CSC220 Lab 12
Heaps

The goal of this week's assignment is:

1. Practice using heaps
2. Learn about the importance of debugging

Things you must do:

1. There are many details in this assignment. Make sure you read the whole thing carefully before writing any code and closely follow this instruction.
2. You must complete your assignment **individually**.
3. Always remember Java is case sensitive.
4. Your file names, class names, package name, and method signatures must match exactly as they are specified here.

Things you must not do:

1. You must not change the file names, class names, package names.
2. You must not change the signature of any of these methods (name, parameters, …).
3. You must not create any different class (other than the ones instructed).
4. **DO NOT modify the implementation of the methods provided.**

Important Note: You are going to write a few functions this week but there are many details involved. Start early! We are providing the tests cases we are going to use to grade your assignment. So, focus on getting your code to work properly.

Part 0

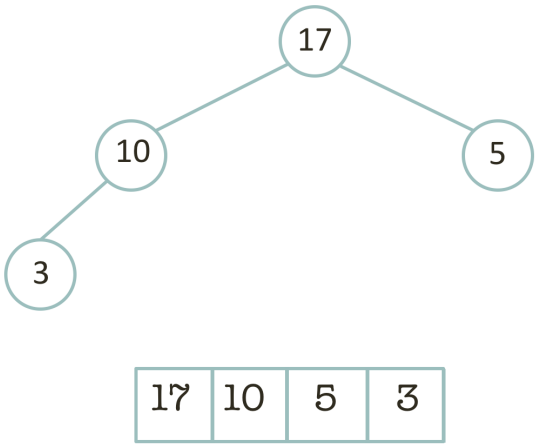
- You are going to create a new project for this (and each of the remaining labs). You learned how to create a project in Eclipse before. Create a new Java project and call it **Lab12** (with no space and no other name – notice the capital 'L')
- Create a package inside your project and call it **lab12** (no space and no other name – all lowercase).
- Create another class (or Java file) in this package and call it: **MaxHeap.java** (nothing else). You will be working on developing methods for this class. This class represents a max heap. This class has been partially implemented for you. The current implementation provides the fie

Part 1 – The problem description

DO NOT MOVE TO NEXT STEPS BEFORE READING THIS PART CAREFULLY

For this lab, you are asked to write methods to complete the implementation of a binary max heap class. In a binary max heap, each node can have at most two children, and the data in the parent node must be greater than or equal to all of its decedents. Remember that like min heaps, there is no relationship between the data values of the siblings of a specific parent (i.e., the right child does not have to be greater or smaller than the left child). The max heap in this case is nothing but an array. Recall from the lectures, for implementing a binary tree (or in this case a heap) using an array, we need the following rule: for any node at index i, its two children are at index (i*2)+1 and (i*2)+2. The functions to access the parent, left child or right child of a node has been implemented for you.

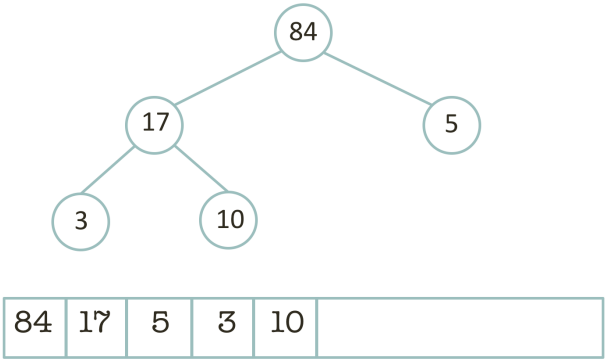
Given an array of N values, a heap containing those values can be built, in situ or in place, by simply "sifting" each internal node down to its proper location. You have seen the opposite behavior during the lecture for a min heap. For example, imagine that we want to add 84 to the following heap that is represented using the following array:



In order to insert 84 in this max heap, we start by putting the value 84 at the last internal node possible. Which one is it?!

Now, we need to make sure whether it is in its current position, meaning if the value of its parent is smaller than 84, we need to swap the current node (84) with its parent till the max heap property is satisfied. Before you continue, take a moment and think about how the array (representing the max heap) should look like after this insert.

Here is how it should look like after the insert:



You need to convince yourself why the internal nodes have been moved around before you continue.

Similar to lab 9, We provided a simple variation of in-order traversal of a tree (suggested in the book – Chapter 17.2 – page 1031): “instead of printing values all on a line, we will print them one per line and use indentation to indicate the level of the tree”. For example, the previous heap can be printed using a call to printSideways():

```
printSideways();
```

The output looks like the following:

```
                0
        0
                0
    5
                0
        0
                0
84
                0
        10
                0
    17
                0
        3
                0
```

The extra zero values are due to the fact that the max heap has been initialized so that it can contain 15 elements.

You need to imagine rotating this output 90 degrees in a clock-wise fashion (or tilting your head to the left) to see the tree structure. This function can come handy. There is also another print method that would simply print the heap array called: `printArray`.

Before you move on to the next part, take a couple of minutes and look at the definition of the methods provided for you and the fields of this class. Remember that you ARE NOT allowed to change the signature of the methods. However, you are allowed to use helper functions if you need to. **If you change the signature of any method in this assignment you WILL be penalized up to 30% of your total grade. For this week, you do not need to worry about the array containing the heap run out of space (keep this in mind, when you are testing your code – not to add more than the array can contain).**

For this week you are required to implement the following methods (only the first few during the lab):

1. `public MaxHeap(int maxsize)`

2. `public void insert(int element)`
3. `public int removemax()`
4. `public void sort()`
5. `public MaxHeap(int[] arr)`

Part 2 – Simple constructor

By the end of this week's assignment you are going to implement two constructors for this class. We start with the easy constructor for the lab. The signature of this construction MUST be the following (only modify where it says "FILL IN" – DO NOT change the signature):

```
public MaxHeap(int maxsize)
```

As the signature of this method suggests, this is a constructor for this class. This method will initialize the status of the objects of this class based on the input parameter (i.e., size). So, this function will create the array (heap) with the specified size and initialize all of its elements to be zero. You need to be careful about whether any other field needs to be initialized at this stage (hint: think about what the value of size should be).

Part 3 – insert method

The signature of this method should be (only modify where it says "FILL IN" – DO NOT change the signature):

```
public void insert(int element)
```

This function will insert the given value (element) into the heap and position it into the correct location. Go back to the problem description part and make sure you are following the steps given EXACTLY. The element is supposed to be added to the latest position available in the heap array and then you must make proper changes to the heap structure to maintain the max heap

property. DO NOT forget to update the size value. You are encouraged to write helper functions to make your code more comprehensible (not required).

Part 4 – removemax method

The signature of this method should be (only modify where it says "FILL IN" – DO NOT change the signature):

```
public int removemax()
```

This function will remove the **maximum** value stored in the max heap (only the maximum and nothing else!). Recall that by definition, this value would be stored in the root. We are not providing the algorithm here for you. But, the logic is similar to removing min from the min heap covered during the class. To implement this function, start by swapping the root with the last leaf, sift the new root down to restore heap property. If the heap is empty this method should return -1.

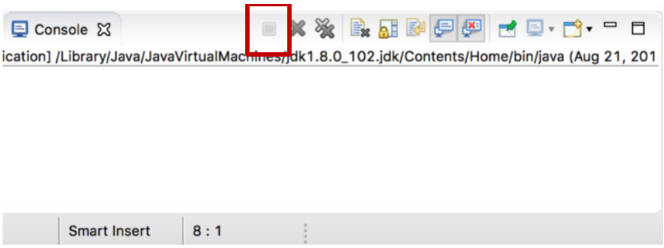
Part 5 – Test your code

As usual you need to test the functionality of the methods you have implemented. A set of test cases has been provided for you as part of **MaxHeap.java**. Uncomment the lab portion of the tests and run the main function. If you see any red text that says "TEST FAILED", you need to debug your code.

How to debug your code?

1. Use the Eclipse debugger you learned about during the first lab
2. If you see JavaStackOverflow, that means that you have an infinite recursive call and your recursive call is filling up the "call stack" (we talked about this concept in class). Go back and debug the method that is causing the problem.
3. Infinite loops! How would you know you have an infinite loop? As you should know from CSC120, if you have an infinite loop in your code, your code will not stop running. An

easy way to inspect that in Eclipse is to look at your console window, if your code is done running the console should look like the following



If the little square marked above is red and continues to stay red, that means your code has an infinite loop!

Make sure to grab your code before your leave the lab, you will develop more functionality for the classes for your assignment. You are required to submit the code you wrote during the lab as part of your assignment.

For all your assignments, please start early and seek help early (either from the instructor or the TAs).