Lab Worksheet 5: Heaps

The goal of this worksheet is:

- To gain better understanding of how a heap can be used to implement a priority queue.
- To understand Java API documentation.
- To write code to implement the key methods of a Proper Binary Tree.
- To gain skills that will be necessary for the programming assignment that begins soon.

Download the `W5-Source.zip` file from Moodle import it into Eclipse.

Like Worksheet 4, this project contains a "`doc`" folder that contains the Javadoc you will need for this lab (to explore this, open the "`index.html`" file). Pay special attention to the `ICompleteBinaryTree` interface in the `dsa.iface` package.

There are two classes in the project:

- `Heap` will become a full implementation of a heap, once you have finished this lab. Your code will go in this file.
- `HeapTest` contains some code to help you to test your implementation.

Part 1: Heap operations

The HeapTest class performs insert operations for the following values:

100, 20, 24, 87, 18, 35, 90, 22, 63, 40, 15, 62, 66, 72, 71, 33, 44, 35, 28, 68

In the code the same value is used for the priority as well as to element to keep it simpler.

Firstly, calculate what the heap will contain after entering all of these values.

Part 2: Read the code!

The `Heap` class already contains some code to help you. Some features it has are:

There is a member variable that is of type `ICompleteBinaryTree`. This will store all of the data

A `HeapEntry` inner class stores a priority and an element. Instances of this class will should be stored in the complete binary tree.

The `insert(…)` and `remove(…)` methods have already been written for you, but rely on some methods that are not implemented.

Part 3: Implement the methods

You should implement the upheap(…) and downheap(…) methods.

Hint: Because the priorities are Comparable, you can use the `compareTo(…)` method from Worksheet 2 to check when an `upheap` or downheap operation is needed.

Hint: Both methods can be implemented using either loops or recursion. Personally, I (David) prefer the recursive approach for this problem, but you may prefer to use loops. Perhaps you can try both.

Hint: Because the `CompleteArrayBinaryTree` class implements the `ITree` interface, you can use the `TreePrinter` class to see what your tree contains at any time.

If you have implemented these correctly, then running the HeapTest main method should result in the numbers being printed in order, from smallest to largest.