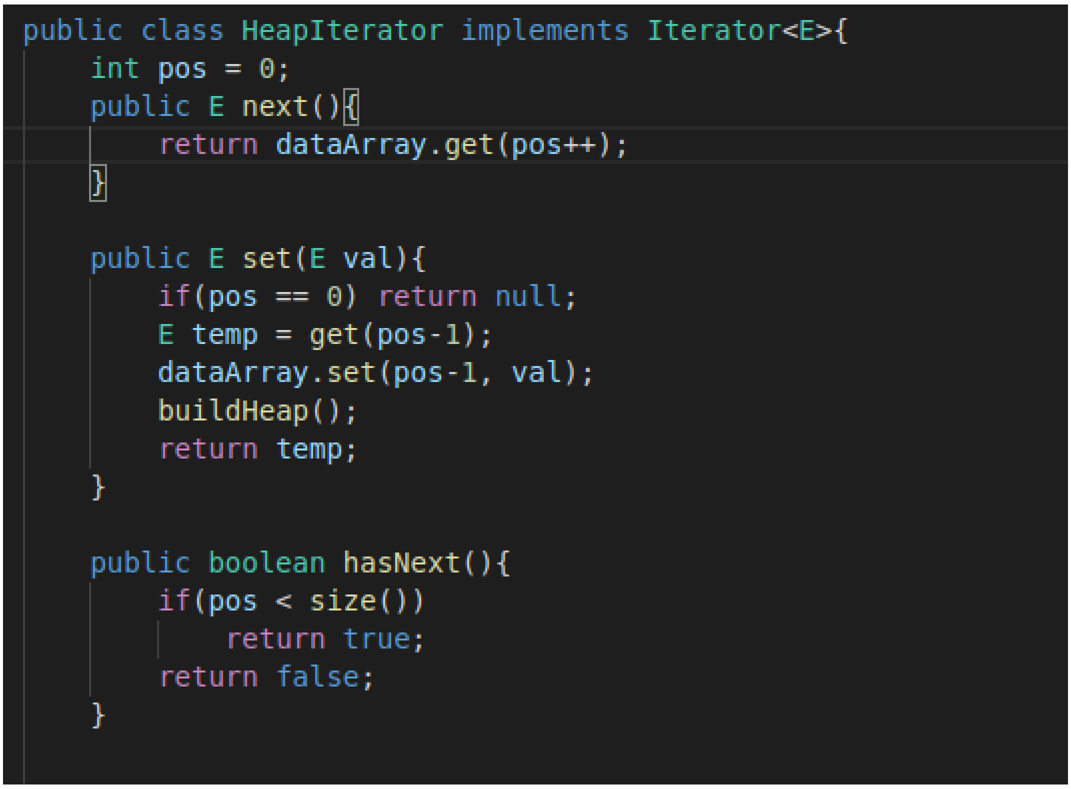
```
Removing 143, current occurence = 1, After removing, occurence = 0
Removing 146, current occurence = 1, After removing, occurence = 0
Removing 148, current occurence = 1, After removing, occurence = 0
Removing 149, current occurence = 1, After removing, occurence = 0
Removing 153, current occurence = 4, After removing, occurence = 3
Removing 153, current occurence = 3, After removing, occurence = 2
Removing 153, current occurence = 2, After removing, occurence = 1
Removing 153, current occurence = 1, After removing, occurence = 0
Removing 155, current occurence = 3, After removing, occurence = 2
Removing 155, current occurence = 2, After removing, occurence = 1
Removing 155, current occurence = 1, After removing, occurence = 0
Removing 158, current occurence = 2, After removing, occurence = 1
Removing 158, current occurence = 1, After removing, occurence = 0
Removing 160, current occurence = 2, After removing, occurence = 1
Removing 160, current occurence = 1, After removing, occurence = 0
Removing 161, current occurence = 2, After removing, occurence = 1
Removing 161, current occurence = 1, After removing, occurence = 0
Removing 162, current occurence = 1, After removing, occurence = 0
Removing 164, current occurence = 2, After removing, occurence = 1
Removing 164, current occurence = 1, After removing, occurence = 0
```



```
Removing 164, current occurence = 1, After removing, occurence = 0
Removing 165, current occurence = 1, After removing, occurence = 0
Removing 166, current occurence = 1, After removing, occurence = 0
Removing 171, current occurence = 1, After removing, occurence = 0
-----Finding mode of tree-----
Mode of tree = 4469
```

PART3

Heap class

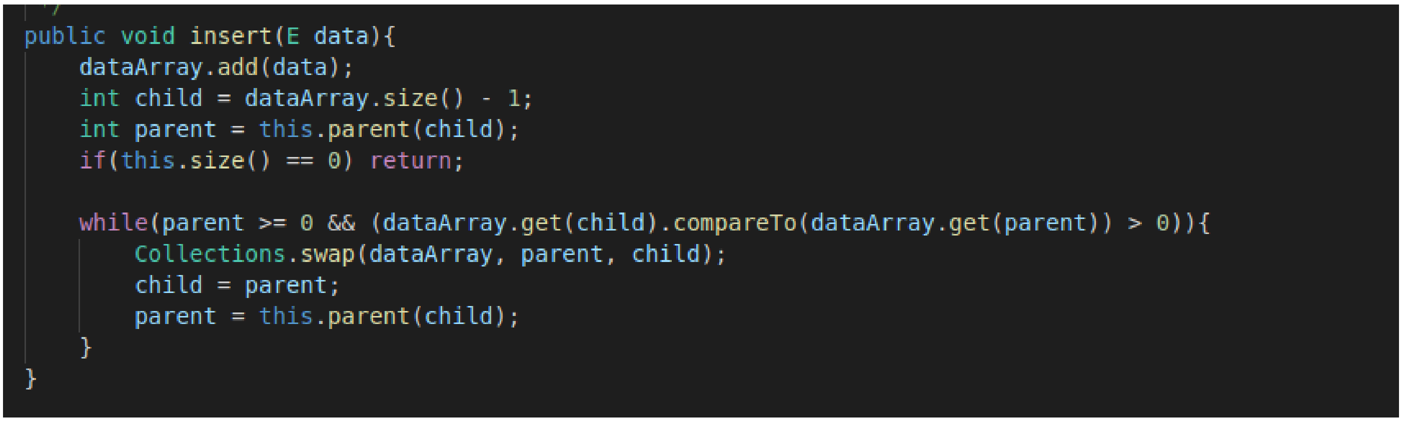


```
public class HeapIterator implements Iterator<E>{
    int pos = 0;
    public E next(){
        return dataArray.get(pos++);
    }

    public E set(E val){
        if(pos == 0) return null;
        E temp = get(pos-1);
        dataArray.set(pos-1, val);
        buildHeap();
        return temp;
    }

    public boolean hasNext(){
        if(pos < size())
            return true;
        return false;
    }
```

next() = theta(1)

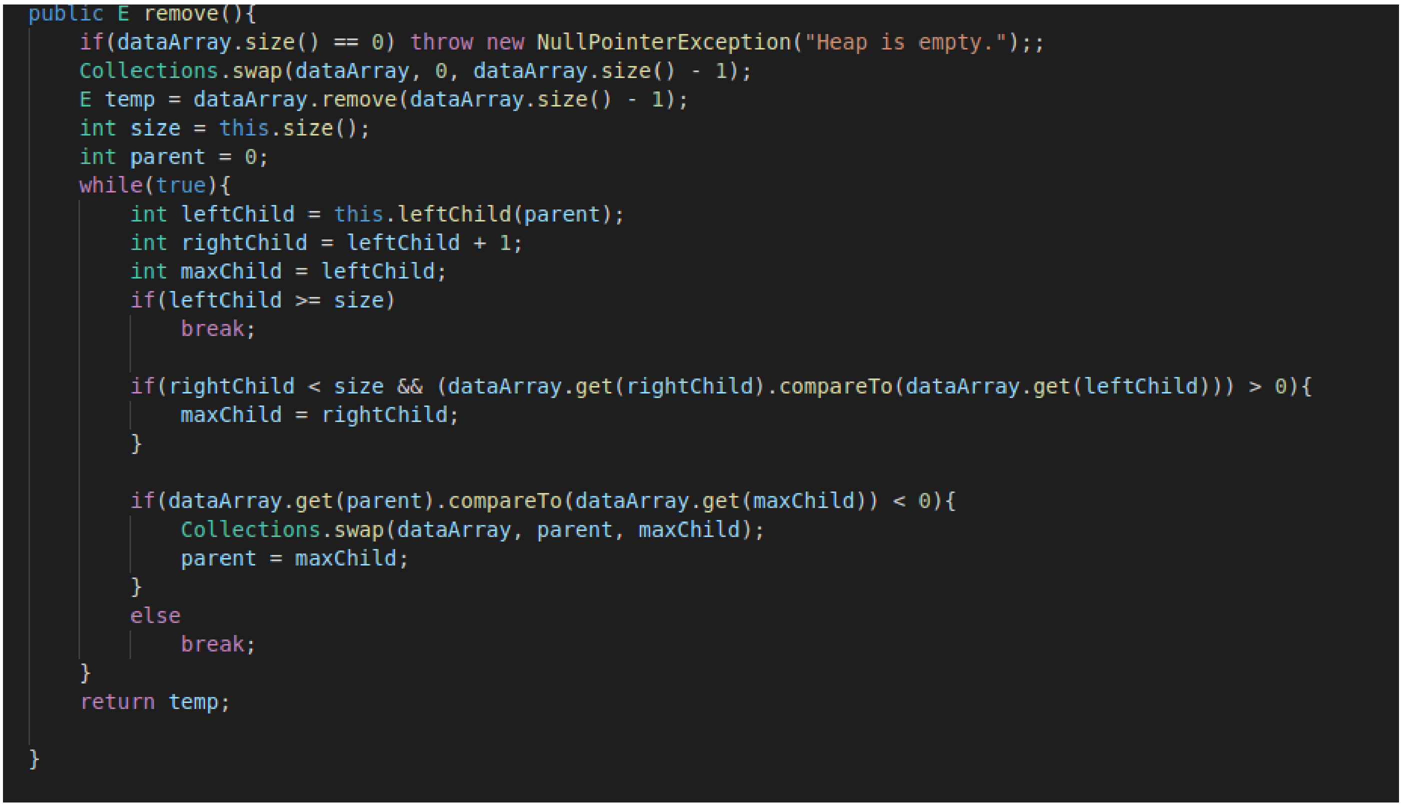
set = O(n)

hasNext = O(1)



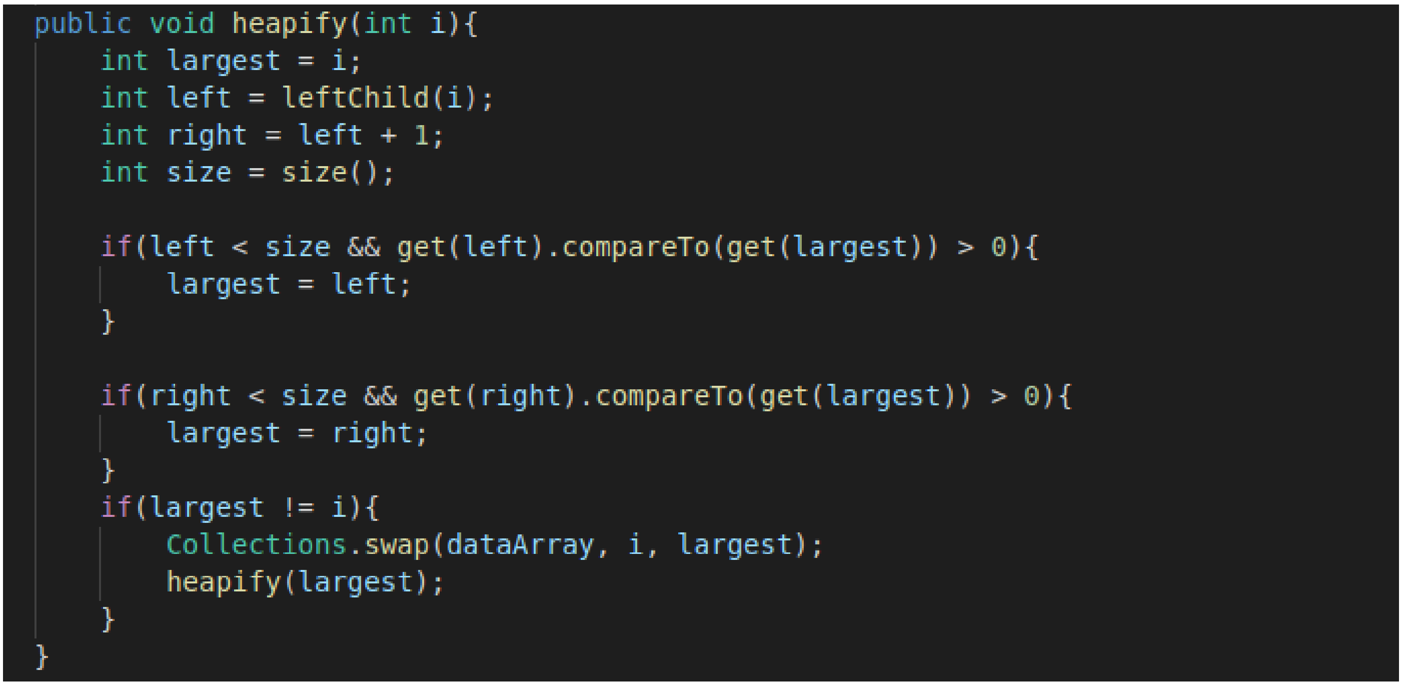
```
public void insert(E data){
    dataArray.add(data);
    int child = dataArray.size() - 1;
    int parent = this.parent(child);
    if(this.size() == 0) return;

    while(parent >= 0 && (dataArray.get(child).compareTo(dataArray.get(parent)) > 0)){
        Collections.swap(dataArray, parent, child);
        child = parent;
        parent = this.parent(child);
    }
}
```

O(n)



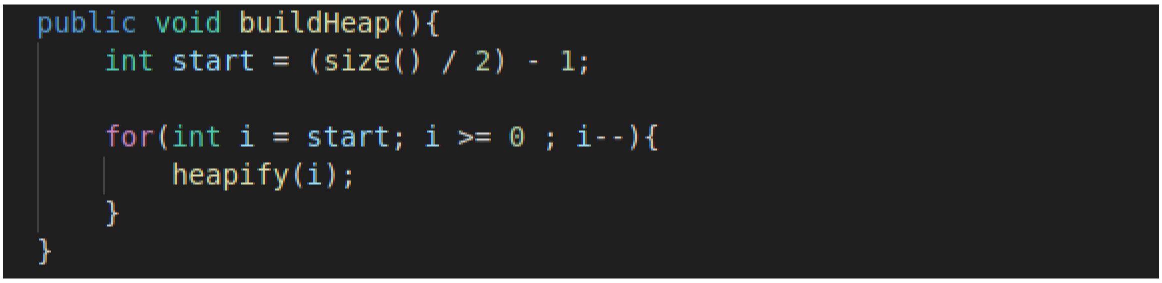
```
public E remove(){
    if(dataArray.size() == 0) throw new NullPointerException("Heap is empty.");;
    Collections.swap(dataArray, 0, dataArray.size() - 1);
    E temp = dataArray.remove(dataArray.size() - 1);
    int size = this.size();
    int parent = 0;
    while(true){
        int leftChild = this.leftChild(parent);
        int rightChild = leftChild + 1;
        int maxChild = leftChild;
        if(leftChild >= size)
            break;

        if(rightChild < size && (dataArray.get(rightChild).compareTo(dataArray.get(leftChild))) > 0){
            maxChild = rightChild;
        }

        if(dataArray.get(parent).compareTo(dataArray.get(maxChild)) < 0){
            Collections.swap(dataArray, parent, maxChild);
            parent = maxChild;
        }
        else
            break;
    }
    return temp;

}
```

O(n)



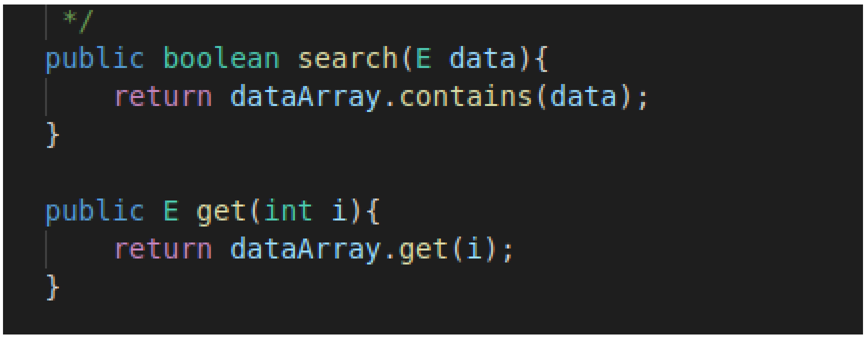
```
public void heapify(int i){
    int largest = i;
    int left = leftChild(i);
    int right = left + 1;
    int size = size();

    if(left < size && get(left).compareTo(get(largest)) > 0){
        largest = left;
    }

    if(right < size && get(right).compareTo(get(largest)) > 0){
        largest = right;
    }
    if(largest != i){
        Collections.swap(dataArray, i, largest);
        heapify(largest);
    }
}
```

O(n)



```
public void buildHeap(){
    int start = (size() / 2) - 1;

    for(int i = start; i >= 0 ; i--){
        heapify(i);
    }
}
```

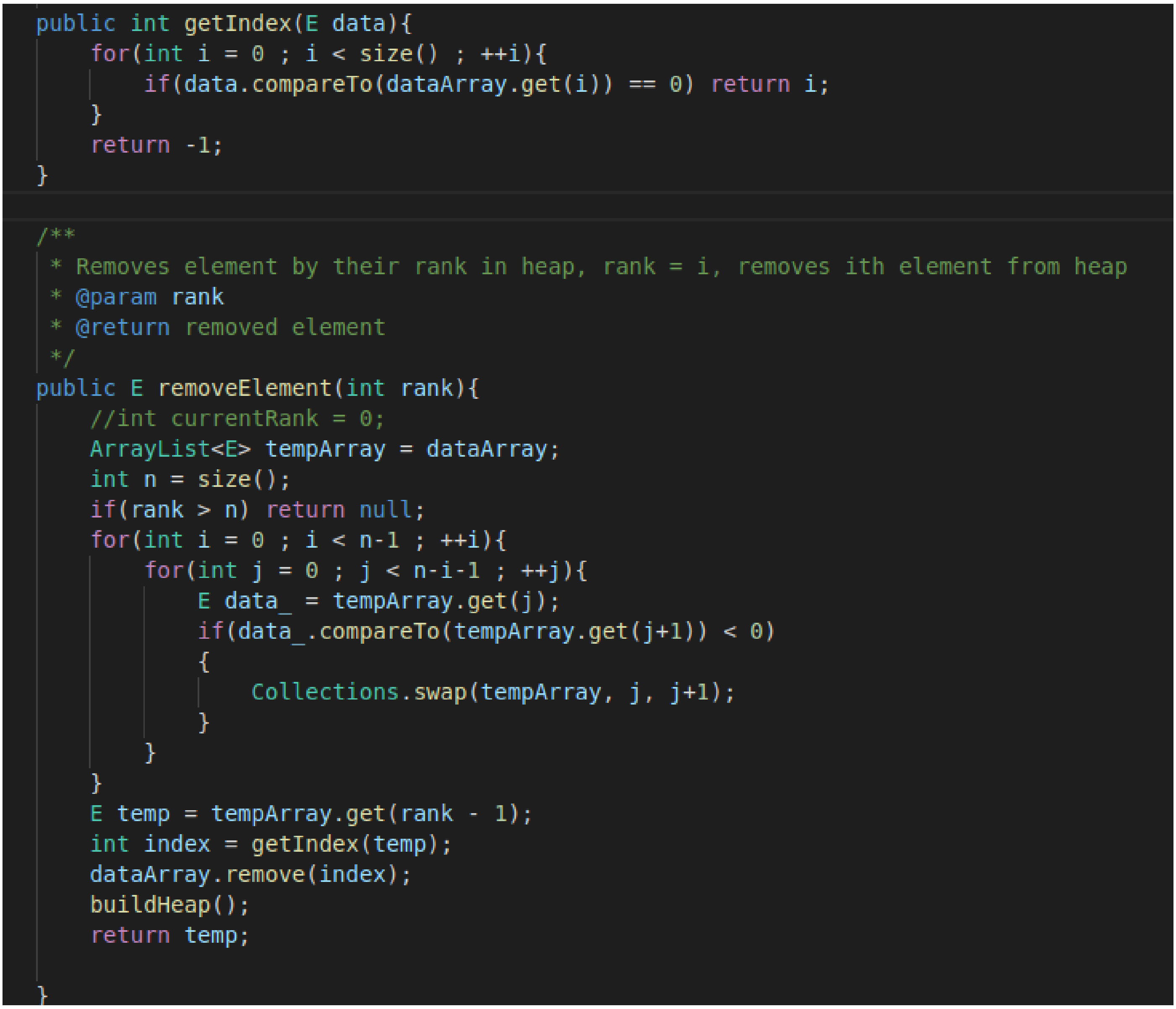
O(n^2)



```
  */
 public boolean search(E data){
     return dataArray.contains(data);
 }

 public E get(int i){
     return dataArray.get(i);
 }
```

search->O(n)

get->theta(1)



```java
public int getIndex(E data){
    for(int i = 0 ; i < size() ; ++i){
        if(data.compareTo(dataArray.get(i)) == 0) return i;
    }
    return -1;
}

/**
 * Removes element by their rank in heap, rank = i, removes ith element from heap
 * @param rank
 * @return removed element
 */
public E removeElement(int rank){
    //int currentRank = 0;
    ArrayList<E> tempArray = dataArray;
    int n = size();
    if(rank > n) return null;
    for(int i = 0 ; i < n-1 ; ++i){
        for(int j = 0 ; j < n-i-1 ; ++j){
            E data_ = tempArray.get(j);
            if(data_.compareTo(tempArray.get(j+1)) < 0)
            {
                Collections.swap(tempArray, j, j+1);
            }
        }
    }
    E temp = tempArray.get(rank - 1);
    int index = getIndex(temp);
    dataArray.remove(index);
    buildHeap();
    return temp;

}
```

getIndex->theta(n)

removeElement->O(n^2)