CS 361 – Lab #6 – Heap implementation

Today we will try our hand at implementing a heap data structure. There is actually not much work to do. The purpose of a heap is to realize the priority queue abstract data type. This means we need to pay attention to 2 major functions: inserting into the heap, and removing the root (i.e. smallest) value. Before you leave today, please demonstrate your program for me, to be sure it can handle both insertions and deletions.

1. To begin, in your account create a new directory called lab06. Copy my code from http://cs.furman.edu/~chealy/cs361/lab06/. You will be spending most of your effort finishing the Heap class. Comments in this file will give you hints on how to get started on each operation.

2. Run the driver program. This program first sets up a new heap object. Then it tests Heap's insert method several times, and then it tests the remove operation several times. Since these functions have not yet been implemented, the heap stays empty. Feel free to change the input values in the array, as well as how many numbers get put into the heap – anywhere from 0 to 31 values, inclusive. For your convenience, I have provided a toString() method that gets called after each insertion and deletion in the driver program.

3. Now, the fun begins. Open Heap.java and complete the implementation of the insert() method. you are done, compile and run your program. Test with small input sets first. When testing insert(), the interesting case occurs when we have input values that must float up the heap. So, try an input array that begins inserting large numbers, followed by smaller ones.

4. Next, complete the implementation of removeMin(). This method may be slightly more complex than insert() because when we float down the heap, we must compare with both children to see which one we need to swap with. When you are done, compile and run the driver program. Looking at the output, make sure the removal and heapify operations are done correctly. You can also take this opportunity to make sure you understand how to do this by hand. In particular, be sure to test a situation where a parent needs to compare itself with its only child.

5. You're done! If you have time left, try to implement the "bottom up heap" building strategy we saw in class. Even though this is recursive, it's is a more efficient way to create a heap out of a sequence of values, O(n) rather than O(n log n). Don't be concerned if you don't finish this part.