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

**Burak Yıldırım**
**1901042609**

1. SYSTEM REQUIREMENTS

Non-functional Requirements:

User must have Java Runtime Environment to run this project.

Functional Requirements:

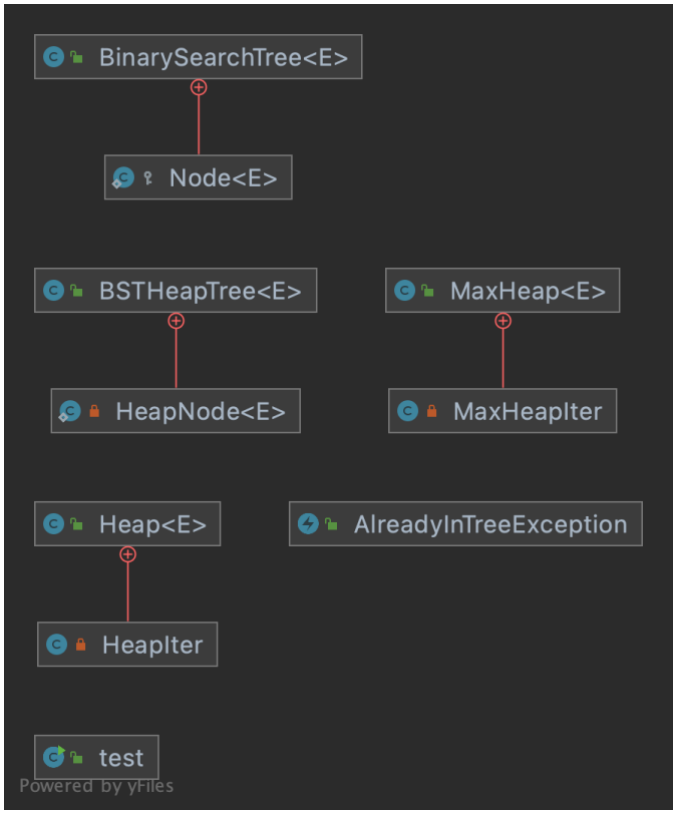
Part1:

- User shall be able to add new elements to the heap.
- User shall be able to remove $i^{th}$ largest element from the heap.
- User shall be able to merge the current heap with another heap.
- User shall be able to set the value of current iteration using iterator.
- User shall be able to check if the specified element is in the heap.
- User shall be able to check if the heap is empty.
- User shall be able to get the tree representation of the heap.
- User shall be able to get an iterator over the elements in the heap.

Part2:

- User shall be able to add new elements to the tree.
- User shall be able to remove the specified element from the tree.
- User shall be able to find the specified element in the tree.
- User shall be able to find the mode of the elements in the tree.
- User shall be able to get the number of elements in the tree.
- User shall be able to get the element at the top of the tree.

2. DIAGRAMS




## 3. PROBLEM SOLUTION APPROACH

### Part1:

I implemented Heap class as a min heap and HeapIter class as its iterator.

### Part2:

I implemented BinarySearchTree class and used it as a data field in BSTHeapTree class.I also implemented MaxHeap class and used it as nodes in BinarySearchTree.

4. TEST CASES

**Part1:**

Initialization

```
Heap<Integer> heap1 = new Heap<>();
heap1.add(100);
heap1.add(40);
heap1.add(23);
heap1.add(78);
heap1.add(12);
heap1.add(1);
heap1.add(59);
heap1.add(150);
Heap<Integer> heap2 = new Heap<>();
heap2.add(5);
heap2.add(9);
heap2.add(37);
heap2.add(94);
heap2.add(72);
System.out.println("PART1\n\nadded numbers: 100, 40, 23, 78, 12, 1, 59, 150");
System.out.println("\nprinting heap\n" + heap1);
System.out.println("\nprinting heap as tree structure\n" + heap1.treeStructure());
```

Search

```
System.out.println("\nsearching an existing element(100)");
System.out.println("result: " + heap1.find( item: 100));
System.out.println("\nsearching a non-existing element(3)");
System.out.printf("result: ");
try {
    heap1.find( item: 3);
}
catch (NoSuchElementException e){
    System.out.println(e);
}
```

Merge

```
System.out.println("\nheap2: " + heap2);
System.out.println("heap2 as tree structure\n" + heap2.treeStructure());
System.out.println("merging heap with heap2");
System.out.println("heap before merge: " + heap1 + "\nas tree structure\n" + heap1.treeStructure());
heap1.merge(heap2);
System.out.println("heap after merge: " + heap1 + "\nas tree structure\n" + heap1.treeStructure());
```

Remove $i^{th}$ largest element

```
System.out.println("\nremoving 2nd largest element");
System.out.println("before removal: " + heap1 + "\nas tree structure\n" + heap1.treeStructure());
System.out.println("removed element: " + heap1.removeSpecifiedLargestElement( index: 2));
System.out.println("after removal: " + heap1 + "\nas tree structure\n" + heap1.treeStructure());
```

Set

```
System.out.println("\nsetting 3rd element to 200");
System.out.println("before setting: " + heap1 + "\nas tree structure\n" + heap1.treeStructure());
ListIterator<Integer> iterator = heap1.heapIterator();
for(int i = 0; i < 3; i++){
    iterator.next();
}
iterator.set(200);
System.out.println("after setting: " + heap1 + "\nas tree structure\n" + heap1.treeStructure());
```

**Part2:**

Initialization

```
ArrayList<Integer> arrayList = new ArrayList<>();
BSTHeapTree<Integer> bstHeapTree = new BSTHeapTree<>();
Random random = new Random();
int temp;
for(int i = 0; i < 3000; i++){
    temp = random.nextInt( bound: 5000);
    arrayList.add(temp);
    bstHeapTree.add(temp);
}
Collections.sort(arrayList);
```

Find

```
System.out.println("\nPART2\nfinding occurrences of existing elements\n");
int count = 1;
for(int i = 0, j = 0; i < 100; i++, j++){
    System.out.println("element to find occurrence: " + arrayList.get(j));
    System.out.println("tree result: " + bstHeapTree.find(arrayList.get(j)));
    while (arrayList.get(j).equals(arrayList.get(j + 1))){
        count++;
        j++;
    }
    System.out.println("arraylist result: " + count + "\n");
    count = 1;
}

System.out.println("\nfinding occurrences of non-existing elements\n");
for(int i = 0; i < 10; i++){
    System.out.println("element to find occurrence: " + (i + 5001));
    try {
        bstHeapTree.find( item: i + 5001);
    }
    catch (NoSuchElementException e){
        System.out.println("tree result: " + e);
    }
    System.out.println("arraylist result: " + arrayList.indexOf(i + 5001) + "\n");
}
```

Find Mode

```
System.out.println("\nfinding mode\n");
System.out.println("tree result: " + bstHeapTree.find_mode());
int mode = arrayList.get(0);
int index = 0;
while (index < arrayList.size()){
    temp = occurrences(arrayList, arrayList.get(index));
    if(temp > count){
        count = temp;
        mode = arrayList.get(index);
    }
    index += temp;
}
System.out.println("arraylist result: " + mode);
```

Remove

```
System.out.println("\nremoving existing elements\n");
for(int i = 0; i < 100; i++){
    temp = arrayList.get(0);
    System.out.println("element to remove: " + temp);
    System.out.println("occurrence before removal");
    System.out.println("tree result: " + bstHeapTree.find(temp));
    System.out.println("arraylist result: " + occurrences(arrayList, temp));
    System.out.println("occurrence after removal");
    System.out.println("tree result: " + bstHeapTree.remove(arrayList.remove( index: 0)));
    System.out.println("arraylist result: " + occurrences(arrayList, temp) + "\n");
}

System.out.println("\nremoving non-existing elements");
for(int i = 0; i < 10; i++){
    System.out.println("element to remove: " + (i + 5001));
    try {
        bstHeapTree.remove((i + 5001));
    } catch (NoSuchElementException e) {
        System.out.println("tree result: " + e);
    }
    System.out.println("arraylist result: " + arrayList.remove((Object) (i + 5001)) + "\n");
}
```

## 5. RUNNING AND RESULTS

Part1:

```
PART1

added numbers: 100, 40, 23, 78, 12, 1, 59, 150

printing heap
1 12 23 40 59 78 100 150

printing heap as tree structure
1
 23
  100
   150
    null
    null
   null
  78
   null
   null
 12
  40
   null
   null
  59
   null
   null
```

```
searching an existing element(100)
result: 100

searching a non-existing element(3)
result: java.util.NoSuchElementException
```

```
heap2: 5 9 37 72 94
heap2 as tree structure
5
 9
  94
   null
   null
  72
   null
   null
 37
  null
  null

merging heap with heap2
heap before merge: 1 12 23 40 59 78 100 150
as tree structure
1
 23
  100
   150
    null
    null
   null
  78
   null
   null
 12
  40
   null
   null
  59
   null
   null
```

```
heap after merge: 1 5 9 12 23 37 40 59 72 78 94 100 150
as tree structure
1
 5
  23
   150
    null
    null
   100
    null
    null
  9
   78
    null
    null
   37
    null
    null
 12
  40
   72
    null
    null
   94
    null
    null
  59
   null
   null
```

```
removing 2nd largest element
before removal: 1 5 9 12 23 37 40 59 72 78 94 100 150
as tree structure
1
 5
  23
   150
    null
    null
   100
    null
    null
  9
   78
    null
    null
   37
    null
    null
 12
  40
   72
    null
    null
   94
    null
    null
  59
   null
   null
```

```
removed element: 100
after removal: 1 5 9 12 23 37 40 59 72 78 94 150
as tree structure
1
 5
  12
   59
    null
    null
   72
    null
    null
  23
   78
    null
    null
   94
    null
    null
 9
  37
   150
    null
    null
   null
  40
   null
   null
```

```
setting 3rd element to 200
before setting: 1 5 9 12 23 37 40 59 72 78 94 150
as tree structure
1
 5
  12
   59
    null
    null
   72
    null
    null
  23
   78
    null
    null
   94
    null
    null
 9
  37
   150
    null
    null
   null
  40
   null
   null
```

```
after setting: 1 5 12 23 37 40 59 72 78 94 150 200
as tree structure
1
 5
  12
   59
    null
    null
   72
    null
    null
  23
   78
    null
    null
   94
    null
    null
 37
  150
   200
    null
    null
   null
  40
   null
   null
```

## Part2:

```
PART2
finding occurrences of existing elements

element to find occurrence: 4
tree result: 1
arraylist result: 1

element to find occurrence: 5
tree result: 1
arraylist result: 1

element to find occurrence: 6
tree result: 2
arraylist result: 2

element to find occurrence: 7
tree result: 1
arraylist result: 1

element to find occurrence: 9
tree result: 1
arraylist result: 1

element to find occurrence: 15
tree result: 2
arraylist result: 2

element to find occurrence: 17
tree result: 1
arraylist result: 1

element to find occurrence: 21
tree result: 2
arraylist result: 2

element to find occurrence: 22
tree result: 1
arraylist result: 1
```

```
finding occurrences of non-existing elements

element to find occurrence: 5001
tree result: java.util.NoSuchElementException
arraylist result: -1

element to find occurrence: 5002
tree result: java.util.NoSuchElementException
arraylist result: -1

element to find occurrence: 5003
tree result: java.util.NoSuchElementException
arraylist result: -1

element to find occurrence: 5004
tree result: java.util.NoSuchElementException
arraylist result: -1

element to find occurrence: 5005
tree result: java.util.NoSuchElementException
arraylist result: -1

element to find occurrence: 5006
tree result: java.util.NoSuchElementException
arraylist result: -1

element to find occurrence: 5007
tree result: java.util.NoSuchElementException
arraylist result: -1

element to find occurrence: 5008
tree result: java.util.NoSuchElementException
arraylist result: -1

element to find occurrence: 5009
tree result: java.util.NoSuchElementException
arraylist result: -1

element to find occurrence: 5010
tree result: java.util.NoSuchElementException
arraylist result: -1
```

```
finding mode

tree result: 1876
arraylist result: 1876
```

```
removing existing elements

element to remove: 4
occurrence before removal
tree result: 1
arraylist result: 1
occurrence after removal
tree result: 0
arraylist result: 0

element to remove: 5
occurrence before removal
tree result: 1
arraylist result: 1
occurrence after removal
tree result: 0
arraylist result: 0

element to remove: 6
occurrence before removal
tree result: 2
arraylist result: 2
occurrence after removal
tree result: 1
arraylist result: 1

element to remove: 6
occurrence before removal
tree result: 1
arraylist result: 1
occurrence after removal
tree result: 0
arraylist result: 0

element to remove: 7
occurrence before removal
tree result: 1
arraylist result: 1
occurrence after removal
tree result: 0
arraylist result: 0
```

```
removing non-existing elements
element to remove: 5001
tree result: java.util.NoSuchElementException
arraylist result: false

element to remove: 5002
tree result: java.util.NoSuchElementException
arraylist result: false

element to remove: 5003
tree result: java.util.NoSuchElementException
arraylist result: false

element to remove: 5004
tree result: java.util.NoSuchElementException
arraylist result: false

element to remove: 5005
tree result: java.util.NoSuchElementException
arraylist result: false

element to remove: 5006
tree result: java.util.NoSuchElementException
arraylist result: false

element to remove: 5007
tree result: java.util.NoSuchElementException
arraylist result: false

element to remove: 5008
tree result: java.util.NoSuchElementException
arraylist result: false

element to remove: 5009
tree result: java.util.NoSuchElementException
arraylist result: false

element to remove: 5010
tree result: java.util.NoSuchElementException
arraylist result: false
```