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

Emre SEZER
1901042640

SYSTEM REQUIREMENTS:

I created this homework on Windows 10 using terminal (Java Development Kit).My java version is 11.0.8 .You need to compile “driver.java” in order to test my homework.There are java files, a folder named “Javadoc” that includes javadoc files and report file in my homework.

PROBLEM SOLUTION APPROACH:

I wrote a driver class for testing. For testing i wrote some supoprt methods to make testing more understandable.

PART 1:

I implemented a MaxHeap using ArrayList. I implemented add and remove methods and an inner iterator class named HeapIterator. Then, I added requested methods. Whatever operation is done on Heap, heap structure will be preserved. For testing Part 1, I wrote a printHeap() method.That method prints the elements in the Heap.

PART 2:

I implemented a MaxHeap using ArrayList that keeps HeapNodes.

HeapNode is inner class keeps items and their number of occurrences. I implemented add and remove methods for MaxHeap class. I also added support methods for implementing BSTHeapTree. For testing Part 2 ,I wrote a printHeap() method.That method prints the elements in the MaxHeap.

I implemented BSTHeapTree Binary Search Tree class that keeps TreeNodes as its nodes. TreeNode keeps MaxHeap as its data , left and right TreeNodes as its children.Then, I added requested methods.

While testing Part 2, I followed the instructions written on the pdf file .

Test Cases :

PART 1:

1) Search on an empty Heap : Throws exception
2) Merge 2 empty Heaps : Merged Heap will be empty.Since, before merge operation both Heaps have 0 elements.
3) Merge 2 not empty Heaps: Succesfully merges 2 Heaps.
4) Remove i'th largest element from the heap (i >= size()): Since, i is larger than size of the Heap ,throws exception.
5) Remove i'th largest element from the heap (i <= 0): throws exception.
6) Remove i'th largest element from the heap (i < size() && i > 0):

7)Succesfully removes i'th largest element from the Heap.Set an element to entry : Succesfully sets that element to entry and prevents the Heap structure.

PART 2:

1) Add entry to the BSTHeapTree : Succesfully adds entry to the BSTHeapTree
2) Remove element that doesn't exist in the BSTHeapTree: Returns 0
3) Remove element exists in the BSTHeapTree: Returns nunmber of occurences after remove operation
4) Remove element from empty BSTHeapTree: Throws exception.
5) Find element that doesn't exist in the BSTHeapTree : Returns 0
6) Find element that exists in the BSTHeapTree : Returns number of occurences of the element.
7) Find element in an empty BSTHeapTree : Returns 0.
8) Find mode of the BSTHeapTree : Returns the element with most number of occurences if exists.If there are 2 or more elements with most number of occurences, Prints all of them.

SCREENSHOTS:

~/java/cse222/hw4

PART 1:

```
emr3s@DESKTOP-POGAK7B ~/java/cse222/hw4
$ javac driver.java

emr3s@DESKTOP-POGAK7B ~/java/cse222/hw4
$ java driver
PART 1 TESTS:
TRYING TO SEARCH ON EMPTY HEAP:
HEAP IS EMPTY
TRYING TO MERGE 2 EMPTY HEAPS.AFTER THAT OPERATION MERGED HEAP WILL STILL BE EMPTY
HEAP IS EMPTY

HEAP1:
55 34 43 13 23 14 1
HEAP1 SEARCH TESTS:
13 IS FOUND ON HEAP
32 IS NOT FOUND ON HEAP

HEAP1 MERGE TESTS:

HEAP1 BEFORE MERGE:
55 43 34 23 14 13 1

HEAP2 BEFORE MERGE:
99 87 55 66 72 6 7 8 12 34

HEAP1 AFTER MERGE:(AS YOU CAN SEE HEAP2 IS SUCCESFULLY ADDED TO HEAP1 AND HEAP STRUCTURE IS PRESERVED)
99 87 55 55 72 34 12 23 43 14 66 13 34 1 8 7 6

HEAP2 BEFORE REMOVAL:
99 87 72 66 55 34 12 8 7 6

Removing 3'rd Largest Element From The heap2 :
Removed Element : 72

HEAP2 AFTER REMOVAL:(AS YOU CAN SEE 72 IS REMOVED FROM THE HEAP2)
99 87 34 66 6 7 12 8 55

Removing 10'th Largest Element From The heap2 (Since size of the heap2 is less than 10 exception will be thrown):
INDEX OUT OF BOUNDS


HEAP2 BEFORE SET OPERATION:
99 87 34 66 6 7 12 8 55
HEAP2 AFTER SET OPERATION:((AS YOU CAN SEE 6 IS SET TO 101)
101 99 34 66 87 7 12 8 55

PART 2 TESTS:
TYPE ANYTHING TO CONTINUE TESTING:
```

Type here to search

ENG TRQ 16:01 03/05/2021

~/java/cse222/hayat

```
3742 : (In Array : 2 In Tree : 2)
3819 : (In Array : 1 In Tree : 1)
390 : (In Array : 2 In Tree : 2)
3638 : (In Array : 1 In Tree : 1)
4166 : (In Array : 3 In Tree : 3)
2503 : (In Array : 1 In Tree : 1)
3614 : (In Array : 2 In Tree : 2)
1050 : (In Array : 1 In Tree : 1)
4030 : (In Array : 2 In Tree : 2)
116 : (In Array : 1 In Tree : 1)
1132 : (In Array : 1 In Tree : 1)
2802 : (In Array : 3 In Tree : 3)
1687 : (In Array : 2 In Tree : 2)
3286 : (In Array : 2 In Tree : 2)
2250 : (In Array : 1 In Tree : 1)
3062 : (In Array : 1 In Tree : 1)
3417 : (In Array : 2 In Tree : 2)
4155 : (In Array : 2 In Tree : 2)
2684 : (In Array : 1 In Tree : 1)
3064 : (In Array : 2 In Tree : 2)
2358 : (In Array : 3 In Tree : 3)
679 : (In Array : 1 In Tree : 1)
1858 : (In Array : 1 In Tree : 1)
705 : (In Array : 2 In Tree : 2)
3683 : (In Array : 1 In Tree : 1)
1304 : (In Array : 1 In Tree : 1)
3889 : (In Array : 1 In Tree : 1)
1676 : (In Array : 3 In Tree : 3)
2241 : (In Array : 2 In Tree : 2)
262 : (In Array : 2 In Tree : 2)
1191 : (In Array : 2 In Tree : 2)
2081 : (In Array : 1 In Tree : 1)
572 : (In Array : 1 In Tree : 1)
64 : (In Array : 1 In Tree : 1)
3081 : (In Array : 1 In Tree : 1)
762 : (In Array : 2 In Tree : 2)
1203 : (In Array : 3 In Tree : 3)
4441 : (In Array : 1 In Tree : 1)
4866 : (In Array : 1 In Tree : 1)
1958 : (In Array : 1 In Tree : 1)
2261 : (In Array : 3 In Tree : 3)
3669 : (In Array : 2 In Tree : 2)
2829 : (In Array : 1 In Tree : 1)
104 : (In Array : 1 In Tree : 1)
2376 : (In Array : 1 In Tree : 1)
4946 : (In Array : 1 In Tree : 1)
4905 : (In Array : 2 In Tree : 2)
1902 : (In Array : 1 In Tree : 1)
1245 : (In Array : 1 In Tree : 1)
1113 : (In Array : 2 In Tree : 2)
4482 : (In Array : 1 In Tree : 1)
1285 : (In Array : 1 In Tree : 1)
4563 : (In Array : 1 In Tree : 1)
2274 : (In Array : 3 In Tree : 3)
1 : (In Array : 1 In Tree : 1)
4655 : (In Array : 1 In Tree : 1)
3315 : (In Array : 2 In Tree : 2)
4422 : (In Array : 1 In Tree : 1)
547 : (In Array : 1 In Tree : 1)
4933 : (In Array : 1 In Tree : 1)
3436 : (In Array : 1 In Tree : 1)
754 : (In Array : 4 In Tree : 4)
3734 : (In Array : 2 In Tree : 2)
4617 : (In Array : 1 In Tree : 1)
3926 : (In Array : 1 In Tree : 1)
2788 : (In Array : 2 In Tree : 2)
801 : (In Array : 1 In Tree : 1)
519 : (In Array : 2 In Tree : 2)
1137 : (In Array : 1 In Tree : 1)
707 : (In Array : 1 In Tree : 1)
1863 : (In Array : 3 In Tree : 3)
```

PART 2:
(TESTING ADD AND FIND METHODS)

Type here to search

ENG TRQ 16:19 01/05/2021

~/java/cse222/hayat

```
4655 : (In Array : 1 In Tree : 1)
3315 : (In Array : 2 In Tree : 2)
4422 : (In Array : 1 In Tree : 1)
547 : (In Array : 1 In Tree : 1)
4933 : (In Array : 1 In Tree : 1)
3436 : (In Array : 1 In Tree : 1)
754 : (In Array : 4 In Tree : 4)
3734 : (In Array : 2 In Tree : 2)
4617 : (In Array : 1 In Tree : 1)
3926 : (In Array : 1 In Tree : 1)
2788 : (In Array : 2 In Tree : 2)
801 : (In Array : 1 In Tree : 1)
519 : (In Array : 2 In Tree : 2)
1137 : (In Array : 1 In Tree : 1)
707 : (In Array : 1 In Tree : 1)
1863 : (In Array : 3 In Tree : 3)
3772 : (In Array : 1 In Tree : 1)
2559 : (In Array : 1 In Tree : 1)
4313 : (In Array : 1 In Tree : 1)
3451 : (In Array : 1 In Tree : 1)
2371 : (In Array : 1 In Tree : 1)
1645 : (In Array : 3 In Tree : 3)
248 : (In Array : 1 In Tree : 1)
91 : (In Array : 1 In Tree : 1)
3613 : (In Array : 2 In Tree : 2)
2533 : (In Array : 1 In Tree : 1)
3983 : (In Array : 1 In Tree : 1)
4193 : (In Array : 1 In Tree : 1)
3939 : (In Array : 1 In Tree : 1)
4776 : (In Array : 2 In Tree : 2)
798 : (In Array : 3 In Tree : 3)
2169 : (In Array : 1 In Tree : 1)
4143 : (In Array : 2 In Tree : 2)
951 : (In Array : 1 In Tree : 1)
3378 : (In Array : 2 In Tree : 2)
908 : (In Array : 3 In Tree : 3)
1392 : (In Array : 2 In Tree : 2)
2722 : (In Array : 2 In Tree : 2)
152 : (In Array : 1 In Tree : 1)
1549 : (In Array : 1 In Tree : 1)
3176 : (In Array : 2 In Tree : 2)
1060 : (In Array : 1 In Tree : 1)
564 : (In Array : 1 In Tree : 1)
907 : (In Array : 1 In Tree : 1)
520 : (In Array : 2 In Tree : 2)

TYPE ANYTHING TO CONTINUE TESTING:
FINDING ELEMENTS THERE ARE NOT ON TREE:

5092 : (In Array : 0 In Tree : 0)
5033 : (In Array : 0 In Tree : 0)
5052 : (In Array : 0 In Tree : 0)
5004 : (In Array : 0 In Tree : 0)
5037 : (In Array : 0 In Tree : 0)
5082 : (In Array : 0 In Tree : 0)
5027 : (In Array : 0 In Tree : 0)
5048 : (In Array : 0 In Tree : 0)
5031 : (In Array : 0 In Tree : 0)
5081 : (In Array : 0 In Tree : 0)

TYPE ANYTHING TO CONTINUE TESTING:

MODE OF THE TREE IS:
431 with 4 occurences
754 with 4 occurences
816 with 4 occurences
1012 with 4 occurences
1386 with 4 occurences
2521 with 4 occurences
3640 with 4 occurences
4792 with 4 occurences
```

TESTING FIND METHODS ON NON-EXISTING NUMBERS

TESTING MODE METHOD

Type here to search

ENG TRQ 16:19 01/05/2021

~/java/cse222/hayat

TESTING REMOVE AND FIND METHODS

```
After remove 1 4370 left
4370 (In Array : 2 In Tree : 1)
After remove 0 4317 left
4317 (In Array : 1 In Tree : 0)
After remove 2 3386 left
3386 (In Array : 3 In Tree : 2)
After remove 1 3331 left
3331 (In Array : 2 In Tree : 1)
After remove 0 317 left
317 (In Array : 1 In Tree : 0)
After remove 0 2395 left
2395 (In Array : 1 In Tree : 0)
After remove 1 4030 left
4030 (In Array : 2 In Tree : 1)
After remove 0 3772 left
3772 (In Array : 1 In Tree : 0)
After remove 3 3640 left
3640 (In Array : 4 In Tree : 3)
After remove 0 3798 left
3798 (In Array : 1 In Tree : 0)
After remove 0 4706 left
4706 (In Array : 1 In Tree : 0)
After remove 0 4781 left
4781 (In Array : 1 In Tree : 0)
After remove 1 2085 left
2085 (In Array : 2 In Tree : 1)
After remove 0 385 left
385 (In Array : 1 In Tree : 0)
After remove 1 2268 left
2268 (In Array : 2 In Tree : 1)
After remove 0 1998 left
1998 (In Array : 1 In Tree : 0)
After remove 0 3368 left
3368 (In Array : 1 In Tree : 0)
After remove 2 4648 left
4648 (In Array : 3 In Tree : 2)
After remove 0 1201 left
1201 (In Array : 1 In Tree : 0)
After remove 0 3722 left
3722 (In Array : 1 In Tree : 0)
After remove 1 3386 left
3386 (In Array : 3 In Tree : 1)
After remove 1 779 left
779 (In Array : 2 In Tree : 1)
After remove 0 3103 left
3103 (In Array : 1 In Tree : 0)
After remove 1 4820 left
4820 (In Array : 2 In Tree : 1)
After remove 0 1310 left
1310 (In Array : 1 In Tree : 0)
After remove 1 1881 left
1881 (In Array : 2 In Tree : 1)
After remove 2 998 left
998 (In Array : 3 In Tree : 2)
After remove 1 1635 left
1635 (In Array : 2 In Tree : 1)
After remove 0 613 left
613 (In Array : 1 In Tree : 0)
After remove 0 2179 left
2179 (In Array : 1 In Tree : 0)
After remove 2 3001 left
3001 (In Array : 3 In Tree : 2)
After remove 2 2824 left
2824 (In Array : 3 In Tree : 2)
After remove 1 997 left
997 (In Array : 2 In Tree : 1)
After remove 0 4748 left
4748 (In Array : 1 In Tree : 0)
After remove 0 4203 left
4203 (In Array : 1 In Tree : 0)
After remove 0 2130 left
2130 (In Array : 1 In Tree : 0)
```

Type here to search

ENG TRQ 16:20 01/05/2021

~/java/cse222/hayat

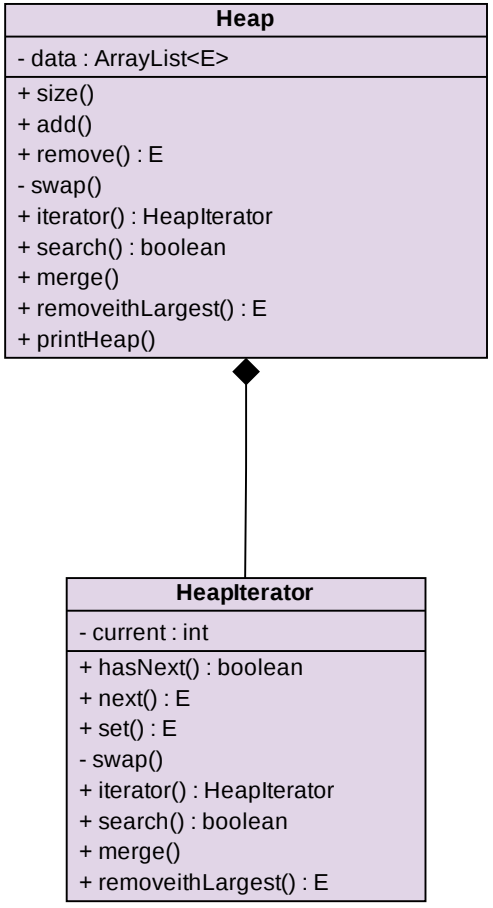
```
After remove 2 223 left
223 (In Array : 3 In Tree : 2)
After remove 0 2037 left
2037 (In Array : 1 In Tree : 0)
After remove 3 1386 left
1386 (In Array : 4 In Tree : 3)
After remove 0 4809 left
4809 (In Array : 1 In Tree : 0)
After remove 1 4115 left
4115 (In Array : 2 In Tree : 1)
After remove 0 1695 left
1695 (In Array : 1 In Tree : 0)
After remove 0 1826 left
1826 (In Array : 1 In Tree : 0)
After remove 0 801 left
801 (In Array : 1 In Tree : 0)
After remove 0 3659 left
3659 (In Array : 1 In Tree : 0)
After remove 0 3683 left
3683 (In Array : 1 In Tree : 0)
After remove 0 2402 left
2402 (In Array : 1 In Tree : 0)
After remove 2 3700 left
3700 (In Array : 3 In Tree : 2)
After remove 0 3130 left
3130 (In Array : 1 In Tree : 0)
After remove 2 3173 left
3173 (In Array : 3 In Tree : 2)
After remove 0 655 left
655 (In Array : 1 In Tree : 0)
After remove 0 3952 left
3952 (In Array : 1 In Tree : 0)
After remove 0 4488 left
4488 (In Array : 1 In Tree : 0)
After remove 2 70 left
70 (In Array : 3 In Tree : 2)
After remove 0 2950 left
2950 (In Array : 1 In Tree : 0)
After remove 0 4535 left
4535 (In Array : 1 In Tree : 0)
After remove 0 4468 left
4468 (In Array : 1 In Tree : 0)
After remove 0 1726 left
1726 (In Array : 1 In Tree : 0)
After remove 1 4297 left
4297 (In Array : 2 In Tree : 1)
After remove 1 3357 left
3357 (In Array : 2 In Tree : 1)
After remove 0 4801 left
4801 (In Array : 1 In Tree : 0)
After remove 0 2002 left
2002 (In Array : 1 In Tree : 0)
After remove 2 4787 left
4787 (In Array : 3 In Tree : 2)
After remove 0 2309 left
2309 (In Array : 1 In Tree : 0)
After remove 1 768 left
768 (In Array : 2 In Tree : 1)
After remove 0 3443 left
3443 (In Array : 1 In Tree : 0)
After remove 0 4539 left
4539 (In Array : 1 In Tree : 0)
After remove 1 188 left
188 (In Array : 2 In Tree : 1)
After remove 0 4636 left
4636 (In Array : 1 In Tree : 0)
After remove 0 4375 left
4375 (In Array : 1 In Tree : 0)
After remove 0 3012 left
3012 (In Array : 1 In Tree : 0)
After remove 1 3190 left
3190 (In Array : 2 In Tree : 1)
```

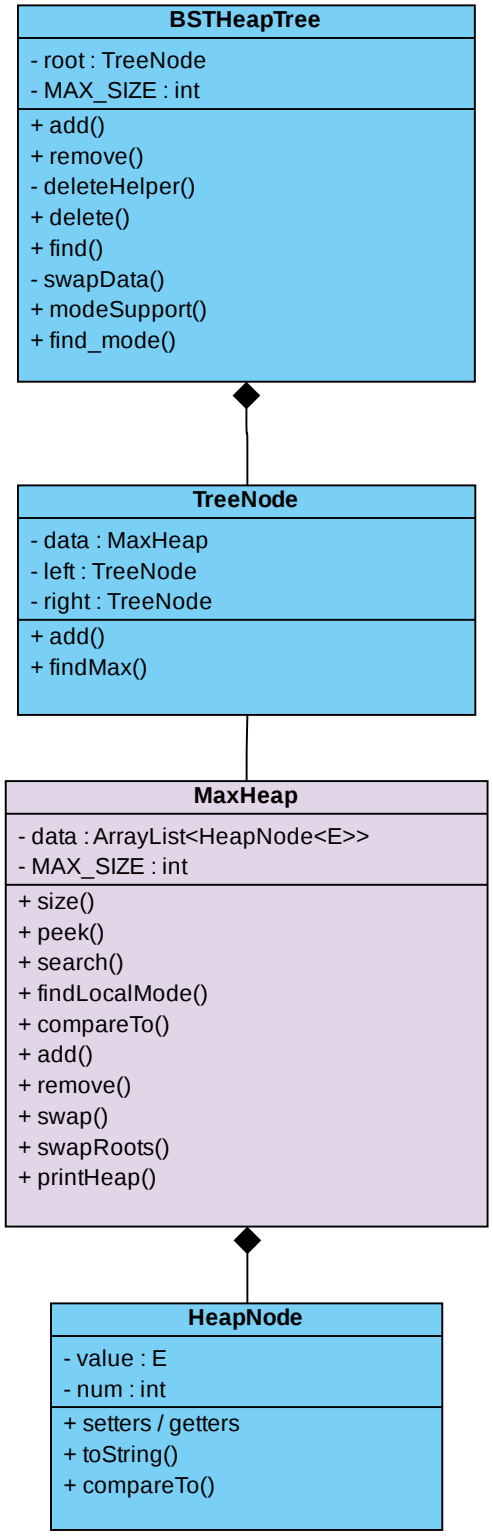
~/java/cse222/hayat

```
After remove 0 4168 left
4168 (In Array : 1 In Tree : 0)
After remove 0 2667 left
2667 (In Array : 1 In Tree : 0)
After remove 0 3403 left
3403 (In Array : 1 In Tree : 0)
After remove 0 923 left
923 (In Array : 1 In Tree : 0)
After remove 0 3584 left
3584 (In Array : 1 In Tree : 0)
After remove 1 949 left
949 (In Array : 2 In Tree : 1)
After remove 1 3539 left
3539 (In Array : 2 In Tree : 1)
After remove 0 3743 left
3743 (In Array : 1 In Tree : 0)
After remove 1 1457 left
1457 (In Array : 2 In Tree : 1)
After remove 2 763 left
763 (In Array : 3 In Tree : 2)
After remove 1 1714 left
1714 (In Array : 2 In Tree : 1)
After remove 0 2393 left
2393 (In Array : 2 In Tree : 0)
After remove 1 1029 left
1029 (In Array : 2 In Tree : 1)
After remove 0 4617 left
4617 (In Array : 1 In Tree : 0)
After remove 1 1733 left
1733 (In Array : 2 In Tree : 1)
After remove 1 519 left
519 (In Array : 2 In Tree : 1)
After remove 2 1309 left
1309 (In Array : 3 In Tree : 2)
After remove 1 4265 left
4265 (In Array : 2 In Tree : 1)
After remove 1 4895 left
4895 (In Array : 2 In Tree : 1)

TYPE ANYTHING TO CONTINUE TESTING:
REMOVING ELEMENTS THERE ARE NOT ON TREE:

5062 : (Before Remove)(In Tree : 0)
5062 : (After Remove)(In Tree : 0)

5010 : (Before Remove)(In Tree : 0)
5010 : (After Remove)(In Tree : 0)

5085 : (Before Remove)(In Tree : 0)
5085 : (After Remove)(In Tree : 0)

5037 : (Before Remove)(In Tree : 0)
5037 : (After Remove)(In Tree : 0)

5065 : (Before Remove)(In Tree : 0)
5065 : (After Remove)(In Tree : 0)

5024 : (Before Remove)(In Tree : 0)
5024 : (After Remove)(In Tree : 0)

5085 : (Before Remove)(In Tree : 0)
5085 : (After Remove)(In Tree : 0)

5075 : (Before Remove)(In Tree : 0)
5075 : (After Remove)(In Tree : 0)

5045 : (Before Remove)(In Tree : 0)
5045 : (After Remove)(In Tree : 0)

5055 : (Before Remove)(In Tree : 0)
5055 : (After Remove)(In Tree : 0)
```

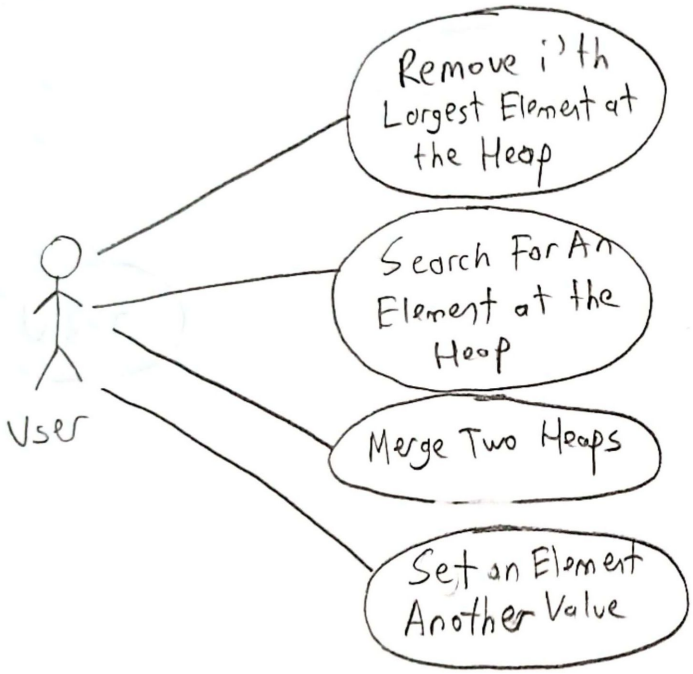
TESTING REMOVE AND FIND METHODS ON NON-EXISTING NUMBERS

DIAGRAMS:

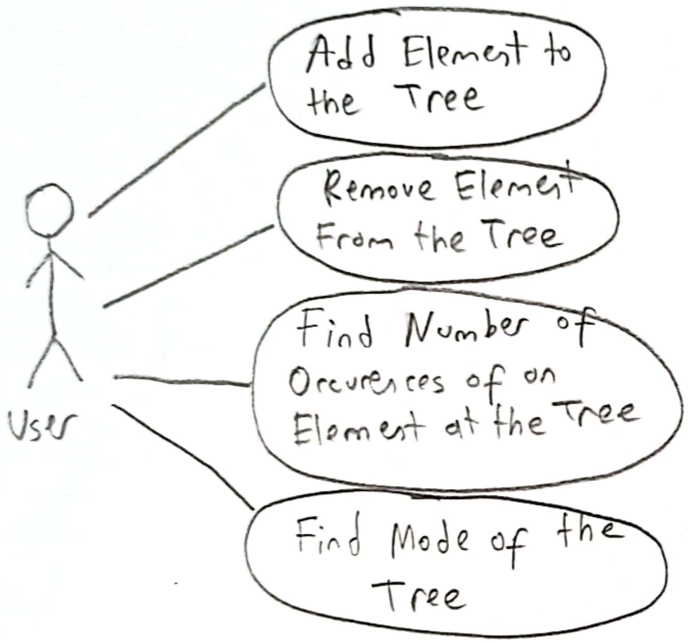




USE CASE DIAGRAM PART 1:



USE-CASE DIAGRAM PART 2:



PART 3:

PART 1:

I did calculate time complexitys of getter and setter methods.Since, their time complexity’s will be O(1).

C:\cygwin64\home\emr3s\java\cse222\hayat\Heap.java - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

Heap.java | MaxHeap.java | BSTHeapTree.java | driver.java

```java
* @return true if entry is on the heap, false if entry is not on the heap
*/
    public boolean search(E entry)
    {
        if(size() == 0) throw new IndexOutOfBoundsException("HEAP IS EMPTY");

        int length = size();
        boolean result = false;
        ArrayList<E> temp = new ArrayList<E>();

        for(int i = 0; i < length; i++)
        {
            E x = remove();) O(logn)
            temp.add(x);
            if(x.equals(entry))
            {
                result = true;
                break;
            }
        }

        for(int i = 0; i < temp.size(); i++)    add(temp.get(i));
        return result;
    }
```

$\Theta(1)$

$O(n \log n)$

$O(\log n)$

$T_w(n) = \Theta(n.\log n), T_B(n$

$T(n) = O(n.\log n)$

[Time complexity of remove and insert operations are O(n.logn)]

```java
/**
* Merges 2 Heaps
* Heap structure will be preserved after this operation.
* @param  entry Heap<E>
*/
    public void merge(Heap<E> entry)
    {
        ArrayList<E> temp = new ArrayList<E>();
        int length = entry.size();

        for(int i = 0; i < length; i++)
        {
            E x = entry.remove();) O(logn)
            this.add(x);
            temp.add(x);  O(log m)
        }

        for(int i = 0; i < temp.size(); i++)
        {
            entry.add(temp.get(i));
        }
    }
```

T(m,n) = O(max(logn, logm).n)

[m is size of this, n is size of entry]

```java
/**
```

Line 95, Column 16 | Tab Size: 4 | Java

Type here to search | ENG TRQ | 13:17 01/05/2021

C:\cygwin64\home\emr3s\java\cse222\hayat\Heap.java - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

Heap.java | MaxHeap.java | BSTHeapTree.java | driver.java

```java
        }
    }

/**
* Removes the index'th largest element from the heap.It calls remove method index times and inserts the removed elements back except the target
* @param  index int
*/
    public E removeithLargest(int index)
    {
        if(index <= 0 || size() == 0 || index >= size())    throw new IndexOutOfBoundsException();

        ArrayList<E> temp = new ArrayList<E>();
        E result;
        result = remove();
        if(index != 1) temp.add(result);

        for(int i = 2; i <= index; i++)
        {
            result = remove();
            if(i != index)
            temp.add(result);
        }

        for(int i = 0; i < temp.size(); i++)
        {
            add(temp.get(i));
        }
        return result;
    }

    ////////////////////////////////////////////////////////

/**
*  @author Emre Sezer
*  Iterator class for Heap
*/
    public class HeapIterator
    {
        private int current;

/**
* HeapIterator constructor
*/
        public HeapIterator()
        {
            current = 0;
        }

/**
* Method that returns true if current is smaller than size, else false
```

$$T_W(n) = \Theta(n \log n), \quad T_B = \Theta(1)$$

$$T(n) = O(n \log n)$$

Line 95, Column 16 | Tab Size: 4 | Java

Type here to search | ENG TRQ | 13:18 01/05/2021

C:\cygwin64\home\emr3s\java\cse222\hayat\Heap.java - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

Heap.java | MaxHeap.java | BSTHeapTree.java | driver.java

```java
            if(current >= 0 && current < size())    return true;
            else return false;
        }

/**
* Iterator next method
* @return the next element on the Heap
*/
        public E next()
        {
            E result = data.get(current);
            current++;
            return result;
        }

/**
* Iterator set method. Sets the current element to entry.Then sets Heap to Heap structure.
* After set operation Heap structure will be preserved.
* @param entry E
*/
        public void set(E entry)
        {
            data.set(current, entry);

            int child = current;

            while(2 * child + 2 < size())
            {
                if(data.get(2 * child + 1).compareTo(data.get(2 * child + 2)) > 0)  child = 2 * child + 1;
                else child = 2 * child + 2;
            }

            int parent = (child - 1) / 2;

            while(parent >= 0 && data.get(parent).compareTo(data.get(child)) < 0)
            {
                swap(child, parent);
                child = parent;
                parent = (child - 1) / 2;
            }

        }
    }

}
```

$\Theta(1)$

$T(n) = O(\log n)$

Line 95, Column 16 | Tab Size: 4 | Java

Type here to search

ENG TRQ 13:18 01/05/2021

C:\cygwin64\home\emr3s\java\cse222\hayat\Heap.java - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

Heap.java | MaxHeap.java | BSTHeapTree.java | driver.java

```java
            leftChild = (2 * parent) + 1;
            rightChild = leftChild + 1;

            if(leftChild >= size()) break;

            maxChild = leftChild;

            if(rightChild < size() && data.get(rightChild).compareTo(data.get(leftChild)) > 0)  maxChild = rightChild;

            if(data.get(parent).compareTo(data.get(maxChild)) < 0)
            {
                swap(parent, maxChild);
                parent = maxChild;
            }
            else break;

        }
        return result;
    }

/**
* Swaps x'th and y'th elements of the ArrayList
* @param  int x
* @param  int y
*/
    private void swap(int x, int y)
    {
        if(size() >= x && size() >= y)
        {
            E temp = data.get(x);
            data.set(x, data.get(y));
            data.set(y, temp);
        }
        else return;
    }

/**
* This is a method for printing Heap.It makes it makes testing easier.
* Prints the Heap's elements
*/
    public void printHeap()
    {
        if(size() == 0) System.out.println("HEAP IS EMPTY");
        for(int i = 0; i < size(); i++)
        {
            System.out.printf("%d ", data.get(i));
        }
    }

/**
```

Line 95, Column 16 | Tab Size: 4 | Java

Type here to search | ENG TRQ | 13:18 01/05/2021

PART 2:

I did calculate time complexitys of getter and setter methods.Since, their time complexity’s will be O(1).

MAXHEAP METHODS

C:\cygwin64\home\emr3s\java\cse222\hayat\MaxHeap.java - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

Heap.java | MaxHeap.java | BSTHeapTree.java | driver.java

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

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

```java
/**
* Method For Getting Top Element of the Heap
* @return  Top Element of the Heap
*/
    public HeapNode<E> peek()
    {
        if(size() == 0) return null;
        return data.get(0);
    }
```

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

```java
/**
* Searches for an element if it is stored on the heap.
* Support Method For BSTHeapTree's add Method.
* @param  entry E
* @return true if entry is on the heap, false if entry is not on the heap
*/
    public int search(E entry)
    {
        if(size() == 0) return 0;

        ArrayList<HeapNode<E>> temp1 = new ArrayList<HeapNode<E>>();
        ArrayList<Integer> temp2 = new ArrayList<Integer>();

        int result  = 0;
        HeapNode<E> x;
        int length = size();
        int nodeLength;

        for(int i = 0; i < length; i++)
        {
            x = remove();
            nodeLength = x.num + 1;

            if(nodeLength > 1)
            {
                for(int j = 0; j < nodeLength - 1; j++) remove();
            }

            temp2.add(nodeLength);
            temp1.add(x);

            if(entry.compareTo(x.getValue()) == 0)
            {
                result = nodeLength;
                break;
            }
        }

        for(int i = 0; i < temp1.size(); i++)
        {
            temp1.get(i).num = temp2.get(i);
            add(temp1.get(i));
        }

        return result;
    }
```

[n is size of this, m is nodeLength of i'th element]

$T(n,m) = O(\max(m, n\log(n)))$

Line 17, Column 45 | Tab Size: 4 | Java

Type here to search | ENG TRQ | 14:02 01/05/2021

C:\cygwin64\home\emr3s\java\cse222\hayat\MaxHeap.java - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

Heap.java | MaxHeap.java | BSTHeapTree.java | driver.java

```java
/**
*  Finds Mode of the MaxHeap
*  @return Mode of the MaxHeap
*/
    public HeapNode<E> findLocalMode()
    {
        if(size() == 0) return null;

        ArrayList<HeapNode<E>> temp1 = new ArrayList<HeapNode<E>>();
        ArrayList<Integer> temp2 = new ArrayList<Integer>();

        HeapNode<E> result;
        HeapNode<E> x;
        int length = size();
        int nodeLength;

        for(int i = 0; i < length; i++)
        {
            x = remove();
            nodeLength = x.num + 1;

            if(nodeLength > 1)
            {
                for(int j = 0; j < nodeLength - 1; j++) remove();
            }

            temp2.add(nodeLength);
            temp1.add(x);
        }

        for(int i = 0; i < temp1.size(); i++)
        {
            temp1.get(i).num = temp2.get(i);
            add(temp1.get(i));
        }

        result = temp1.get(0);
        for(int i = 0; i < temp1.size(); i++)
        {
            if(temp1.get(i).getNum() > result.getNum()) result = temp1.get(i);
        }
        return result;
    }

/**
*  compareTo Method
*/
    public int compareTo(MaxHeap other)
    {
        return other.peek().compareTo(peek());
    }

/**
* If heap has a node with value 'entry' . Incereases this nodes num by 1.
* If heap has not a node with value 'entry'.Inserts entry to end of the heap which is also end of the ArrayList
* @param  entry E
*/
    public int add(E entry)
    {
        return add(new HeapNode(entry));
    }
```

$\Theta(1)$

$\Theta(1)$

$O(n.m)$

$O(n)$

$O(n)$

[n is length, m is nodeLength of i'th element]

$$T(n,m) = O(n.m)$$

Line 17, Column 45 Tab Size: 4 Java

Type here to search

ENG TRQ 14:03 01/05/2021

C:\cygwin64\home\emr3s\java\cse222\hayat\MaxHeap.java - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

Heap.java | MaxHeap.java | BSTHeapTree.java | driver.java

```java
        return other.peek().compareTo(peek());
    }

/**
* If heap has a node with value 'entry' . Incereases this nodes num by 1.
* If heap has not a node with value 'entry'.Inserts entry to end of the heap which is also end of the ArrayList
* @param  entry E
*/
    public int add(E entry)
    {
        return add(new HeapNode(entry));
    }

    public int add(HeapNode<E> entry)
    {
        int child;
        int parent;

        for(int i = 0; i < size(); i++)
        {
            if(data.get(i).getValue().equals(entry.getValue()))
            {
                data.get(i).num++;
                return data.get(i).num;
            }
        }

        if(size() != 7)
        {
            data.add(entry);
            child = size() - 1;
            parent = (child - 1) / 2;

            while(parent >= 0 && data.get(parent).getValue().compareTo(data.get(child).getValue()) < 0)
            {
                swap(child, parent);
                child = parent;
                parent = (child - 1) / 2;
            }
            return 1;
        }
        else
        {
            System.out.println("Reached to MAX_SIZE");
            return -1;
        }
    }

/**
* Remove method for MaxHeap.If Head Node's Value's Occurences are bigger than 1.Decreases it by 1.
* If Head Node's Value's Occurence is 1. Removes the Head Node.
* @return root of the Heap which is 0'th element of the ArrayList
*/
    public HeapNode<E> remove()
    {
        if(data.size() == 0)    throw new IndexOutOfBoundsException();

        HeapNode<E> result;
        int parent;
        int leftChild;
        int rightChild;
        int maxChild;
```

$O(n)$

$O(\log n)$

$T(n) = O(n)$

Line 17, Column 45 Tab Size: 4 Java

Type here to search ENG TRQ 14:03 01/05/2021

C:\cygwin64\home\emr3s\java\cse222\hayat\MaxHeap.java - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

Heap.java | MaxHeap.java | BSTHeapTree.java | driver.java

```java
        }
    }

/**
* Remove method for MaxHeap.If Head Node's Value's Occurences are bigger than 1.Decreases it by 1.
* If Head Node's Value's Occurence is 1. Removes the Head Node.
* @return root of the Heap which is 0'th element of the ArrayList
*/
    public HeapNode<E> remove()
    {
        if(data.size() == 0)    throw new IndexOutOfBoundsException();

        HeapNode<E> result;
        int parent;
        int leftChild;
        int rightChild;
        int maxChild;

        if(data.get(0).num != 1)
        {
            data.get(0).num--;
            result = data.get(0);
        }
        else if (size() == 1)
        {
            data.get(0).num--;
            result = data.get(0);
            data.remove(0);
        }
        else
        {
            data.get(0).num--;
            result = data.get(0);
            HeapNode<E> x = data.remove(size() - 1);
            data.set(0, x);
            parent = 0;
            while(true)
            {
                leftChild = (2 * parent) + 1;
                rightChild = leftChild + 1;

                if(leftChild >= size()) break;

                maxChild = leftChild;

                if(rightChild < size() && data.get(rightChild).getValue().compareTo(data.get(leftChild).getValue()) > 0)    maxChild = rightChild;

                if(data.get(parent).getValue().compareTo(data.get(maxChild).getValue()) < 0)
                {
                    swap(parent, maxChild);
                    parent = maxChild;
                }
                else break;

            }

        }
        return result;
    }

/**
* Swaps x'th and y'th elements of the ArrayList
* @param  int x
```

$\Theta(1)$

$O(n)$

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

$O(\log n)$

$T(n) = O(n)$

Line 17, Column 45 Tab Size: 4 Java

Type here to search ENG TRQ 14:03 01/05/2021

C:\cygwin64\home\emr3s\java\cse222\hayat\MaxHeap.java - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

Heap.java | MaxHeap.java | BSTHeapTree.java | driver.java

```java
 * @param  int y
 */
    private void swap(int x, int y)
    {
        if(size() >= x && size() >= y)
        {
            HeapNode<E> temp = data.get(x);
            data.set(x, data.get(y));
            data.set(y, temp);
        }
        else return;
    }

/**
* Swaps Head's of the 2 MaxHeap
* This a support method for
* @param  other MaxHeap<E>
*/
    public void swapRoots(MaxHeap <E> other)
    {
        HeapNode<E> temp1 = peek();
        HeapNode<E> temp2 = other.peek();

        if(temp1 == null || temp2 == null)  return;

        int length1 = peek().getNum();
        int length2 = other.peek().getNum();

        for(int i = 0; i < length2; i++)
        {
            temp2 = other.remove();
        }

        for(int i = 0; i < length1; i++)
        {
            temp1 = remove();
        }

        temp1.num = length1;
        temp2.num = length2;

        add(temp2);
        other.add(temp1);
    }

/**
* Prints the Heap's elements
*/
    public void printHeap()
    {
        System.out.printf("NODE: ");
        for(int i = 0; i < size(); i++)
        {
            System.out.printf("(%d,%d) ", data.get(i).getValue(), data.get(i).getNum());
        }
        System.out.println();
    }

    ////////////////////////////////////////////////////////////

/**
*  Inner Class For MaxHeap
*  Stores Value and Number of Occurences
```

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

[m is size of this, n is size of other]

$O(n^2)$

$O(m^2)$

add(temp2); $\rightarrow O(n)$

other.add(temp1); $\rightarrow O(m$

$T(m,n) = O(\max(m^2, n^2))$

Line 17, Column 45 Tab Size: 4 Java

Type here to search ENG TRQ 14:03 01/05/2021

BSTHEAPTREE METHODS

C:\cygwin64\home\emr3s\java\cse222\hw4\BSTHeapTree.java - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

MaxHeap.java | BSTHeapTree.java

```java
 * @param item E
 * @return added element's number of occurences
 */
    public int add (E item)
    {
        int result = -1;
        TreeNode<E> localRoot = this.root;
        int searchResult = find(item);

        if(searchResult > 0)                // If that element already exists in the tree
        {
            TreeNode<E> tempRoot = root;

            while(true)
            {
                if(tempRoot.data.search(item) > 0)      // Searches if current node has that element.(Uses MaxHeap's Search Method)
                {
                    result = searchResult + 1;          // If found that element in the MaxHeap, increases that elements occurences by 1.
                    tempRoot.data.add(item);
                    break;
                }
                else if (item.compareTo(tempRoot.data.peek().getValue()) < 0)
                {
                    tempRoot = tempRoot.left;
                }
                else if (item.compareTo(tempRoot.data.peek().getValue()) > 0)
                {
                    tempRoot = tempRoot.right;
                }
            }
        }
        else                // If element doesn't exist in the tree
        {

        while(true)
        {
            if(root == null)                            // If there is a need for creating a new node
            {
                root = new TreeNode<E>();               // Creates a new node
                result = root.add(item);                // Adds entry to that new created TreeNode
                localRoot = root;
                break;
            }
            else if(localRoot.data.size() < MAX_SIZE)       // If appropriate node is not out of space, adds that element to the MaxHeap
            {
                result = localRoot.add(item);

                    while(localRoot.right != null && item.compareTo(localRoot.right.data.peek().getValue()) > 0)    // If item is bigger than right Node's head
                    {
                        localRoot.data.swapRoots(localRoot.right.data);             // It modifies the tree by changing heads of the current node and right node.
                                                                                    // This continues until current nodes is a leaf.
                    }

                break;
            }
            else
            {
                if(item.compareTo(localRoot.data.peek().getValue()) < 0 && localRoot.left == null)
                {
                    TreeNode<E> temp = new TreeNode<E>();
                    localRoot.left = temp;
                    localRoot = localRoot.left;

                }
                else if(item.compareTo(localRoot.data.peek().getValue()) < 0)   localRoot = localRoot.left;     // If left TreeNode exists, moves to left TreeNode
                else if(item.compareTo(localRoot.data.peek().getValue()) > 0 && localRoot.right == null)
                {
                    TreeNode<E> temp = new TreeNode<E>();
                    localRoot.right = temp;
                    localRoot = localRoot.right;
                }
                else if(item.compareTo(localRoot.data.peek().getValue()) > 0)   localRoot = localRoot.right;    // If right TreeNode exists, moves to right TreeNode
            }
        }
```

Line 66, Column 75 Tab Size: 4 Java

[n is total nodes in the tree]

$O(m.\log n)$

[Since, MAX_SIZE is 7, equations that use MaxHeap's size will take constant time.From that point i will not include MaxHeap's size in the equatrions.]

$O(\log n)$

$O(\log^2 n)$

$T(n) = O(\max(m\log n, \log^2 n))$

Type here to search

ENG TRQ 15:28 03/05/2021

C:\cygwin64\home\emr3s\java\cse222\hayat\BSTHeapTree.java - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

Heap.java | MaxHeap.java | BSTHeapTree.java | driver.java

```java
    public int remove (E item)
    {
        if(root == null) throw new IndexOutOfBoundsException();

        if(find(item) <= 0) return 0;

        TreeNode<E> localRoot = root;
        ArrayList<MaxHeap.HeapNode> temp1 = new ArrayList<MaxHeap.HeapNode>();
        ArrayList<Integer> temp2 = new ArrayList<Integer>();

        MaxHeap.HeapNode x;

        int result  = 0;
        boolean done = false;
        boolean flag = false;

        while(true)
        {
            if(localRoot.data.search(item) > 0)         // Searches is the entry exists in the current MaxHeap
            {
                int length = localRoot.data.size();
                int nodeLength;

                for(int i = 0; i < length; i++)
                {
                    if(localRoot.data.size() == 1 && localRoot.data.peek().getNum() == 1)
                    {
                        result = 0;                                     // This part is for : If the item is the only element in the MaxHeap and it's occurence is 1.
                        this.root = delete(this.root, localRoot.data);      // When it deletes that element from the MaxHeap, there will be no element left on the MaxHeap.So, the empty TreeNode will be deleted.
                        return result;
                    }

                    x = localRoot.data.remove();
                    nodeLength = x.getNum() + 1;

                    if(item.equals(x.getValue()))                   // If item is found in the current MaxHeap
                    {
                        if(i == 0 && x.getNum() == 0) flag = true;      // If item is at the top of the MaxHeap
                        result = x.getNum();
                        done = true;
                        break;
                    }

                    if(nodeLength > 1)
                    {
                        for(int j = 0; j < nodeLength - 1; j++) localRoot.data.remove();
                    }

                    temp2.add(nodeLength);
                    temp1.add(x);

                }

                for(int i = 0; i < temp1.size(); i++)       // It addes the removed HeapNodes back to the MaxHeap
                {
                    temp1.get(i).setNum(temp2.get(i));
                    localRoot.data.add(temp1.get(i));
                }
                if(done)    break;

            }
            else if(done)   break;
            else if(item.compareTo(localRoot.data.peek().getValue()) < 0)   // It moves current TreeNode to the left Node of the current TreeNode
            {
                localRoot = localRoot.left;
            }
            else if(item.compareTo(localRoot.data.peek().getValue()) > 0)   // It moves current TreeNode to the right Node of the current TreeNode
            {
                localRoot = localRoot.right;
            }

        }
```

$O(m.\log n)$

$\rightarrow O(n)$

$\Theta(1)$

$T(m,n) = O(m.n \log n)$

Line 318, Column 50 | Tab Size: 4 | Java

Type here to search | ENG TRQ | 14:55 01/05/2021

C:\cygwin64\home\emr3s\java\cse222\hayat\BSTHeapTree.java - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

Heap.java | MaxHeap.java | BSTHeapTree.java | driver.java

```java
*/
    private TreeNode deleteHelper(MaxHeap data, TreeNode root)
    {
        if(root == null)    return null;

        else if(data.compareTo(root.data) < 0)  root.left = deleteHelper(data, root.left);
        else if(data.compareTo(root.data) > 0)  root.right = deleteHelper(data, root.right);
        else
        {
            if(root.left == null && root.right == null) root = null;
            else if(root.left == null || root.right == null)
            {
                root = root.left == null ? root.right : root.left;
            }
            else
            {
                TreeNode predecessorNode = root.left.findMax();
                swapData(predecessorNode, root);
                root.left = deleteHelper(predecessorNode.data, root.left);
            }
        }

        return root;
    }

/**
* Modified Delete Method For BinarySearchTree. It uses Helper method
* This method is for removing node from tree and still keeping the tree order
* @param data MaxHeap
* @param root TreeNode
* @return TreeNode
*/
    public TreeNode delete(TreeNode node, MaxHeap data)
    {
        return node = deleteHelper(data, node);
    }


    ////////////////////////////////////////////////////////////


/**
* Find Method For BSTHeapTree.It returns number of occurences of the item
* @param item E
* @return number of occurences of the item
*/
    public int find(E item)
    {
        if(root == null) return 0;

        TreeNode<E> localRoot = root;
        int result = 0;

        while(true)
        {
            if(localRoot == null)
            {
                result = 0;
                break;
            }

            result = localRoot.data.search(item);
            if(result > 0) break;
            else if (item.compareTo(localRoot.data.peek().getValue()) < 0)
            {
                localRoot = localRoot.left;
            }
            else if (item.compareTo(localRoot.data.peek().getValue()) > 0)
            {
                localRoot = localRoot.right;
            }

        }
```

$T(n) = O(\log n)$

$T(n) = O(\log n)$

$T(m,n) = O(m \cdot \log n)$

Line 185, Column 1 | Tab Size: 4 | Java

Type here to search | ENG TRQ | 15:18 01/05/2021

C:\cygwin64\home\emr3s\java\cse222\hayat\BSTHeapTree.java - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

Heap.java | MaxHeap.java | BSTHeapTree.java | driver.java

```java
/**
* Support Method For delete, swaps 2 TreeeNodes data
* @param node1 TreeNode
* @param node2 TreeNode
*/
    private void swapData(TreeNode node1, TreeNode node2)
    {
        MaxHeap temp = node1.data;
        node1.data = node2.data;
        node2.data = temp;
    }

/**
* Support Method For find_mode()
* @param LocalRoot TreeNode <E>
* @param LocalModeList ArrayList<MaxHeap.HeapNode>
*/
    public void modeSupport(TreeNode <E> LocalRoot,ArrayList<MaxHeap.HeapNode> LocalModeList)
    {
        if(localRoot == null)   return;

        modeSupport(localRoot.left, localModeList);
        modeSupport(localRoot.right, localModeList);

        localModeList.add(localRoot.data.findLocalMode());
    }

/**
*    Method for finding mode
*/
    public void find_mode()
    {
        ArrayList<MaxHeap.HeapNode> localModeList = new ArrayList<MaxHeap.HeapNode>();
        modeSupport(root, localModeList);
        MaxHeap.HeapNode mode = localModeList.get(0);
        int max = 0;

        for(int i = 0;i < localModeList.size(); i++)
        {
            if(localModeList.get(i).getNum() > max) max = localModeList.get(i).getNum();
        }

        for(int i = 0; i < localModeList.size(); i++)
        {
            if(localModeList.get(i).getNum() == max) System.out.println(localModeList.get(i).getValue() + " with " + localModeList.get(i).getNum() + " occurences");
        }

    }

    ////////////////////////////////////////////////////////

/**
*  Inner class for representing Nodes of the Tree
*  It keeps Maxheap for data, left and right TreeNodes for children
*/
    protected class TreeNode <E extends Comparable<E>>
    {
        public MaxHeap <E> data;
        public TreeNode <E> left;
        public TreeNode <E> right;

/**
*  Constructor
*/
        public TreeNode()
        {
            data = new MaxHeap();
            left = null;
            right = null;
        }

/**
```

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

$$T(n,m) = O(\log n \cdot m)$$

$$T(n,m) = O(\log n \cdot m)$$

Line 231, Column 1 Tab Size: 4 Java

Type here to search ENG TRQ 15:20 01/05/2021

C:\cygwin64\home\emr3s\java\cse222\hayat\BSTHeapTree.java - Sublime Text (UNREGISTERED)

File Edit Selection Find View Goto Tools Project Preferences Help

Heap.java | MaxHeap.java | BSTHeapTree.java | driver.java

```java
            if(localModeList.get(i).getNum() > max) max = localModeList.get(i).getNum();
        }

        for(int i = 0; i < localModeList.size(); i++)
        {
            if(localModeList.get(i).getNum() == max) System.out.println(localModeList.get(i).getValue() + " with " + localModeList.get(i).getNum() + " occurences");
        }

    }

    ////////////////////////////////////////////////////

/**
*  Inner class for representing Nodes of the Tree
*  It keeps Maxheap for data, left and right TreeNodes for children
*/
    protected class TreeNode <E extends Comparable<E>>
    {
        public MaxHeap <E> data;
        public TreeNode <E> left;
        public TreeNode <E> right;

/**
*  Constructor
*/
        public TreeNode()
        {
            data = new MaxHeap();
            left = null;
            right = null;
        }

/**
*  add Method for TreeNode
*  @param entry E
*  @return number of occurences of the entry
*/
        public int add (E entry)
        {
            return data.add(entry);
        }

/**
*  Support method for delete method
*  @param entry E
*  @return Treenode
*/
        public TreeNode findMax()
        {
            if(right != null)   return right.findMax();
            return this;
        }

    }

}
```

$$T(n) = O(\log n)$$

Line 314, Column 10 Tab Size: 4 Java

Type here to search ENG TRQ 15:29 01/05/2021