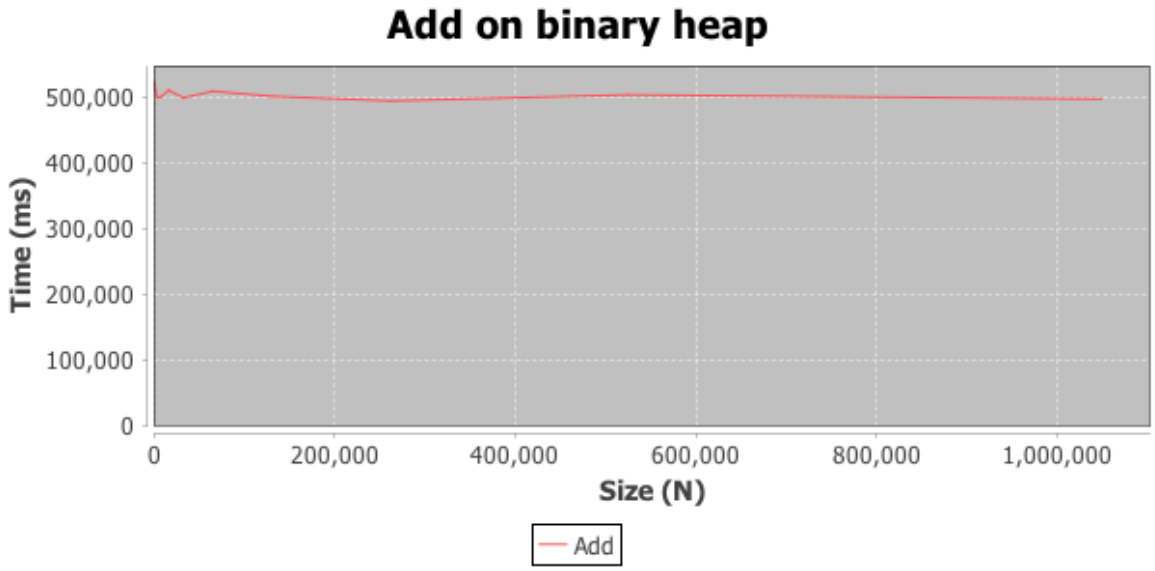
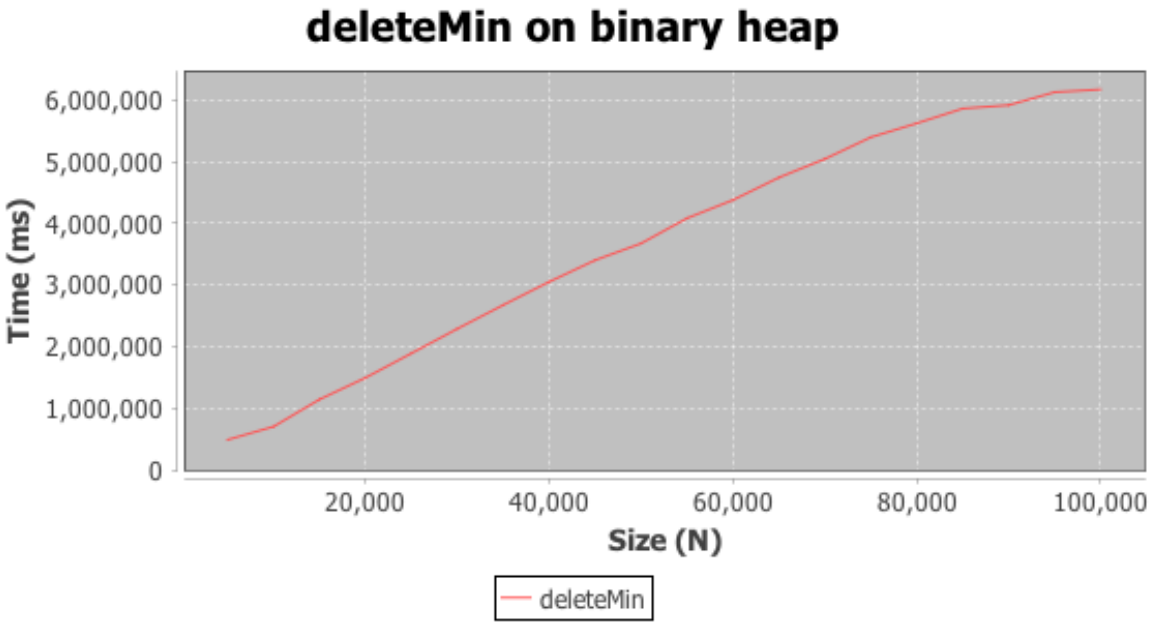
Kyle Price
U1018878
Assignment11
Analysis Document

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. I chose to test both add and deleteMin for this timing experiment.

For add, I added 10_000 items to the heap from 2^10-2^20 times. However, in between iterations, I didn't reset the heap. That means that through the test, the size of the heap that was being added to was increasing. According to the slides in class, even as N grows, the add time on a growing list should remain about constant. As you can see below, the graph displays a very constant add time as the size of the heap grew.



For remove, the number of items being removed was constantly growing. However, on this test, I did reset the heap each time as it was taking far too long to complete the tests. After all, removing does take longer than add. I figured the fact that I was increasing the number of items being removed each time would counteract that aspect. The time needed to remove a growing list of items from a binary heap is Log(N). This is because 75% of the items are on the bottom two rows. The graph below nicely displays a logarithmic growth pattern.



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.) findMin naturally performs in O(c). This is because the minimum value is always at array[0].

Add performs at O(c) as well. The reason for this is that items are added at the bottom of the list and then they have to percolate upwards until they find their correct position. 75% of the items in the heap are in the bottom two rows. This means that the huge majority of items don't have to travel very far. According to Dr. Meyers, on average, only 2.6 comparisons have to be made. This falls well within the bounds of O(c).

Remove performs in O(log(N)). The reason for this is precisely the opposite of add. Items are removed from the top. From there, their successor must percolate down to find their correct position. Again, as most of the items are at the bottom, the majority of items have to travel to the bottom two rows. Traversal from the top to the bottom of a complete tree is done in O(log(N)). Therefore, removal is done logarithmically.

All of these complexities matched the data I gathered from my timing tests.

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) The most important application I can think of is definitely Dijkstra's Algorithm. DA uses a min-heap priority queue like the one I just made. DA finds the cheapest path from one destination to another. I believe we will be implementing it in our final project to find the cheapest air routes across the country.

4. How many hours did you spend on this assignment? This was the least time consuming assignment thus far. I probably spent around 7-8 hours on it.