GTU Department of Computer Engineering
CSE 222/505 - Spring 2022
Homework #05 Report

Ali Kaya

1901042618

a.kaya2019@gtu.edu.tr

1.Detailed System Requirements

There is not so much system requirements except generics type user must give it to the program.

```
BinaryHeap<Character> myHeap = new BinaryHeap<>();
```

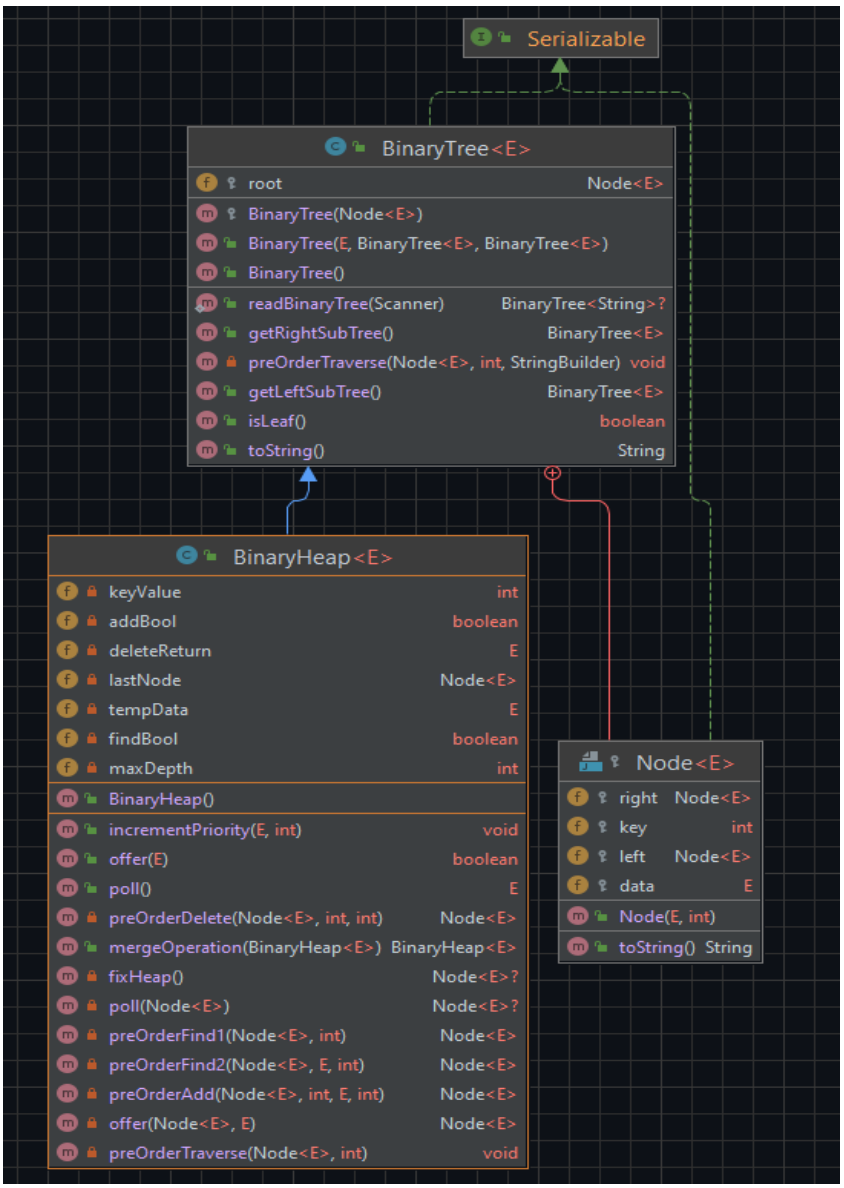
```
BinaryHeap<Character> myHeap2 = new BinaryHeap<>();
```

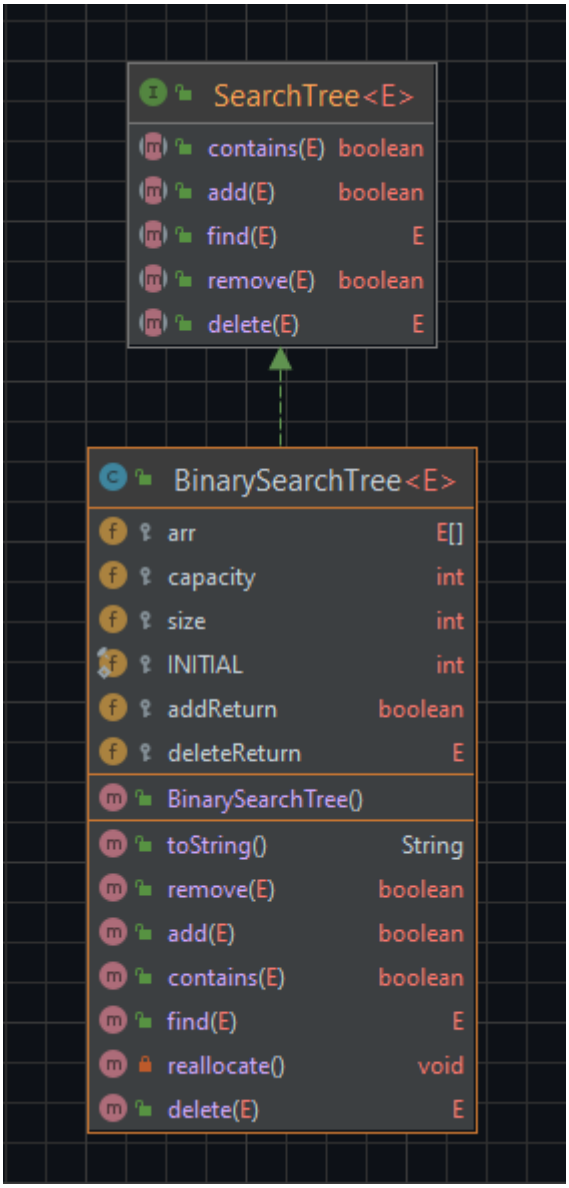
Here, Character is our type..

```
BinarySearchTree<Integer> myBinary = new BinarySearchTree<>();
```

Here, Integer is our type..

2. Class Diagrams

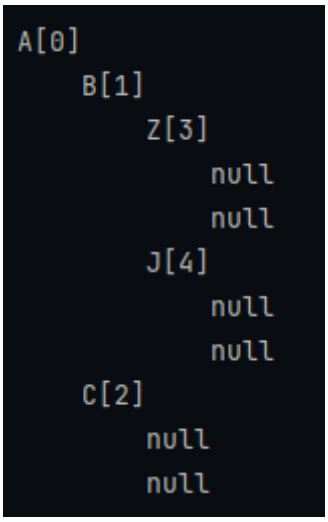




3. Problem solutions approach

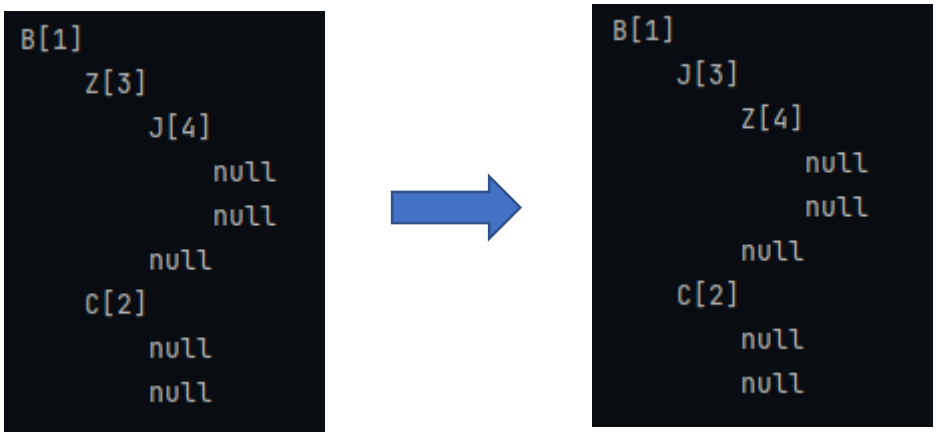
#For BinaryHeap<E>

For simple heap, sorting is done according to data, but the heap I wrote is sorting according to key .



Here [0] , [1] , [2]… values represents key values of nodes..

When user want to increment the priority value of an element user need to use incrementPriority() method.



Here we wrote this code = >

```
myHeap.incrementPriority( data: 'J', keyValue: 4);
```

To mention to mergeOperation() method ;

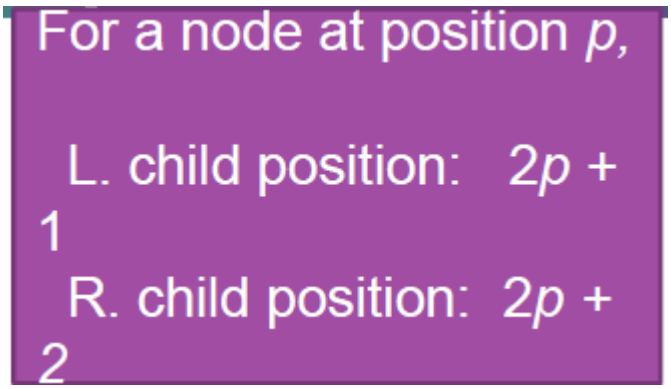
This method simply transfers the second heap elements to the first heap. If the user wants to change the priority between elements in merged heap, She/He can use the incrementPriority() method again.

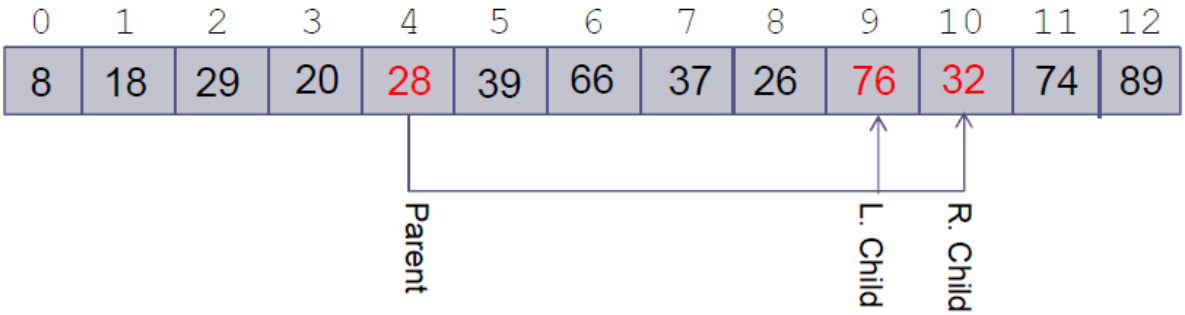
Analyzing methods => generally methods' time complexities are O(N)

Because we use linked data structure instead of using array => methods need to use preOrderTraverse (O(N))

#For BinarySearchTree<E>

I used the formulas below in order to be able to navigate in the array according to the parent child relationship.





Time complexity in add(item) , contains(target) and find(target) methods are O(logN) even though we use array structure

We used while loops, but parent value increases each time as 2x +1 or 2x +2

```
leftChild = 2*parent + 1;
rightChild = 2*parent + 2;
```

4)Test Cases

```
myHeap.offer( item: 'A');
myHeap.offer( item: 'B');
myHeap.offer( item: 'C');
myHeap.offer( item: 'Z');
myHeap.offer( item: 'J');
```

```
myHeap2.offer( item: 'W');
myHeap2.offer( item: 'X');
myHeap2.offer( item: 'Y');
myHeap2.offer( item: 'O');
```

```
myBinary.add(9);
myBinary.add(12);
myBinary.add(5);
myBinary.add(14);
myBinary.add(11);
myBinary.add(3);
myBinary.add(6);
```

5) Running command and results

```
A[0]
    B[1]
        Z[3]
            null
            null
        J[4]
            null
            null
    C[2]
        null
        null
```

```
myHeap.poll();
```

```
B[1]
    Z[3]
        J[4]
            null
            null
        null
    C[2]
        null
        null
```

```
myHeap.incrementPriority( data: 'J', keyValue: 4);
```

```
B[1]
    J[3]
        Z[4]
            null
            null
        null
    C[2]
        null
        null
```

```
W[0]
    X[1]
        O[3]
            null
            null
        null
    Y[2]
        null
        null
```

```
myHeap2.incrementPriority( data: 'Y',  keyValue: 2);
```

```
W[0]
    Y[1]
        O[3]
            null
            null
        null
    X[2]
        null
        null
```

```
myHeap = myHeap.mergeOperation(myHeap2);
System.out.println(myHeap.toString());
```

```
B[1]
    J[3]
        Z[4]
            O[8]
                null
                null
            null
        W[5]
            null
            null
    C[2]
        Y[6]
            null
            null
        X[7]
            null
            null
```

```
{9} {5} {12} {3} {6} {11} {14} {null} {null} {null} {null} {null} {nu
```

```
myBinary.remove( target: 6);
```

```
{9} {5} {12} {3} {null} {11} {14} {null} {null} {null} {null} {nul
```

```
System.out.println(myBinary.delete( target: 12));
```

```
{9} {5} {11} {3} {null} {null} {14} {null} {null} {null} {null} {nu
```