PART1:

```java
public boolean contains(E item){
    for(E i : theData){
        if(i.compareTo(item) == 0){
            return true;
        }
    }
    return false;
}
```

O(n) O(n) O(1) O(1) O(1) O(n)

```java
public void add(E item){
    theData.add(item);
    int child = theData.size() - 1;
    int parent =  (child - 1)/2;
    E temp;

    while (theData.get(parent).compareTo(theData.get(child)) > 0 && parent >= 0){
        temp = theData.get(parent);
        theData.set(parent, theData.get(child));
        theData.set(child, temp);
        child = parent;
        parent = (child - 1)/2;
    }
}
```

θ(log n) θ(1) θ(1) θ(1) θ(1) θ(1) θ(1) θ(1) θ(1) θ(log n) θ(log n)

```java
public E find(E item){
    for(E element : theData){
        if(item.compareTo(element) == 0){
            return element;
        }
    }
    throw new NoSuchElementException();
}
```

O(n) O(n) O(1) O(1) O(1) O(n)

```java
private void heapify(ArrayList<E> arrayToHeapify, int size, int startIndex){
    int parent = startIndex;
    int leftChild = startIndex*2 + 1;
    int rightChild = startIndex*2 + 2;
    int check1;
    int check2;
    E temp;

    while (true){
        check1 = 0;
        check2 = 0;
        if(leftChild < size && arrayToHeapify.get(parent).compareTo(arrayToHeapify.get(leftChild)) > 0){
            check1++;
        }
        if(rightChild < size && arrayToHeapify.get(parent).compareTo(arrayToHeapify.get(rightChild)) > 0){
            check2++;
        }
        if(check1 == 0 && check2 == 0){
            break;
        }
        else{
            if(check1 == 1 && check2 == 1){
                if(arrayToHeapify.get(leftChild).compareTo(arrayToHeapify.get(rightChild)) < 0){
                    temp = arrayToHeapify.get(leftChild);
                    arrayToHeapify.set(leftChild, arrayToHeapify.get(parent));
                    arrayToHeapify.set(parent, temp);
                    parent = leftChild;
                }
                else{
                    temp = arrayToHeapify.get(rightChild);
                    arrayToHeapify.set(rightChild, arrayToHeapify.get(parent));
                    arrayToHeapify.set(parent, temp);
                    parent = rightChild;
                }
            }
            else if(check1 == 1){
                temp = arrayToHeapify.get(leftChild);
                arrayToHeapify.set(leftChild, arrayToHeapify.get(parent));
                arrayToHeapify.set(parent, temp);
                parent = leftChild;
            }
            else{
                temp = arrayToHeapify.get(rightChild);
                arrayToHeapify.set(rightChild, arrayToHeapify.get(parent));
                arrayToHeapify.set(parent, temp);
                parent = rightChild;
            }
            leftChild = parent*2 + 1;
            rightChild = parent*2 + 2;
        }
    }
}
```

```java
public E remove(){
    if(theData.size() == 0){
        return null;
    }
    E removedValue = theData.get(0);
    E lastValue = theData.remove( index: theData.size() - 1);
    if(!theData.isEmpty()) {
        theData.set(0, lastValue);
    }
    heapify(theData, theData.size(), startIndex: 0);
    return removedValue;
}
```

$\theta(1)$ $\theta(1)$ $\theta(1)$ $\theta(1)$ $\theta(1)$ $\theta(1)$ $O(\log n)$ $\theta(1)$

$O(\log n)$

```java
public void merge(Heap<E> other){
    while (!other.isEmpty()){
        add(other.remove());
    }
}
```

$\theta(1)$ $\theta(n)$ $\theta(\log n) + O(\log m)$ $O(\log m)$

$\Theta(n.\log n) + O(n.\log m)$

```java
private void sort(ArrayList<E> arrayToSort){
    int size = arrayToSort.size() - 1;
    E temp;
    while (size >= 0){
        temp = arrayToSort.get(0);
        arrayToSort.set(0, arrayToSort.get(size));
        arrayToSort.set(size, temp);
        heapify(arrayToSort, size, startIndex: 0);
        size--;
    }
}
```

$\theta(1)$ $O(n)$ $\theta(1)$ $\theta(1)$ $\theta(1)$ $O(\log n)$ $\theta(1)$

$O(n.\log n)$

$O(n.\log n)$

```java
public E removeSpecifiedLargestElement(int index){
    ArrayList<E> sortedArray = new ArrayList<>(theData);
    sort(sortedArray);
    E returnValue = sortedArray.remove( index: index - 1);
    theData.clear();
    while (!sortedArray.isEmpty()) {
        add(sortedArray.remove( index: sortedArray.size() - 1));
    }
    return returnValue;
}
```

```java
public boolean isEmpty(){
    return theData.size() == 0;
}
```

```java
public String toString() {
    ArrayList<E> sorted = new ArrayList<>(theData);
    sort(sorted);
    StringBuilder s = new StringBuilder();
    while (!sorted.isEmpty()){
        s.append(sorted.remove( index: sorted.size() - 1)).append(" ");
    }
    return s.toString();
}
```

```java
private StringBuilder treeStructure(int index, StringBuilder s, int level){
    for(int i = 1; i < level; i++){
        s.append(" ");
    }
    if(index >= theData.size()){
        return s.append("null\n");
    }
    s.append(theData.get(index)).append("\n");
    treeStructure( index: 2*index + 1, s,  level: level + 1);
    treeStructure( index: 2*index + 2, s,  level: level + 1);
    return s;
}
```

$O(n)$

```java
public String treeStructure(){
    return treeStructure( index: 0, new StringBuilder(),  level: 1).toString();
}
```

$O(n)$

```java
public HeapIter() {
    sortedArray = new ArrayList<>(theData);}Θ(n)
    sort(sortedArray);}O(n.log n)
    count = sortedArray.size() - 1;}Θ(1)
    lastItemReturned = null;}Θ(1)
}
```

$O(n.\log n)$

```java
public boolean hasNext() {
    return count >= 0;} Θ(1)
}
```

$\Theta(1)$

```java
public E next() {
    lastItemReturned =  sortedArray.get(count--);} Θ(1)
    return lastItemReturned;} Θ(1)
}
```

$\Theta(1)$

```java
private int findIndex(E item){
    int index;
    for(index = 0; index < theData.size() && theData.get(index).compareTo(item) != 0; index++);
    return index;
}
```

O(n)

```java
public void set(E e){
    if(lastItemReturned == null){
        throw new IllegalStateException();
    }
    int index = findIndex(lastItemReturned);
    theData.set(index, e);
    for(int i = theData.size()/2 - 1; i >= 0; i--){
        heapify(theData, theData.size(), i);
    }
}
```

O(n.log n)

PART2:

MaxHeap's time complexities of find, add, remove, getData, contains methods are constant since the size of MaxHeap is maximum 7 in BSTHeapTree.

```
private BinarySearchTree.Node<MaxHeap<HeapNode<E>>> add(BinarySearchTree.Node<MaxHeap<HeapNode<E>>> startNode, E item){
    if(startNode == null){
        size++;
        HeapNode<E> newHeapNode = new HeapNode<>(item);
        startNode = new BinarySearchTree.Node<>(new MaxHeap<>(newHeapNode));
        numOfOccurrences = newHeapNode.occurrence;
        return startNode;
    }
    else if(startNode.data.size() < MAX_HEAP_NODE){
        try {
            HeapNode<E> node = startNode.data.find(new HeapNode<>(item));
            node.occurrence++;
            numOfOccurrences = node.occurrence;
            size++;
        }
        catch (NoSuchElementException e){
            size++;
            HeapNode<E> newHeapNode = new HeapNode<>(item);
            startNode.data.add(newHeapNode);
            numOfOccurrences = newHeapNode.occurrence;
        }
        return startNode;
    }
    else if(startNode.data.contains(new HeapNode<>(item))){
        size++;
        HeapNode<E> node = startNode.data.find(new HeapNode<>(item));
        node.occurrence++;
        numOfOccurrences = node.occurrence;
    }
    else if(item.compareTo(startNode.data.getData().data) < 0){
        startNode.left = add(startNode.left, item);
    }
    else{
        startNode.right = add(startNode.right, item);
    }
    return startNode;
}
```

θ(1)

O(n)

```
public int add(E item){
    tree.setRoot(add(tree.getRoot(), item));
    return numOfOccurrences;
}
```

O(n)

O(n)

```java
private BinarySearchTree.Node<MaxHeap<HeapNode<E>>> remove(BinarySearchTree.Node<MaxHeap<HeapNode<E>>> startNode, E item){
    if(startNode == null){
        throw new NoSuchElementException();
    }
    else if(startNode.data.contains(new HeapNode<>(item))){
        size--;
        HeapNode<E> node = startNode.data.find(new HeapNode<>(item));
        node.occurrence--;
        numOfOccurrences = node.occurrence;
        if(startNode.data.size() == 1 && numOfOccurrences == 0){
            return tree.delete(startNode.data, startNode);
        }
        else if(numOfOccurrences == 0){
            startNode.data.remove(node);
            if(startNode.left != null){
                if(startNode.left.right == null){
                    if(startNode.left.data.size() == 1){
                        HeapNode<E> tempData = startNode.left.data.getData();
                        startNode.left = startNode.left.left;
                        startNode.data.add(tempData);
                        return startNode;
                    }
                    startNode.data.add(startNode.left.data.remove());
                }
                else{
                    startNode.data.add(biggestValue(startNode.left));
                }
            }
            else if(startNode.right != null){
                if(startNode.right.left == null){
                    if(startNode.right.data.size() == 1){
                        HeapNode<E> tempData = startNode.right.data.getData();
                        startNode.right = startNode.right.right;
                        startNode.data.add(tempData);
                        return startNode;
                    }
                    else {
                        Iterator<HeapNode<E>> iterator = startNode.right.data.iterator();
                        for(int i = 0; i < startNode.right.data.size() - 1; i++){
                            iterator.next();
                        }
                        HeapNode<E> returnValue = iterator.next();
                        startNode.right.data.remove(returnValue);
                        startNode.data.add(returnValue);
                    }
                }
                else{
                    for (HeapNode<E> heapNode : startNode.data) {
                        startNode.right.data.add(heapNode);
                    }
                    startNode = startNode.right;
                }
            }
        }
    }
    else if(item.compareTo(startNode.data.getData().data) < 0){
        startNode.left = remove(startNode.left, item);
    }
    else{
        startNode.right = remove(startNode.right, item);
    }
    return startNode;
}
```

```java
private HeapNode<E> biggestValue(BinarySearchTree.Node<MaxHeap<HeapNode<E>>> startNode){
    if(startNode.right.right == null){
        HeapNode<E> returnValue;
        if(startNode.right.data.size() == 1){
            returnValue = startNode.right.data.getData();
            startNode.right = startNode.right.left;
        }
        else{
            returnValue = startNode.right.data.remove();
        }
        return returnValue;
    }
    else {
        return biggestValue(startNode.right);
    }
}
```

θ(1) θ(1) θ(1) θ(1) θ(1) θ(1) θ(1) O(log n)

```java
private int find(BinarySearchTree.Node<MaxHeap<HeapNode<E>>> startNode, E item){
    if(startNode == null){
        throw new NoSuchElementException();
    }
    else if(startNode.data.contains(new HeapNode<>(item))){
        return startNode.data.find(new HeapNode<>(item)).occurrence;
    }
    else if(item.compareTo(startNode.data.getData().data) < 0){
        return find(startNode.left, item);
    }
    else{
        return find(startNode.right, item);
    }
}
```

θ(1) θ(1) θ(1) θ(1) θ(1) O(log n)

```java
public int find(E item){
    return find(tree.getRoot(), item);
}
```

O(log n)

```java
private HeapNode<E> mostOccurrence(MaxHeap<HeapNode<E>> heap){
    HeapNode<E> data = heap.getData();
    for(HeapNode<E> temp : heap){
        if(temp.occurrence > data.occurrence){
            data = temp;
        }
    }
    return data;
}
```

$\Theta(1)$

```java
private void find_mode(BinarySearchTree.Node<MaxHeap<HeapNode<E>>> startNode){
    if(startNode == null){
        return;
    }
    HeapNode<E> temp = mostOccurrence(startNode.data);
    if(temp.occurrence > mode.occurrence){
        mode = temp;
    }
    find_mode(startNode.left);
    find_mode(startNode.right);
}
```

$\Theta(1)$ $\Theta(n)$

```java
public E find_mode(){
    find_mode(tree.getRoot());
    return mode.data;
}
```

$\Theta(n)$

```java
public String toString() {
    return tree.toString();
}
```

$O(\tilde{n})$

```java
public E getData(){
    return tree.getData().getData().data;
}
```

$\Theta(1)$