Jiwon Nam (u0950753)

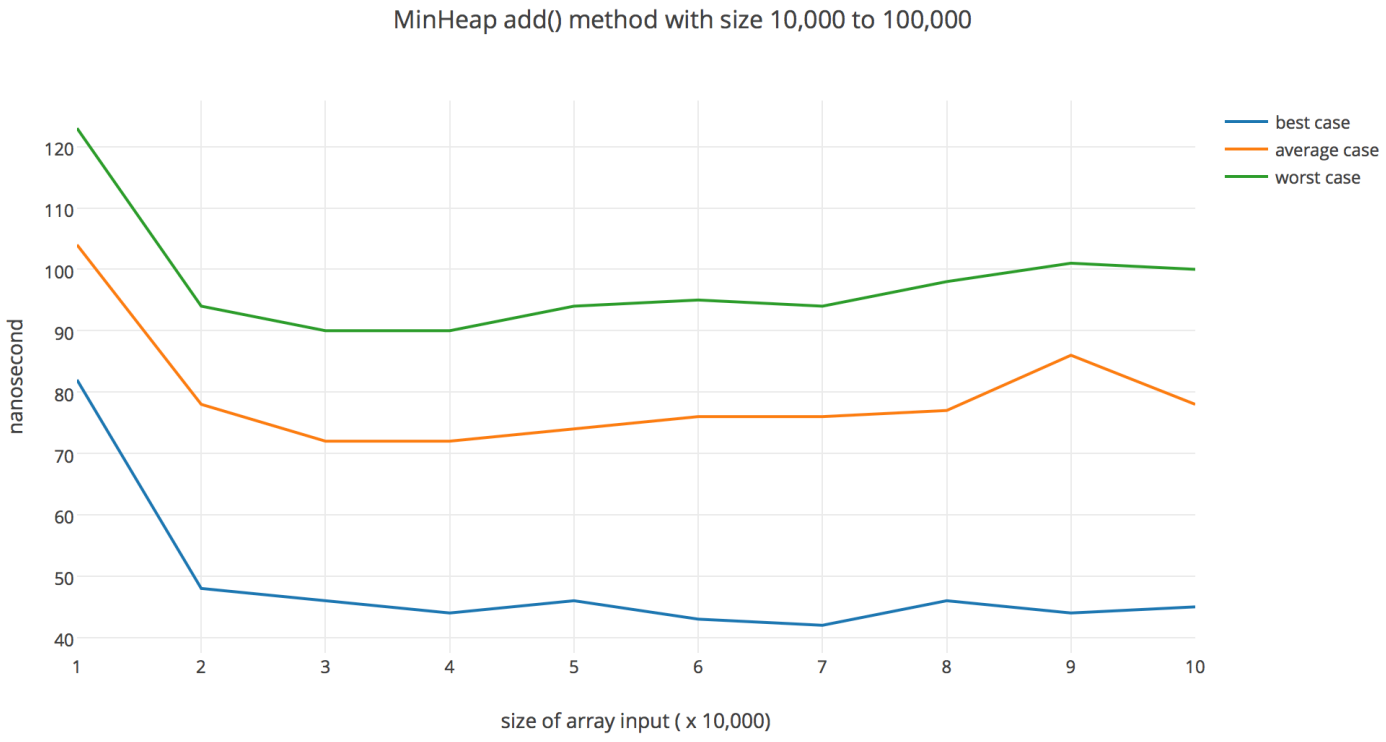
When you are satisfied that your program is correct, write a brief analysis document. The analysis document is 20% of your Assignment 11 grade. Ensure that your analysis document addresses the following.

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 experiment for PriorityQueue class, I tested for add method, findMin method, and deleteMin method. First, for add method, I set three cases which are worst case, average case, and best case. To simply make priority queue, I use integer object to build priority queue. The worst case is to store 1-N numbers by descending order like N-1 adding. The average case is to store random order numbers 1-N adding. The best case is ascending order of numbers 1-N adding.

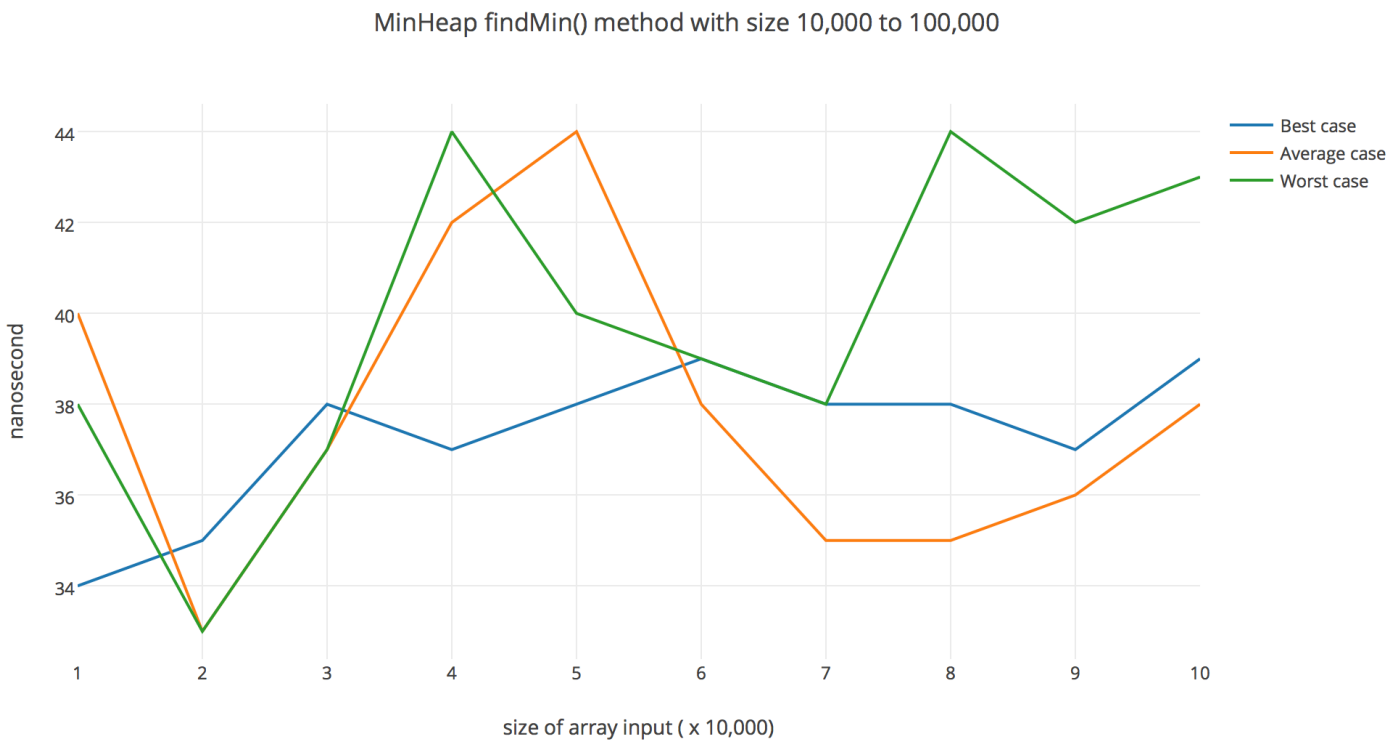
I set N to be 10,000 to 100,000 items input. I collect each run-time for all N items, and divided by the size N for getting average run-time for one single adding in the size. To make the experiment briefly, I set repeating loop for 100 times. From this experiment of add method, here is the result graph for all three cases.

From the graph, for worst case, the graph looks like O(logN) shape, but average case and best case look like O(c) graph shape. average case is slightly higher run-time, but not big difference between them. Therefore, I can conclude for my test as O(logN) for worst case, but O(c) for average case and best case.



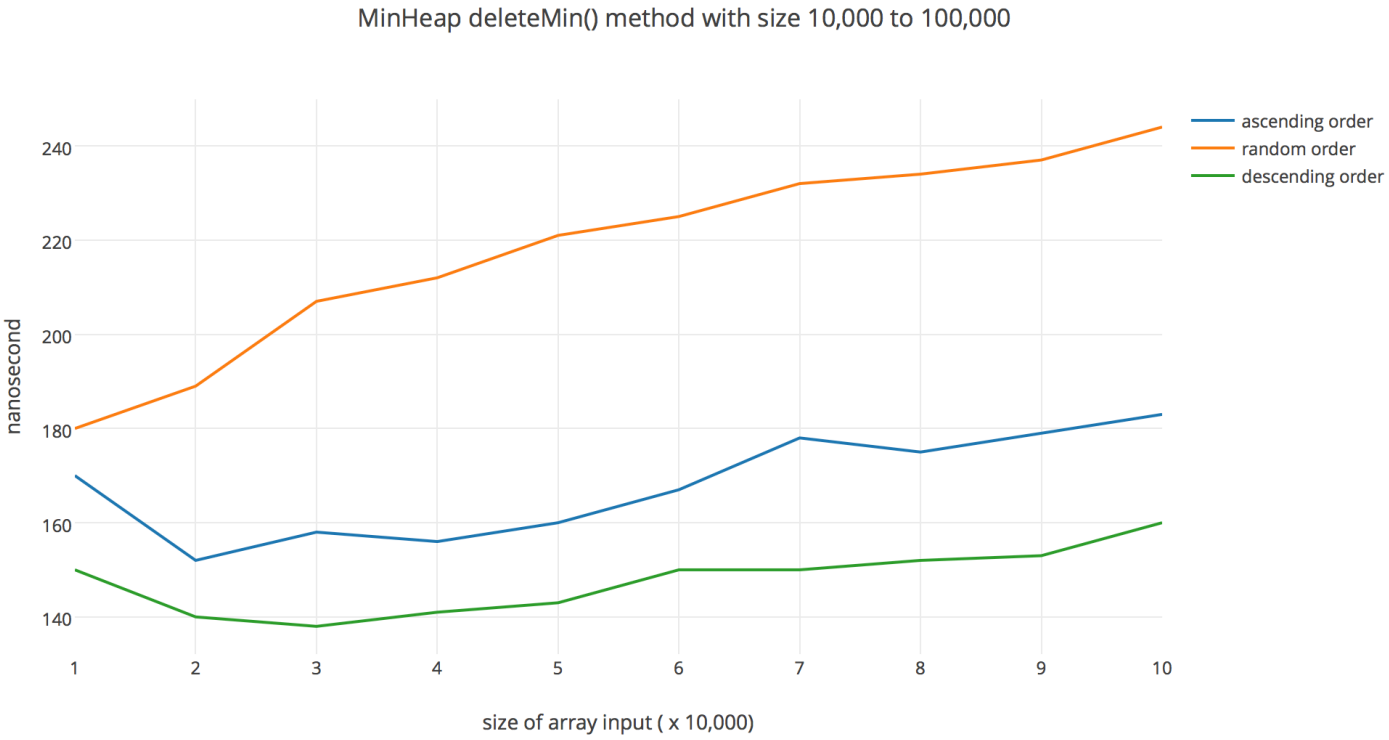
Second, for finding min method, I also set three cases, and already build heap with three cases sized by 10,000 to 100,000. In this case, I set repeated loop 100 times to make it briefly, and find the graph for three cases. here is the result of this experiment.

All three case shows same result graph. The graphs look like O(c) for every case, so I can conclude that the big-O complexity for findingMin method is O(c) for all cases.

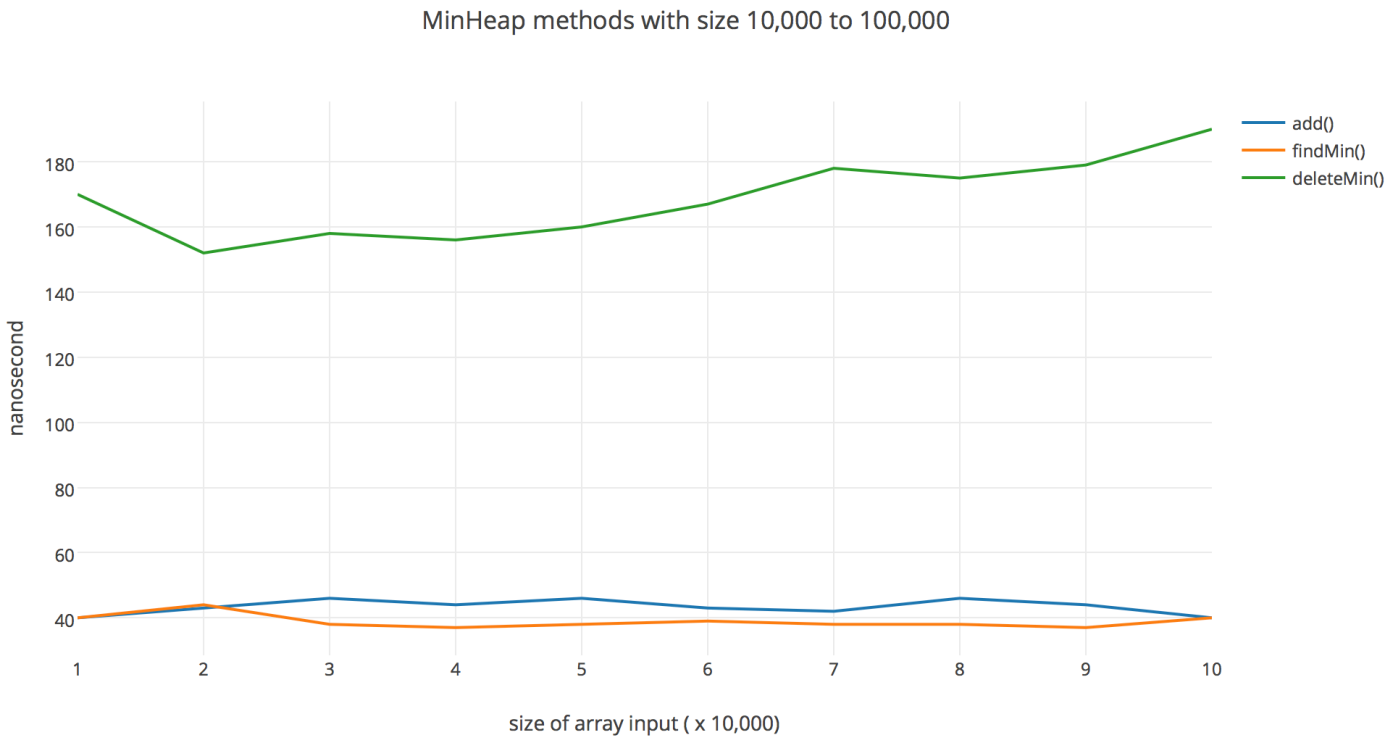


Finally, for delete min method, I already build three cases of heap by using add method strategy, but didn't get time for it. From existed heap with N size, I delete min item and get the time. To make it briefly, I set repeated loop as 1,000 for doing it again. Here is the result of this experiment.

From the graphs, all three cases have same result because the heap is slightly sorted set, so percolate down method hit similar result for every cases. Therefore, I can conclude that the Big-O complexity for deleteMin method, it is O(logN) for every cases.



comparing best case for all add, findMin, and deleteMin method result so that I can find Big-O complexity.



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.)

In my Priority Queue, I have add, findMin, and deleteMin methods.
First, in add method, I have strategy that add every input and store them in the given array by using min heap algorithm. The progress of algorithm, first, if the array is empty, just store the input at first index which is root and increase the size. Second, if I already have element in the array, I store input element at the last index, and then compare with its parent which is positioned (int)((index - 1) / 2). This process called percolate up, and I create helper method to percolate up with recursive strategy until it reach that it is bigger than. From this percolate up method, it used to take run-time less than log(N) and greater than O(c) because the usual case don't go through all the array to the root from last index using percolate up strategy. Therefore, we usually call this method Big-O complexity is O(c) for best case and average case. however, in the worst case, if I input smallest item every time, it should go through all parent in percolate up method; therefore, it takes O(logN) for the worst case. By my experiment, it is same result that I mentioned here.
Second, in findMin, it is always throw run time as O(c) because the minimum value is always located at first index of array by add method already min heap sorted the array. Therefore, it just return first element if it exists. It doesn't have any worst, average, and best cases because it just return first index of array. In my test also shows it always have O(c) for run-time at every cases.
The last method, delete Min. I need to find find minimum value first, but it is located at the first index of array, so run-time to find min element is O(c) at this point. And then, I swap the min element and the last index of array. This also occur run-time as O(c). Then, delete the last index (O(c)). Finally, I need to find another min value to store at first index so that it satisfy min heap. First, I need to compare root's children, and find smaller value. If I found smaller one, swap with first index. And then, the swapped child need to swap again with this same progress. This called percolate down, and I created this percolate down helper method to find correct position for the swapped element that is swapped with delete item. Until it reach both children are bigger than or no child, the percolate down method should be repeated by recursive strategy. However, usually, the last index of array element is smallest because of add method strategy. It implies that the run-time of percolate down occurs O(logN) for all cases, and from my experiment, it is same result that i mentioned here.

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)
From min heap, it is easy to find minimum value in the array, and it costs constant big-O complexity. Same strategy of max heap, it is easy to find maximum value in the array with contant Big-O complexity. For min-max heap also take constant Big-O complexity for getting maximum value or minimum value in the stored array. Also, it is less time to build sorted heap within O(logN) complexity because of its properties. For this reason, if I want to find biggest or smallest value to return with fast time, this heap is really efficient. Any other method, if I want to find smallest or biggest element, can not take less than this heap method with this adding strategy. Others need more time to add with sorting the algorithm, so heap strategy is really benefit for some of specific perporses.
4. How many hours did you spend on this assignment?
5 hours.
Programming partners are encouraged to collaborate on the answers to these questions. However, each partner must write and submit his/her own solutions.

Upload your solution (.pdf, only) here by 11:59pm on Wednesday, November 16.