CS 222 Assignment 3

Thomas Trimble

Binary Heap

For this assignment, I decided to implement an array-based implementation of a Max Binary Heap. For its instance variables, I decided to keep track of the number of values currently in the array, the index of the last user-added value in the array, and a few attributes about the array itself to make it easier to edit, such as what the initial size the array should be, how much the size should be multiplied by when resizing, and the current array size. Initially, I thought it would be easier to create a “Node” object for the values in the heap to keep track of their children and its index, but as I was brainstorming how to implement the insert() method, I realized that it would be redundant, as I would already have easy access to all of the information I needed.

insert()

Of all of the methods I had to create, this took the longest. As I was going into this, I thought that I could simply insert the value at the top, and bubble the old value down to its proper place. However, this presented an issue: I would need to know which values to swap, and that required knowing the path in the tree to where the final value was. This would require me to create some sort of algorithm to calculate it, but as I started to think about how that would work, I realized how much time and effort it would take. So, I paused and started brainstorming with a sheet of paper. This is computer science, where every problem has 5 solutions; Surely, there was a simpler way. After about 10-15 minutes, I realized that rather than starting at the top and working my way down, I could start at the bottom and work my way up, comparing the value to its parent and, if necessary, swap them. It was far easier to implement, and it accomplished the task I needed it to.

display()

During this brainstorming session, I realized that the array listed the numbers in level order, and that each level had double the amount of values compared to the one previous to it. I used this information to create the display() method, which simply prints the array with dividers between the levels, using a while loop and an incrementer, which triggers a level threshold at the appropriate times to print the dividers.

extractMax()

Because I had already created a way to traverse the tree through its parents and children, this method was much faster to implement, as I used similar code to the insert() method. However, since I didn't need to know the specific path to the last available value in this method, I started at the top instead of the bottom, where I simply compare the two children of the current value I am on while traversing the heap, and swap the greater one with the now "empty" slot where the maximum used to be. Once I hit the bottom, I made sure to finish moving the "empty" slot to the end by sliding any remaining values to the left in the heap until the "empty" slot is at the end, which was the result of ~10 minutes of debugging.

resize()

Since I chose to implement this heap using arrays, it needs to be resized anytime the array is full. It was fairly simple to program, as it only requires creating a new, larger array using the appropriate instance variables, and then transferring the old array into the new one, and assigning the new array as the new heap.