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

Mehmet Acar
1801042095

1-SYSTEM REQUIREMENTS

Functional Requirements

1-In Heap class, search, merge, and remove functions are functional requirements. In MyIterator class, set function is functional requirement. In BSTHeapTree class, add, remove and find class are functional requirements.

2-The program must print an error message when applying search operation on empty heap.

3-The program must print an error message when applying merge operation on empty heap1 and heap2.

4-The program must print an error message when applying remove operation if i^{th} largest element number exceeds heap size.

5-The program must print an error message when applying remove operation if heap is empty.

6-The program must print an error message if wanted to removed element can not be find in BST or wanted to removed element's amount in BST is equal to 0.

7-The program must print an error message if searched element can not find in BST.

Non-functional Requirements

1-Recoverability

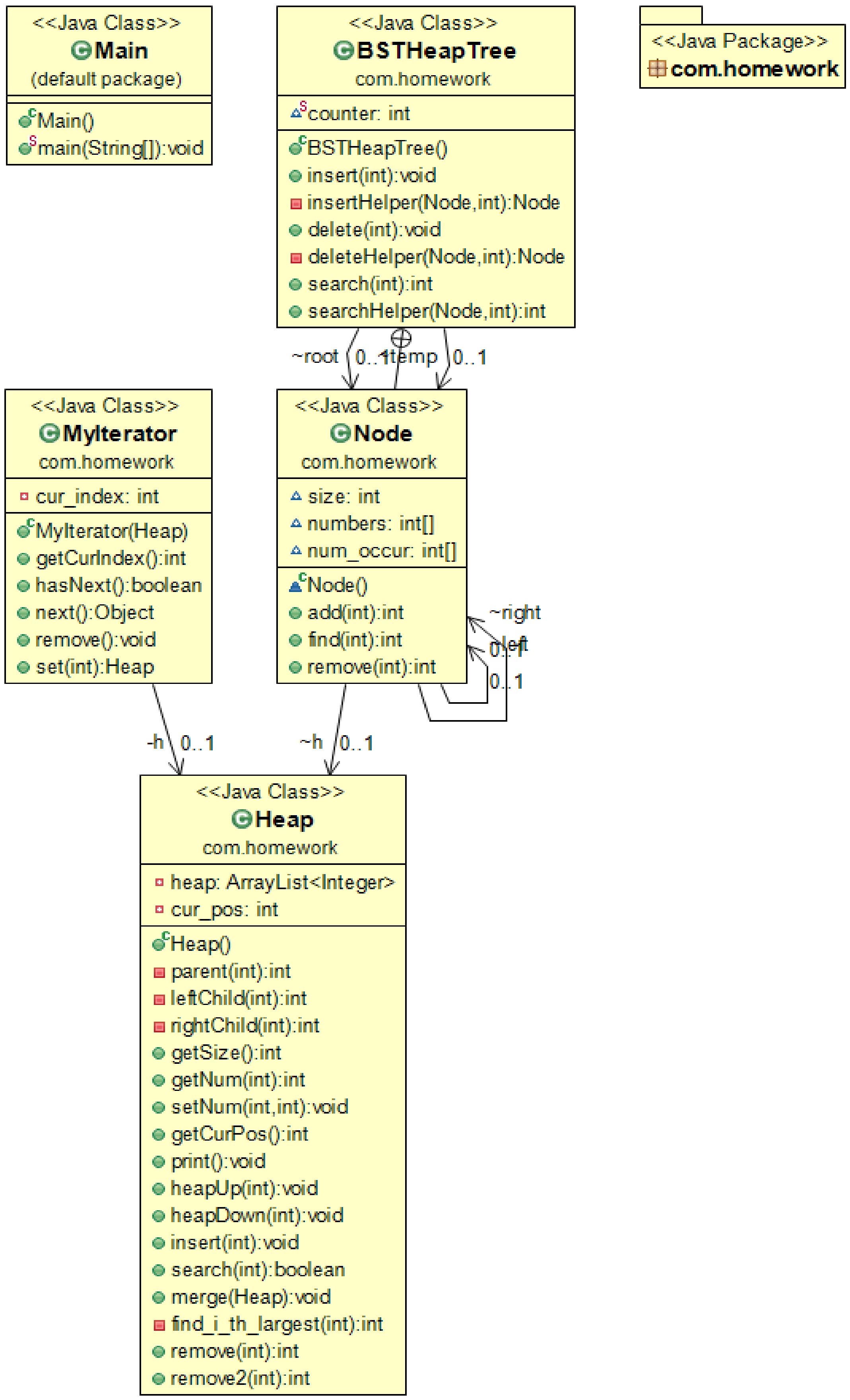
2-Maintainability

3-Manageability

4-Security

5-Usability

2-CLASS DIAGRAM



3-PROBLEM SOLUTION APPROACH

Part 1

1-Identify The Problem

Problem is implementing features for Heap structure.

2-Gather Information

Heap structure will have search for an element, merge with another heap, removing ith largest element from the Heap and extend the Iterator class by adding a method to set the value features.

3-Iterate Potential Solutions

I created heap class and I hold heap elements in arraylist. In order to provide heap features, I checked heap both parent and child direction with heapUp and heapDown methods. Also, I created MyIterator class in order to implement set method.

Part 2

1-Identify The Problem

Problem is implementing a BSTHeapTree class that keeps the elements in a Binary Search Tree where the nodesstore max-Heap with a maximum depth of 2.

2-Gather Information

BSTHeapTree will have node in the heap holds two data: a value and the number of occurrences of the value, movement on BST is based on values at the root nodes of the heap, if the heap at a node of BST is full and a new number still needs to be added, a new BST node should be created as the left or the right child of the BST node, remove operation and the mode is the value in the BSTHeapTree that occurs most frequently features.

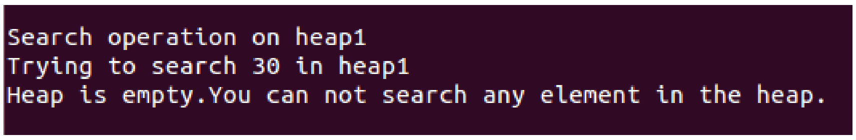
3-Iterate Potential Solutions

I created BSTHeapTree class and I hold heaps in Node class. Also, when I connect these nodes to each other and applying wanted methods in BST, I take care of BST feature.

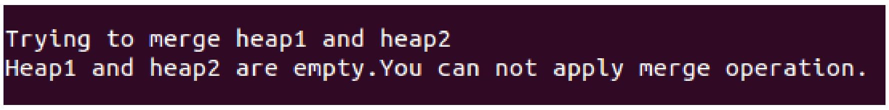
4-TEST CASES

PART 1

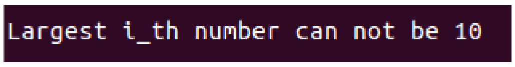
1-If heap is empty when searching an element


```
Search operation on heap1
Trying to search 30 in heap1
Heap is empty.You can not search any element in the heap.
```

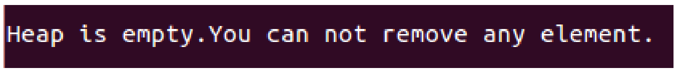
2-If trying to merge heap1 and heap2 when both heaps are empty


```
Trying to merge heap1 and heap2
Heap1 and heap2 are empty.You can not apply merge operation.
```

3-If largest i^th number exceeds heap size when applying remove operation

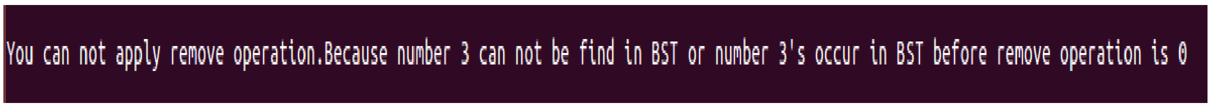

```
Largest i_th number can not be 10
```

4- If heap is empty when applying remove operation

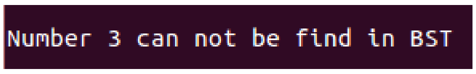

```
Heap is empty.You can not remove any element.
```

PART 2

1-If wanted to removed element can not be find in BST or wanted to removed element's amount in BST is equal to 0


```
You can not apply remove operation.Because number 3 can not be find in BST or number 3's occur in BST before remove operation is 0
```

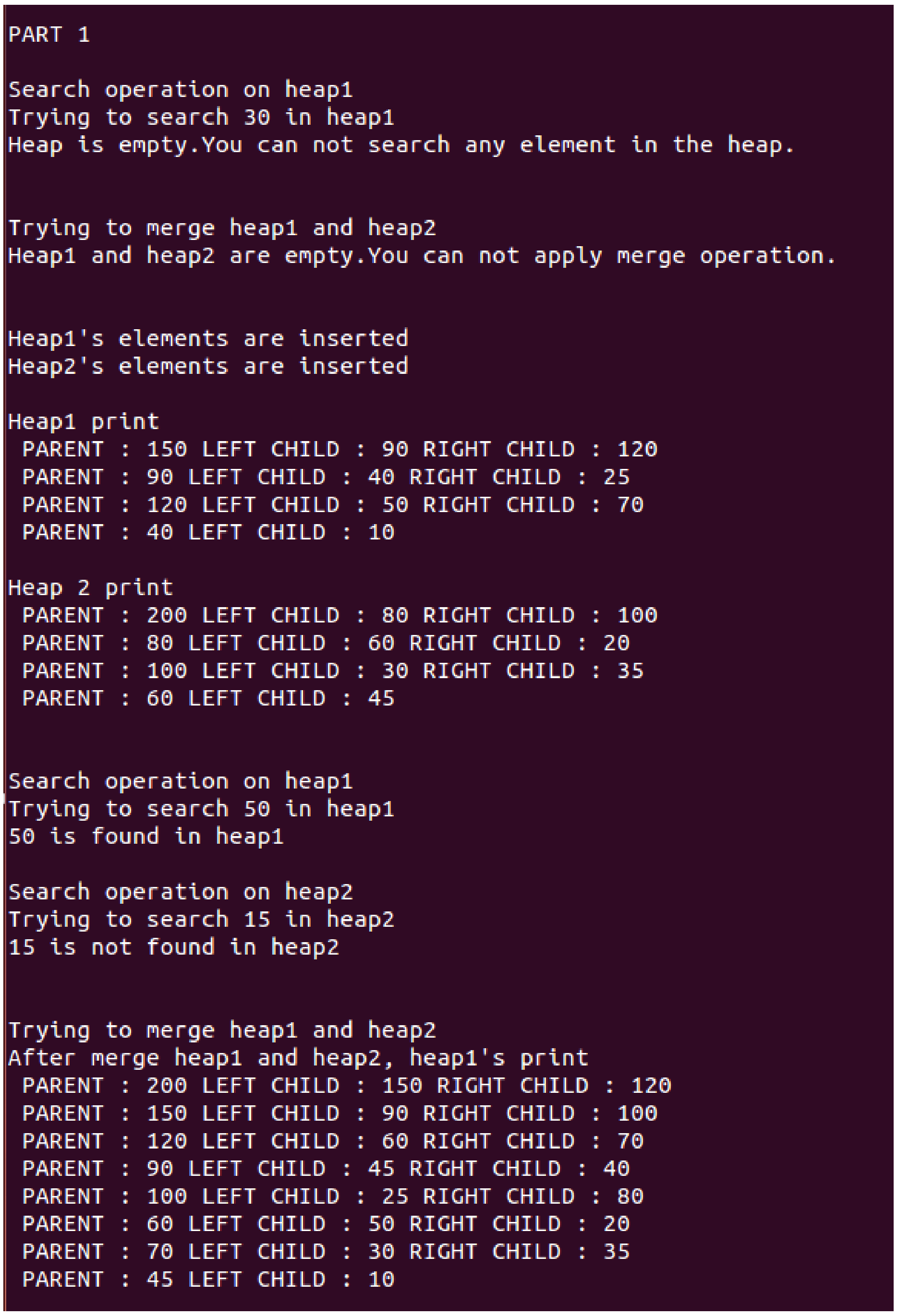
2-If searched element can not find in BST


```
Number 3 can not be find in BST
```

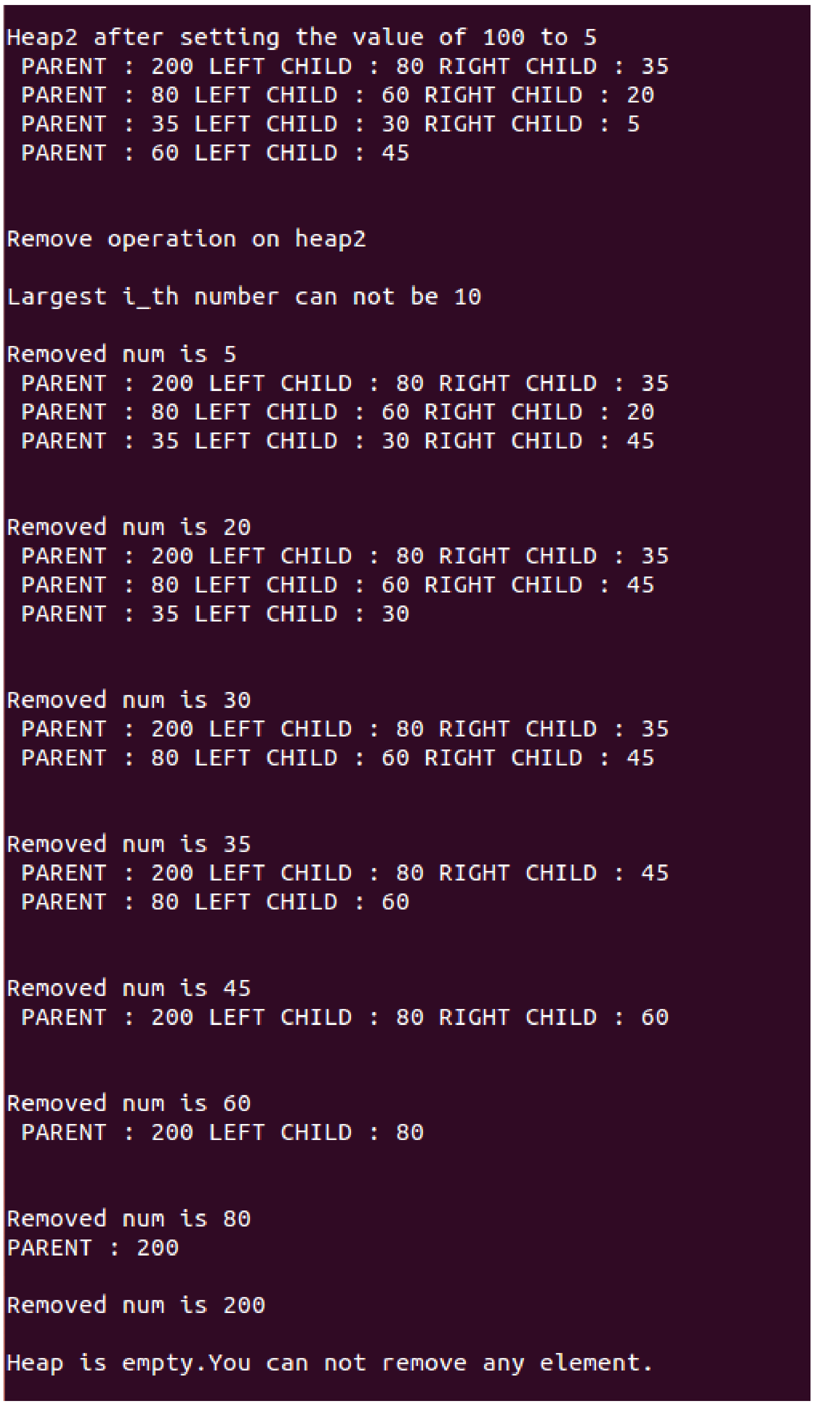
5-RUNNING AND RESULTS

I used max heap rule in both parts.

PART 1

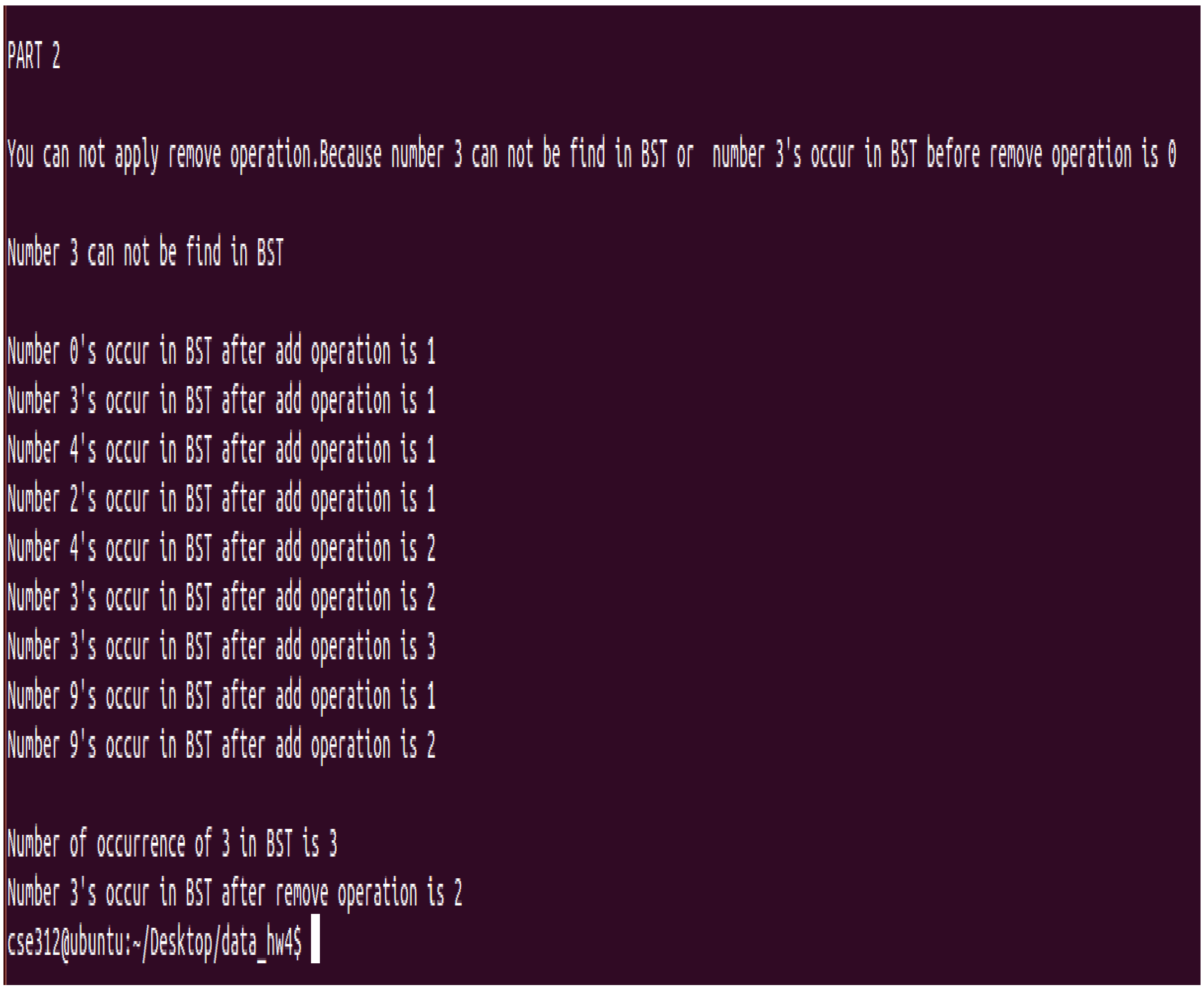


```
PART 1

Search operation on heap1
Trying to search 30 in heap1
Heap is empty.You can not search any element in the heap.


Trying to merge heap1 and heap2
Heap1 and heap2 are empty.You can not apply merge operation.


Heap1's elements are inserted
Heap2's elements are inserted

Heap1 print
 PARENT : 150 LEFT CHILD : 90 RIGHT CHILD : 120
 PARENT : 90 LEFT CHILD : 40 RIGHT CHILD : 25
 PARENT : 120 LEFT CHILD : 50 RIGHT CHILD : 70
 PARENT : 40 LEFT CHILD : 10

Heap 2 print
 PARENT : 200 LEFT CHILD : 80 RIGHT CHILD : 100
 PARENT : 80 LEFT CHILD : 60 RIGHT CHILD : 20
 PARENT : 100 LEFT CHILD : 30 RIGHT CHILD : 35
 PARENT : 60 LEFT CHILD : 45


Search operation on heap1
Trying to search 50 in heap1
50 is found in heap1

Search operation on heap2
Trying to search 15 in heap2
15 is not found in heap2


Trying to merge heap1 and heap2
After merge heap1 and heap2, heap1's print
 PARENT : 200 LEFT CHILD : 150 RIGHT CHILD : 120
 PARENT : 150 LEFT CHILD : 90 RIGHT CHILD : 100
 PARENT : 120 LEFT CHILD : 60 RIGHT CHILD : 70
 PARENT : 90 LEFT CHILD : 45 RIGHT CHILD : 40
 PARENT : 100 LEFT CHILD : 25 RIGHT CHILD : 80
 PARENT : 60 LEFT CHILD : 50 RIGHT CHILD : 20
 PARENT : 70 LEFT CHILD : 30 RIGHT CHILD : 35
 PARENT : 45 LEFT CHILD : 10
```



```
Heap2 after setting the value of 100 to 5
 PARENT : 200 LEFT CHILD : 80 RIGHT CHILD : 35
 PARENT : 80 LEFT CHILD : 60 RIGHT CHILD : 20
 PARENT : 35 LEFT CHILD : 30 RIGHT CHILD : 5
 PARENT : 60 LEFT CHILD : 45


Remove operation on heap2

Largest i_th number can not be 10

Removed num is 5
 PARENT : 200 LEFT CHILD : 80 RIGHT CHILD : 35
 PARENT : 80 LEFT CHILD : 60 RIGHT CHILD : 20
 PARENT : 35 LEFT CHILD : 30 RIGHT CHILD : 45


Removed num is 20
 PARENT : 200 LEFT CHILD : 80 RIGHT CHILD : 35
 PARENT : 80 LEFT CHILD : 60 RIGHT CHILD : 45
 PARENT : 35 LEFT CHILD : 30


Removed num is 30
 PARENT : 200 LEFT CHILD : 80 RIGHT CHILD : 35
 PARENT : 80 LEFT CHILD : 60 RIGHT CHILD : 45


Removed num is 35
 PARENT : 200 LEFT CHILD : 80 RIGHT CHILD : 45
 PARENT : 80 LEFT CHILD : 60


Removed num is 45
 PARENT : 200 LEFT CHILD : 80 RIGHT CHILD : 60


Removed num is 60
 PARENT : 200 LEFT CHILD : 80


Removed num is 80
PARENT : 200

Removed num is 200

Heap is empty.You can not remove any element.
```

PART 2



```
PART 2

You can not apply remove operation.Because number 3 can not be find in BST or  number 3's occur in BST before remove operation is 0

Number 3 can not be find in BST

Number 0's occur in BST after add operation is 1
Number 3's occur in BST after add operation is 1
Number 4's occur in BST after add operation is 1
Number 2's occur in BST after add operation is 1
Number 4's occur in BST after add operation is 2
Number 3's occur in BST after add operation is 2
Number 3's occur in BST after add operation is 3
Number 9's occur in BST after add operation is 1
Number 9's occur in BST after add operation is 2

Number of occurrence of 3 in BST is 3
Number 3's occur in BST after remove operation is 2
cse312@ubuntu:~/Desktop/data_hw4$
```

6-Time Complexity

Heap.java Methods Time Complexities

```
public Heap() {
	heap= new ArrayList<Integer>();
}
```

T(n)= $\Theta(1)$

```
private int parent(int child_index) {
	return (child_index-1)/2;
}
```

T(n)= θ(1)

```java
private int leftChild(int parent_index) {
	return (parent_index*2)+1;
}
```

T(n)= θ(1)

```java
private int rightChild(int parent_index) {
	return (parent_index*2)+2;
}
```

T(n)= θ(1)

```java
public int getSize() {
	return heap.size();
}
```

T(n)= θ(1)

```java
public int getNum(int index) {
	return heap.get(index);
}
```

T(n)= θ(1)

```java
public void setNum(int index,int val) {
	heap.set(index, val);
}
```

T(n)= θ(1)

```java
public int getCurPos() {
	return cur_pos;
}
```

T(n)= θ(1)

```java
public void print() {

	if(heap.size()==1) {
		System.out.print("PARENT : " + heap.get(0));
	}

	else {
	  for (int i = 0; i < heap.size() / 2; i++) {

	       System.out.print(" PARENT : " + heap.get(i));

	       if(leftChild(i)<heap.size()) {
	       System.out.print(" LEFT CHILD : " + heap.get(leftChild(i)));
	       }
	       if(rightChild(i)<heap.size()) {
	       System.out.print(" RIGHT CHILD : " + heap.get(rightChild(i)));
	       }
	       System.out.println();
```

```
                }
            }

    }
```

T(n)= O(n)

```
    public void heapUp(int cur_index) {

            while(cur_index!=0 && heap.get(cur_index)>heap.get(parent(cur_index))) {
                    Collections.swap(heap,cur_index,parent(cur_index));
                    cur_index=(cur_index-1)/2;
            }
            cur_pos=cur_index;
    }
```

T(n)= O(n)

```
    public void heapDown(int cur_index) {

            boolean control=true;
            int biggestChildIndex=0;

            while (control==true) {

            if(leftChild(cur_index)>=heap.size() && rightChild(cur_index)>=heap.size())
    {
            control=false;
    }

            else {
                    if(leftChild(cur_index)<heap.size()) {
                            biggestChildIndex = leftChild(cur_index);
                     }

                    if(rightChild(cur_index)<heap.size() &&
    heap.get(rightChild(cur_index))>heap.get(leftChild(cur_index))) {
                        biggestChildIndex = rightChild(cur_index);
                    }

                    if(heap.get(cur_index)<heap.get(biggestChildIndex)) {
                            Collections.swap(heap, cur_index, biggestChildIndex);
                            cur_index=biggestChildIndex;
                    }

                    else {
                            control=false;
                    }

            }

      }
                    cur_pos=cur_index;
    }
```

T(n)= O(n)

```
public void insert(int val) {

	heap.add(val);
	int cur_index=heap.size()-1;
	heapUp(cur_index);

}
```

T(n)= O(n)

```
public boolean search(int target) throws Exception{

	if(heap.isEmpty())
	  throw new Exception("Heap is empty.You can not search any element in the
heap.");
	else
	  return heap.contains(target);

}
```

T(n)= O(n)

```
public void merge(Heap heap2) throws Exception {

		if(getSize()==0 && heap2.getSize()==0)
			throw new Exception("Heap1 and heap2 are empty.You can not
apply merge operation.");

		for(int i=0;i<heap2.getSize();i++) {
			insert(heap2.getNum(i));
		}

	}
```

T(n)= O(n^2)

```
private int find_i_th_largest(int i) {

	ArrayList<Integer> dest= new ArrayList<Integer>();
```

```
        for(int j=0;j<heap.size();j++) {
                dest.add(heap.get(j));
        }

        Collections.sort(dest);
        return dest.get(dest.size()-i);

}
```

T(n)= O(n*log(n))

```
public int remove(int largest_index) throws Exception{

        if(heap.isEmpty()) {
                throw new Exception("Heap is empty.You can not remove any element.");
        }

        else if(heap.size()-largest_index<0) {
                throw new Exception("Largest i_th number can not be " +
largest_index);
                }

        else if(heap.size()==1) {
                int removed_num=heap.get(0);
                heap.remove(0);
                return removed_num;
        }

        else {

                int removed_num=find_i_th_largest(largest_index);
                int removed_index=heap.indexOf(removed_num);
                heap.remove(removed_index);
                heap.add(removed_index, heap.get(heap.size()-1));
                heap.remove(heap.size()-1);

                int cur_index=removed_index;
                heapDown(cur_index);
                return removed_num;

             }

        }
```

T(n)= O(n*log(n))

```
public int remove2(int index) throws Exception{

        if(heap.isEmpty()) {
                throw new Exception("Heap is empty.You can not remove any element.");
        }
```

```java
      else if(heap.size()==1) {
            int removed_num=heap.get(0);
            heap.remove(0);
            return removed_num;
      }

      else {

            int removed_num=heap.get(0);
            heap.remove(0);
            heap.add(0, heap.get(heap.size()-1));
            heap.remove(heap.size()-1);

            int cur_index=0;
            heapDown(cur_index);
            return removed_num;

      }

}
```

T(n)= O(n)

MyIterator.java Methods Time Complexities

```java
public MyIterator(Heap heap) {
      int i;
      h= new Heap();
      for(i=0;i<heap.getSize();i++) {
            h.insert(heap.getNum(i));
      }
      cur_index=0;
}
```

T(n)= O(n^2)

```java
public int getCurIndex() {
      return cur_index;
}
```

T(n)= θ(1)

```java
@Override
public boolean hasNext() {
      return cur_index < h.getSize() && h.getSize()!=0;
}
```

T(n)= θ(1)

```java
@Override
public Object next() {
      Object next_num=  h.getNum(cur_index);
```

```
        cur_index++;
        return next_num;
}

T(n)= θ(1)


@Override
public void remove() {
        throw new UnsupportedOperationException();
}

T(n)= θ(1)


public Heap set(int val) {
        h.setNum(cur_index-1,val);
        return h;
}

T(n)= θ(1)
```