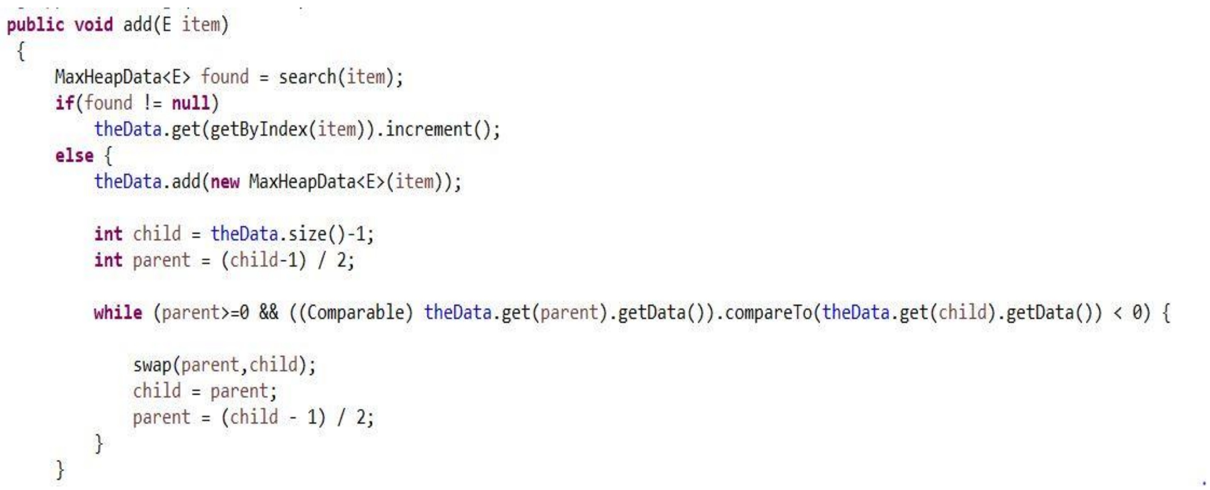


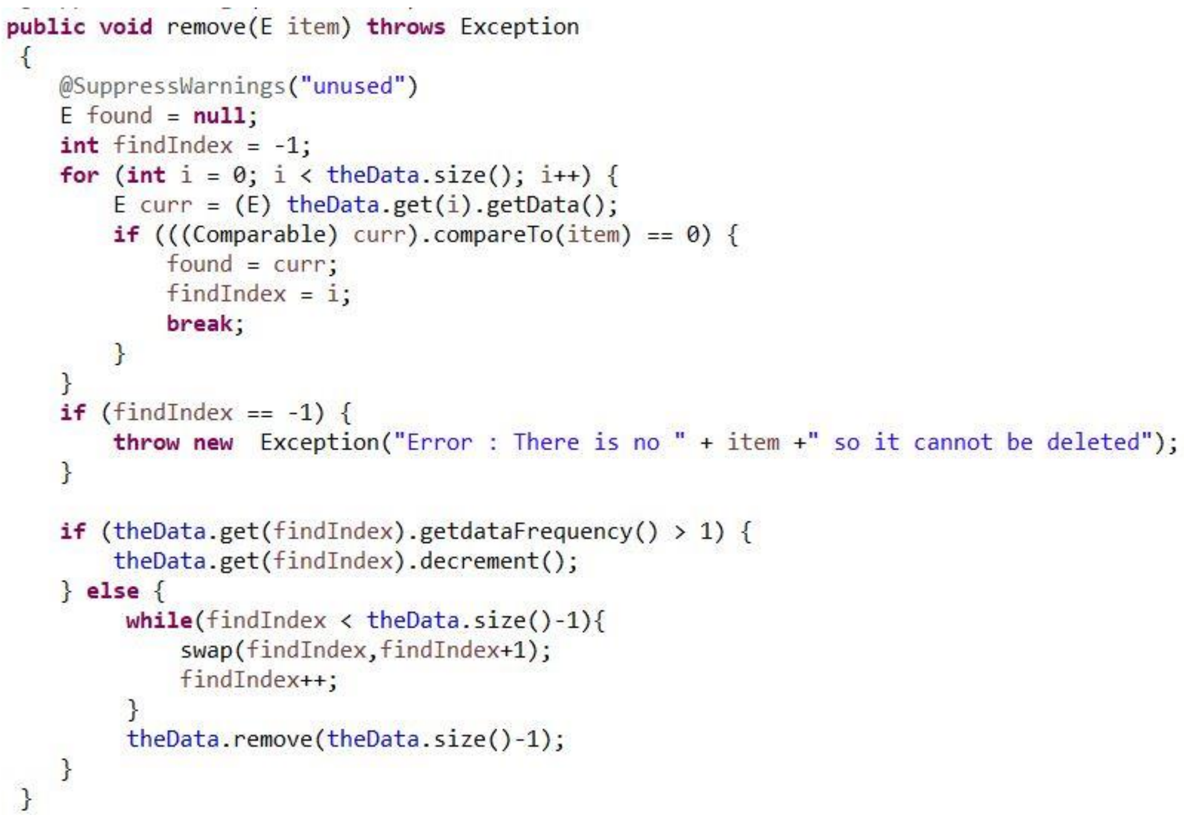
```
public void add(E item)
 {
     MaxHeapData<E> found = search(item);
     if(found != null)
         theData.get(getByIndex(item)).increment();
     else {
         theData.add(new MaxHeapData<E>(item));

         int child = theData.size()-1;
         int parent = (child-1) / 2;

         while (parent>=0 && ((Comparable) theData.get(parent).getData()).compareTo(theData.get(child).getData()) < 0) {

             swap(parent,child);
             child = parent;
             parent = (child - 1) / 2;
         }
     }
```

Search() = Tb(n)= θ(1) , T_W =O(n)

Tb(n)= θ(1)

While loop = $T_B = \theta(1)$, T_W = θ(logn) ,T(n)=O(logn)

T(n) =O(logn+n)



```
public void remove(E item) throws Exception
 {
    @SuppressWarnings("unused")
    E found = null;
    int findIndex = -1;
    for (int i = 0; i < theData.size(); i++) {
        E curr = (E) theData.get(i).getData();
        if (((Comparable) curr).compareTo(item) == 0) {
            found = curr;
            findIndex = i;
            break;
        }
    }
    if (findIndex == -1) {
        throw new  Exception("Error : There is no " + item +" so it cannot be deleted");
    }

    if (theData.get(findIndex).getdataFrequency() > 1) {
        theData.get(findIndex).decrement();
    } else {
         while(findIndex < theData.size()-1){
             swap(findIndex,findIndex+1);
             findIndex++;
         }
         theData.remove(theData.size()-1);
    }
 }
```

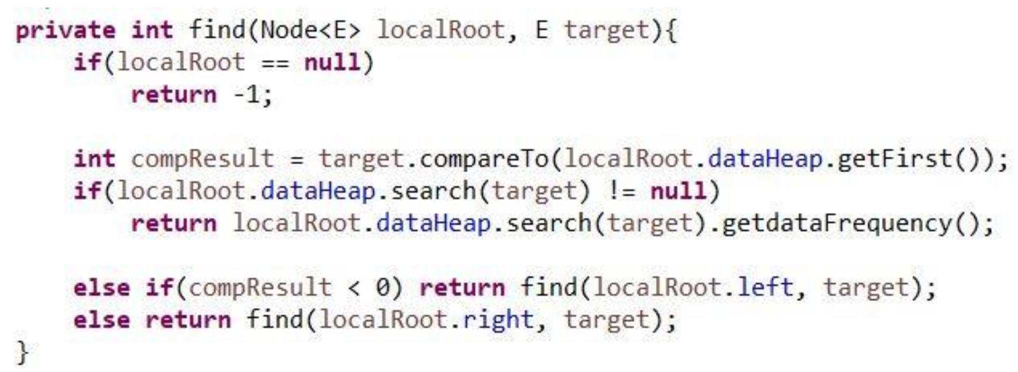
For=T(n)= Tb(n)= θ(1) , T_w =O(n)

İf = = θ(1)

Else = T(n)=O(logn)

Tb(n)= **θ(1) , T_w =θ (n.logn)**

T(n)=O(n.logn)



```
private int find(Node<E> localRoot, E target){
    if(localRoot == null)
        return -1;

    int compResult = target.compareTo(localRoot.dataHeap.getFirst());
    if(localRoot.dataHeap.search(target) != null)
        return localRoot.dataHeap.search(target).getdataFrequency();

    else if(compResult < 0) return find(localRoot.left, target);
    else return find(localRoot.right, target);
}
```

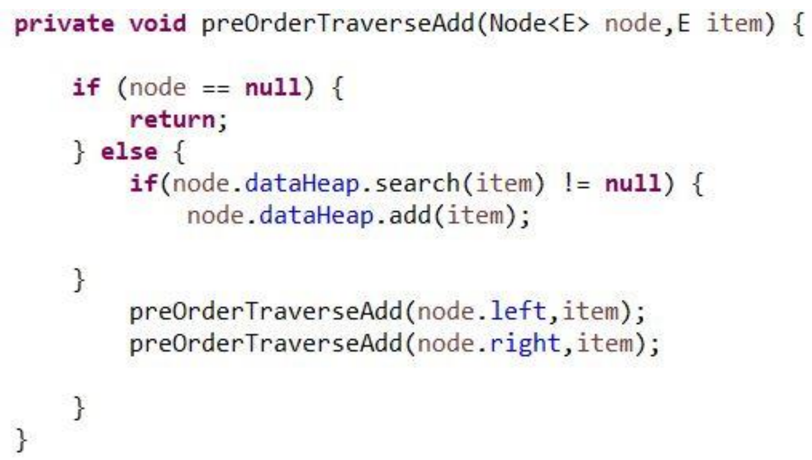
Find traversal = T(n)=O(logn)

Search() = Tb(n)= θ(1) , T_w =O(n)

Tb(n)= θ(1)

Tw(n) =O(n)

Tavr(n)=O(logn)



```
private void preOrderTraverseAdd(Node<E> node,E item) {

    if (node == null) {
        return;
    } else {
        if(node.dataHeap.search(item) != null) {
            node.dataHeap.add(item);

    }
        preOrderTraverseAdd(node.left,item);
        preOrderTraverseAdd(node.right,item);

    }
}
```

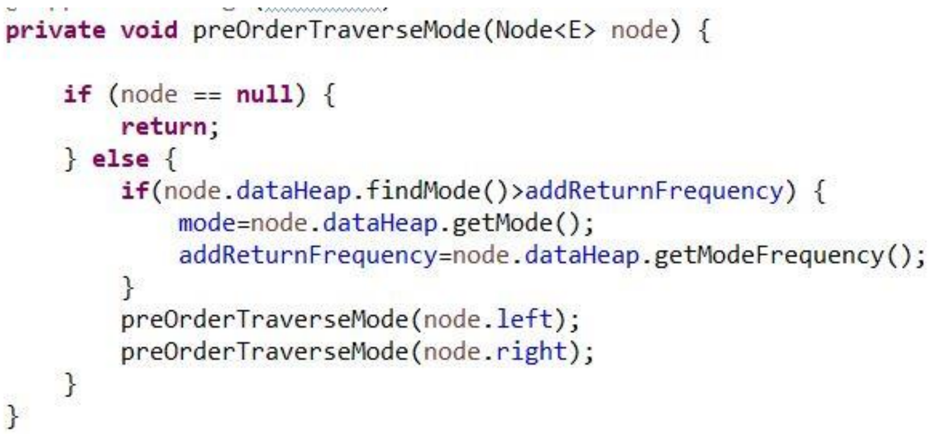
PreOrder = T(n)=O(logn)

Search() = Tb(n)= θ(1) , T_w =O(n)

Tb(n)= θ(1)

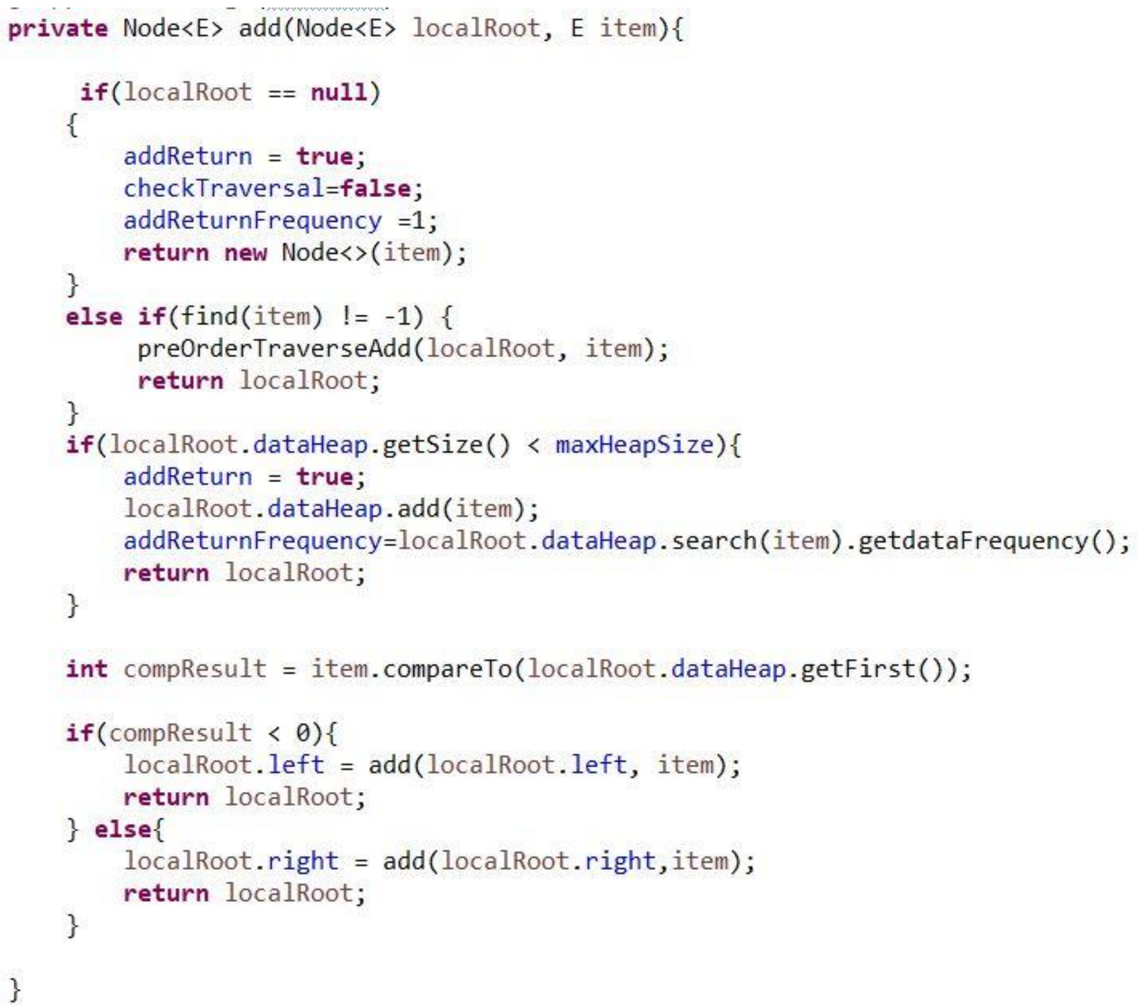
Tw(n) =O(n.logn)

T(n)=O(n.logn)



```
private void preOrderTraverseMode(Node<E> node) {

    if (node == null) {
        return;
    } else {
        if(node.dataHeap.findMode()>addReturnFrequency) {
            mode=node.dataHeap.getMode();
            addReturnFrequency=node.dataHeap.getModeFrequency();
        }
        preOrderTraverseMode(node.left);
        preOrderTraverseMode(node.right);
    }
}
```

Tw(n)=Tb(n) =O(n)



```
private Node<E> add(Node<E> localRoot, E item){

     if(localRoot == null)
    {
        addReturn = true;
        checkTraversal=false;
        addReturnFrequency =1;
        return new Node<>(item);
    }
    else if(find(item) != -1) {
         preOrderTraverseAdd(localRoot, item);
         return localRoot;
    }
    if(localRoot.dataHeap.getSize() < maxHeapSize){
        addReturn = true;
        localRoot.dataHeap.add(item);
        addReturnFrequency=localRoot.dataHeap.search(item).getdataFrequency();
        return localRoot;
    }

    int compResult = item.compareTo(localRoot.dataHeap.getFirst());

    if(compResult < 0){
        localRoot.left = add(localRoot.left, item);
        return localRoot;
    } else{
        localRoot.right = add(localRoot.right,item);
        return localRoot;
    }

}
```

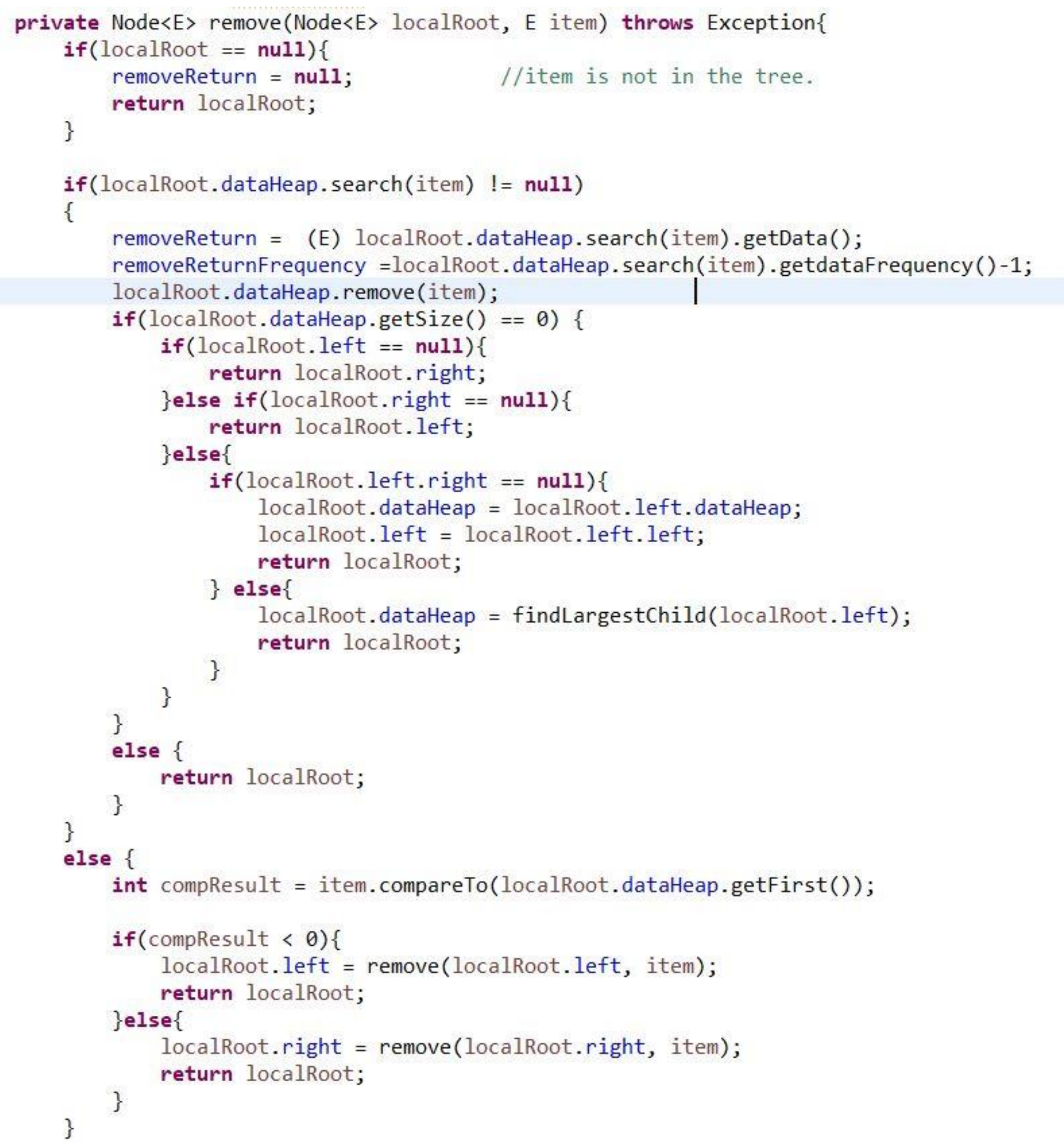
Find()=O(n)

Treversal = Tw(n) = O(n) ,   T(n)(amortized) =O(amortized)(logn)

Tb(n)= θ(1)

Tw(n) =O(n.n)

T(n)=O(n.logn)



```java
private Node<E> remove(Node<E> localRoot, E item) throws Exception{
    if(localRoot == null){
        removeReturn = null;            //item is not in the tree.
        return localRoot;
    }

    if(localRoot.dataHeap.search(item) != null)
    {
        removeReturn =  (E) localRoot.dataHeap.search(item).getData();
        removeReturnFrequency =localRoot.dataHeap.search(item).getdataFrequency()-1;
        localRoot.dataHeap.remove(item);
        if(localRoot.dataHeap.getSize() == 0) {
            if(localRoot.left == null){
                return localRoot.right;
            }else if(localRoot.right == null){
                return localRoot.left;
            }else{
                if(localRoot.left.right == null){
                    localRoot.dataHeap = localRoot.left.dataHeap;
                    localRoot.left = localRoot.left.left;
                    return localRoot;
                } else{
                    localRoot.dataHeap = findLargestChild(localRoot.left);
                    return localRoot;
                }
            }
        }
        else {
            return localRoot;
        }
    }
    else {
        int compResult = item.compareTo(localRoot.dataHeap.getFirst());

        if(compResult < 0){
            localRoot.left = remove(localRoot.left, item);
            return localRoot;
        }else{
            localRoot.right = remove(localRoot.right, item);
            return localRoot;
        }
    }
```

Treversal =O(amortized)(logn)

Tb(n)= θ(2) , because heapDepth=2

Tw(n) =O(n.2^n)   2^n for total search heap

T(n)=O(logn.2^logn)