COMP 3160 – Fall 2014
Homework 3: Heaps

Number of People: Individual. Feel free to ask me for help, or visit the Computer Science Learning Center (www.cs.memphis.edu/cslc).

Due: Thurs., Sept. 25 by 1:00 pm

Submission: Zip all of your Java source files (you can zip the entire project folder if using an IDE) into a single file and upload it to the proper folder in the eCourseware dropbox at https://elearn.memphis.edu. The non-coding portions can be submitted within your zip file, or as a hard copy at the beginning of class.

Coding Style: Use consistent indentation. Use standard Java naming conventions for **variableAndMethodNames**, **ClassNames**, **CONSTANT_NAMES**. Include a reasonable amount of comments.

Grader: TA, Kyle Cherry (kcherry2@memphis.edu). Questions about grading? Please contact him first!

In class we focused on the *min-heap*, which requires each node to be smaller than both of its children. In problems 1 and 2, you'll be working with its opposite, the *max-heap*.

1. **(5 pts)** Draw the max-heap that results from adding the following elements to an initially empty heap, in the specified order.

   6, 3, 8, 10, 12, 4, 0, 5, 7

2. **(5 pts)** Draw what the heap looks like after removing the top element twice (show the heap after each removal).

3. **(15 pts)** Non-honors students: pick any 2 coding problems *(7.5 pts each)*. Honors students: do all 3 *(5 pts each)*.

   a. A heap is not the only way of implementing a priority queue. For example, you could simply use a linked list to store the elements in the queue. If you always enqueue new elements by inserting them at the appropriate location to ensure that the list is sorted by priority, the head node will always be storing the highest-priority element. Thus, a dequeue would require just removing the head node from the list.

      Write a **PriorityQueueLL** class that uses a linked list to implement a priority queue. You don't have to write the linked list from scratch – feel free to use **java.util.LinkedList**. Experiment with timing the enqueue and dequeue operations using large list-based and heap-based queues. What is the Big-O of your enqueue and dequeue operations for the list-based queue? Explain how you got your answers.

   b. Heaps are used as the basis for an efficient sorting algorithm known as (surprise) *heapsort*. Conceptually, heapsort is very easy to explain. To sort an array of elements, simply go through the array and insert each element into a min-heap. Then repeatedly remove the top element from the heap, placing it back into the array. Write and test a **heapsort(Integer[] a)** method that sorts the specified array of integers by using the **PriorityQueue** class we wrote in lecture. What is the Big-O of your method? Explain how you got your answer.

   c. In the **PriorityQueue** class that we wrote in lecture, rewrite the **toString** method to display the elements of the heap organized by their level in the tree, with one line per level. For example, if the data array contains 0 8 4 10 24 7 5, **toString** should return a string formatted like this:

```
0
8 4
10 24 7 5
```