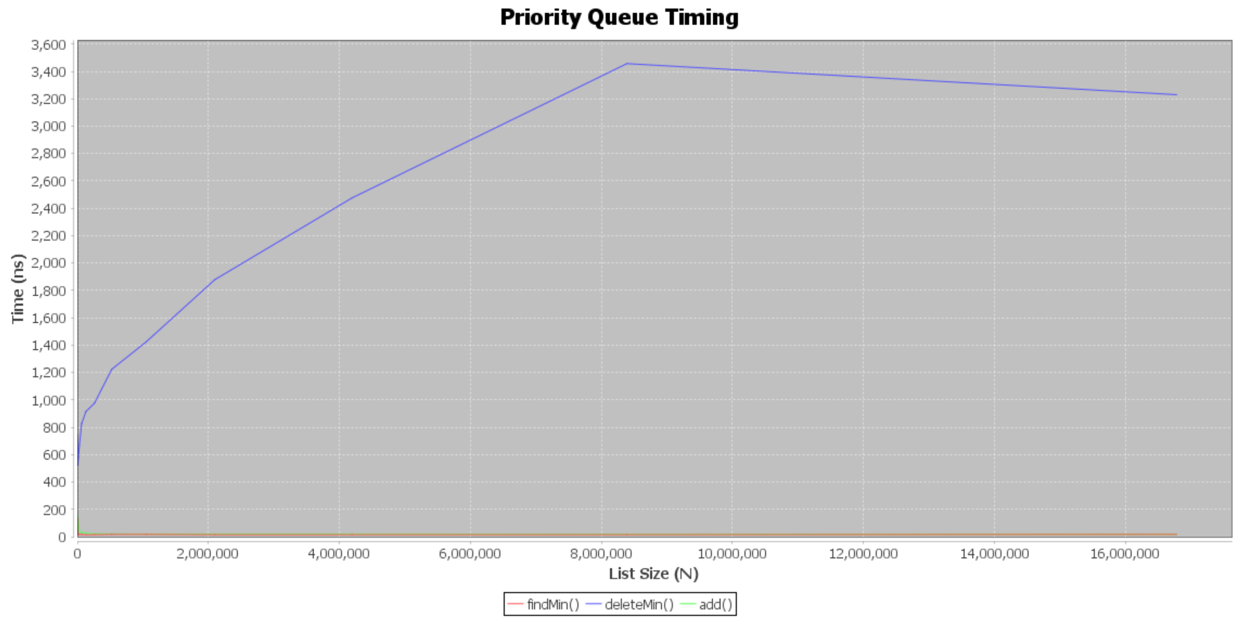
Braeden Barwick

1. 

   For each of my timing methods, I first filled a queue with N items. For the findMin() method, I just recorded the amount of time required to run the method on the priority queue. For the deleteMin() method, I recorded the time it took to delete an item from the queue, and then added another random string to the queue outside of the timing to preserve the queue size N. I did the same for the add() method, deleting an item after every add to maintain the queue size.

2. Both add() and findMin() are constant time operations O(c). This is fairly obvious for the findMin() method, as it just returns the item in array index 0, but it doesn't seem to fit for the add() method. The reason is because the running time of add depends on the relative ordering of the new item being added. The time will only be O(log(N)) if the new item is the new minimum of the set. Instead, because 75% of the items in the queue are in the bottom 2 rows of the tree, the add method only requires on average 1-3 swaps up the tree, making it run in a constant average time. O(c)

3. The first application for a priority queue using a heap I thought of was for an emergency room wait list. When a nurse diagnoses a new patient, she can add the patient to the queue with a relative urgency value. The higher urgency values would be placed further up on the queue, while lower urgency patients could wait a little bit longer. This is kind of similar to how wait lists for emergency rooms already go, but I don't know if they use a system similar to a priority queue.
4. I spent about 3 hours on this assignment.