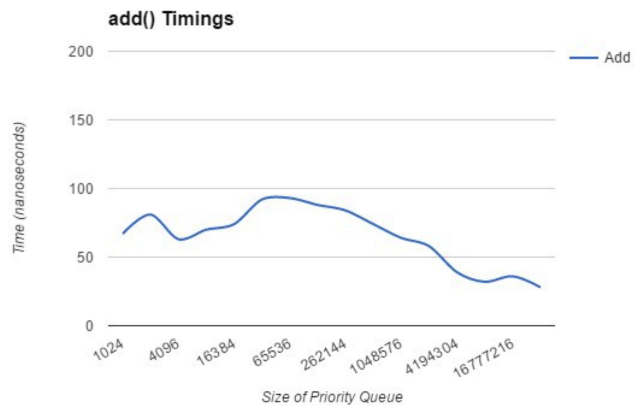
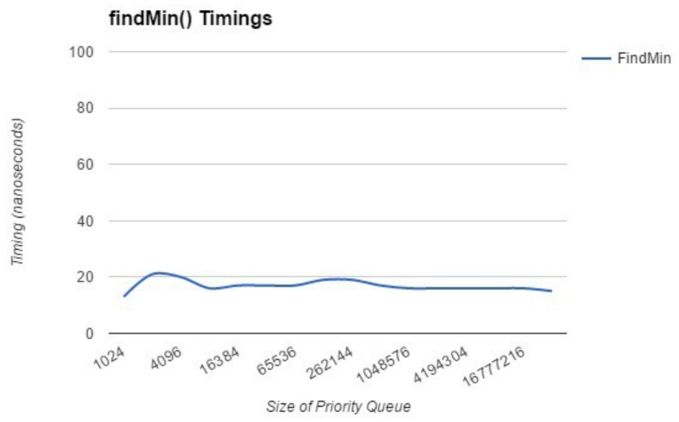
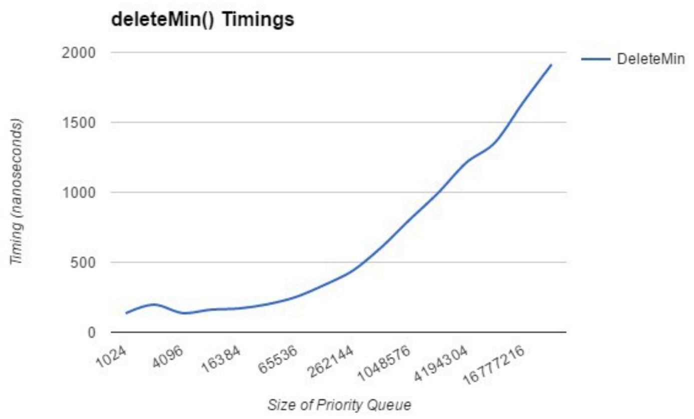
Joshua Shipley
CS2420
Assignment 11 Analysis

1. Design and conduct an experiment to assess the running-time efficiency of your priority queue. Carefully describe your experiment, so that anyone reading this document could replicate your results. Plot the results of your experiment. Since the organization of your plot(s) is not specified here, the labels and titles of your plots(s), as well as, your interpretation of the plots is critical.







- For this experiment, I created a PriorityQueue and filled it up with a large set of randomly generated integers of varying sizes. For each size, for accuracy, I performed a million repetitions of timings, with each rep timing how long it took to add a random integer, find the current min, and delete said min (each function was done and timed separately). I then plotted all of the average running-times, which is shown by the three graphs above. As you can see, even though add() had some weird variation to it, it never peaked above 100 or below 30 nanoseconds, which is such a small measurement, it is practically constant to human perception. My findMin() function was much more neat and consistent due to its straight forward nature of simply returning the first value. DeleteMin(), however, was took the longest of time, resulting in a logarithmic timing due to how it always required the new root to percolate down to an appropriate spot.

2. What is the cost of each priority queue operation (in Big-O notation)? Does your implementation perform as you expected? (Be sure to explain how you made these determinations.)

- add() - O(c) – As there is a random element to it, saying that add is a pure constant is a slight understatement due to the heap's relatively sorted nature. However, as the majority of elements are located within the lower levels, which is where we always initially add the new element, on average the random number doesn't affect much. My timing reflects this, as even though the time did have some variation, the overall trend was relatively flat-lined.

- findMin() - O(c) – Due to the Binary Heap's nature, the minimum number is always located at the "root" of the tree. Because of this, finding the minimum is has a constant nature and simply returns the first element in the collection. As this is what my implementation does as well, it performs quite fast and consistently.

- deleteMin() - O(log(N)) – This one takes longer than the others because when we delete the current minimum number, we replace it with the last of the tree's leaves, which is always one of the largest numbers currently held within. We then have to work it back down the majority of the tree until it reaches a point where everything extending from it is greater than it, thus maintaining the binary heap structure. This act is log(N) due to how we cut out half of the remaining elements to potentially compare to with each iteration down. My implementation does reflect this act rather well.

3. Briefly describe at least one important application for a priority queue. (You may consider a priority queue implemented using any of the three versions of a binary heap that we have studied: min, max, and min-max)

- When you have multiple assignments or reports to work on, it helps to be able to prioritize which ones to work on first based on the date they are needed and get everything done on time in an orderly fashion. Similarly, if you can come up with a form of rating system for difficulty or importance, you can use that to differentiate priority within the queue as well.

4. How many hours did you spend on this assignment?

- I spent about 5 – 6 hours on this assignment.