Homework 5

CSE 274
Deadline: 11/20/2022

I. A Max Heap based on Arrays

A heap data structure can be implemented using arrays. Such a data structure has a time complexity of O(log(n)) for insertion and deletion of data. In this lab we implement a max heap using arrays. When implementing a heap using arrays, there is no tree or node involved in the implementation. Still, we make a correspondence between indexes of the `InternalArray` and nodes of a binary tree to facilitate the implementation.

II. the ArrayMaxHeap class

The `ArrayMaxHeap` class has an `InternalArray`, the `size` of which is set by the the constructor of the class:

```
ArrayMaxHeap(int theSize)
{
LastIndex=0;

size=theSize;

InternalArray =new int[size];

for (int i=0; i<size;i++)
      InternalArray[i]=-1;
}
```

The constructor of the class initializes all entries of `InternalArray` to `-1`. Data elements are stored in indexes 1, 2, ... of the `InternalArray`. The index zero of the `InternalArray` does not store any data. Also, the `LastIndex` variable is the last index of `InternalArray` that stores a data. Accordingly, `LastIndex` is initialized to zero.

As part of its output, the `display` method of the `ArrayMaxHeap` shows a tree that is corresponding to the `InternalArray`. This tree is not an actual tree made up of nodes.

III. The add Method

The `add(value)` method of the `ArrayMaxHeap` class is called to insert data into `InternalArray`. The `add` method calls the `route(LastIndex)` method to obtain the path of `nodes` that would have been visited and rearranged in the corresponding binary tree if the `value` was to be added to the corresponding tree. Then, the `add` method goes over those indexes of `InternalArray` which are corresponding to the nodes in the said path of the corresponding tree, and rearranges the `value`-s in those indexes. The `route` method is the same method we used in previous labs.

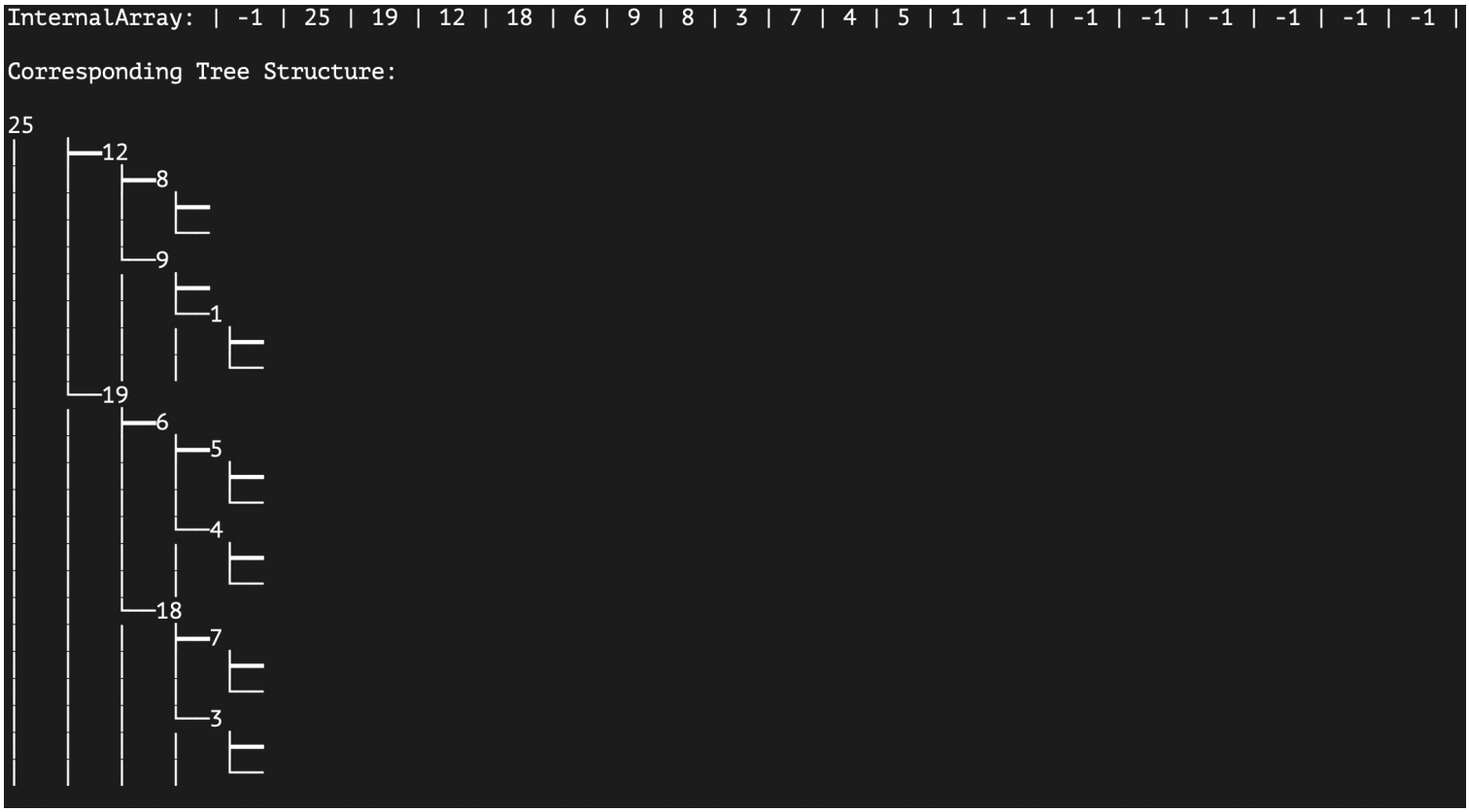
Implement the `add` method using the following logic:

```
public void add(int value)
LastIndex=LastIndex +1

if(LastIndex==1)

      InternalArray[LastIndex]=value;
      return

if(LastIndex==2)

      if (InternalArray[1]<value)
            Swap InternalArray[1] and value

      InternalArray[LastIndex]=value
      return

if(LastIndex==3)

      if (InternalArray[1]<value)
            Swap InternalArray[1] and value

      if (InternalArray[2]<value)
            Swap InternalArray[2] and value

      InternalArray[LastIndex]=value
      return

int index=1
int level = 1
String[] Direction = route(LastIndex)
while(level<Direction.length)

      if (InternalArray[index]<value)
            Swap InternalArray[index] and Value

      if (Direction[level].equals("DownWard"))
            index= index*2

      if (Direction[level].equals("UpWard"))
            index= index*2 +1

      level=level+1

InternalArray[index]=value;
return
```

Insertion of data at index=1, index=2 and index=3 of the `InternalArray` are special cases and are processed without calling the `route` method. Use the following lines of code to test the developed method:

```
ArrayMaxHeap myHeap = new ArrayMaxHeap(20);
myHeap.add(25);
myHeap.add(19);
myHeap.add(1);
myHeap.add(3);
myHeap.add(4);
myHeap.add(8);
myHeap.add(9);
myHeap.add(18);
myHeap.add(7);
myHeap.add(6);
myHeap.add(5);
myHeap.add(12);
myHeap.display();
```

The expected output is printed below:



As it can be seen above, the index zero of the `InternalArray` keeps the initial value of -1 as no data is stored at this index.

IV. The Delete method

The `delete` method of the `ArrayMaxHeap` class is called to delete and `return` the data residing at index=1 of the `InternalArray`. The `delete` method rearranges the values in certain indexes of the `InternalArray` including the `LastIndex`, so as to keep the structure of the corresponding tree a Max Heap data structure. Implement the `delete` method using the following logic:

```
public int delete ()

int DeletedValue=InternalArray[1]

if (LastIndex==1)
      LastIndex=0
      return DeletedValue

if (LastIndex==2)
      InternalArray[1]=InternalArray[2]
      LastIndex=1
      return DeletedValue

if (LastIndex==3)

      if (InternalArray[2]<=InternalArray[3])
            InternalArray[1]=InternalArray[3]

      if (InternalArray[2]>InternalArray[3])
            InternalArray[1]=InternalArray[2]
            InternalArray[2]=InternalArray[3]

      LastIndex=2
      return DeletedValue

int index=1
int lastValue=InternalArray[LastIndex]

while(true)

      if (2*index+1> LastIndex)
            InternalArray[index]=lastValue
            break

      if (InternalArray[2*index+1]>= InternalArray[2*index])
            InternalArray[index]=InternalArray[2*index+1]
            if (InternalArray[index]<lastValue)
                  Swap InternalArray[index] and lastValue
            index=2*index+1
            continue

      if (InternalArray[2*index+1]< InternalArray[2*index])
            InternalArray[index]=InternalArray[2*index]
            if (InternalArray[index]<lastValue)
                  Swap InternalArray[index] and lastValue
            index=2*index
            continue

InternalArray[LastIndex]=-1;
LastIndex=LastIndex-1
return DeletedValue
```

Use the following lines of code to test the developed method:

```
ArrayMaxHeap myHeap = new ArrayMaxHeap(20);
myHeap.add(25);
myHeap.add(19);
myHeap.add(1);
myHeap.add(3);
myHeap.add(4);
myHeap.add(8);
myHeap.add(9);
myHeap.add(18);
myHeap.add(7);
myHeap.add(6);
myHeap.add(5);
myHeap.add(12);

while (!myHeap.isEmpty())
      System.out.println(myHeap.delete());
```

The expected output is printed below:

```
25
19
18
12
9
8
7
6
5
4
3
1
```

V. Submitting the Assignment

As it can be seen above, the `delete` method of the `ArrayMaxHeap` class `return` the data elements of the `InternalArray` in a decreasing order. When submitting your response to this assignment, keep the above lines of code in the body of the `main` method.