**CSC220 Assignment10**

**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, and package name must match exactly as they are specified here.

5. Your project must include the methods you implemented in the lab.

**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, …). Just fill in the missing code inside them.

3. You must not create any different class.

**DO NOT** start your assignment unless you have all the features in the lab working!

**Part 1 –** sort **method**

The signature of this method should be:

public void sort(int[] arr)

This function receives an input array arr and provides an in-place heap sort. That means you are not allowed to have **any auxiliary** array. This method invokes two routines, heapify() and siftDown(). We have provided a starting point for you by implementing sort() and heapify(). **To complete the implementation, you must** **write** siftDown().

Since the data is initially unordered and most likely violates the rules for a valid max heap, it must first be heapified by calling heapify(). This method starts at the ***last*** ***parent node*** and it continually sifts down the smallest descendant at each level byinvoking siftDown(), where this method is passed the index to sift down from. If a child (left or right child of index) has a larger value than the parent (index), the largest child value is swapped with the parent. That node is then sifted down again until no sift occurs, or there are no more children. heapify() calls siftDown() on each parent node, from the last parent until the root, at which point the heap structure is valid. This means the largest value will be at the root of the tree, and every parent will have a value greater than or equal to its descendants.

Now that the heap structure has been built, the largest element is the root. This is swapped with the last leaf node so that it is now in the last position in the array. The trick here is that once the largest element is stored in its final position, size is decremented so that it will remain untouched for the remainder of the sorting.

The heap from the root is then sifted down (by calling siftDown(0); why is 0 passed in as the parameter?), excluding the last position (how?). This process is repeated until all of the nodes except the root have been excluded. At this point, the data is sorted.

You should work out with pen and paper how the algorithm operates, running through the code that is already provided for you. Also think about the task of siftDown() and how it fits into the code (after all, you will be implementing it :-).

**Part 2 – advanced** MaxHeap **constructor**

The signature of this method should be:

**public** MaxHeap(**int**[]arr)

This is a constructor for the max heap class that initializes a heap and put the values of the input array in it in such a way that the constructed max heap is valid. Hint: think about how you can reuse the functions you have implemented so far.

**Part 3 – Test your code**

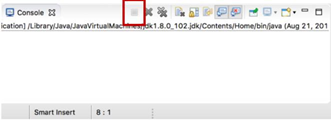
As usual you need to test the functionality of the methods you have implemented. A set of test cases has been provided for you as part of **MaxHeap.java**. Uncomment the assignment portion of the tests and run the main function. If you see any red text that says “TEST FAILED”, you need to debug your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

**Remarks**

* Make sure to submit your assignment by (re-)uploading your **Lab10** folder into your **csc220-cXXXX** folder by the deadline.
* **For all your assignments, please start early and seek help early (either from the instructor or the TAs).**