CS 2420

Assignment 3

Jacob Bullard

Assignment 3 Analysis

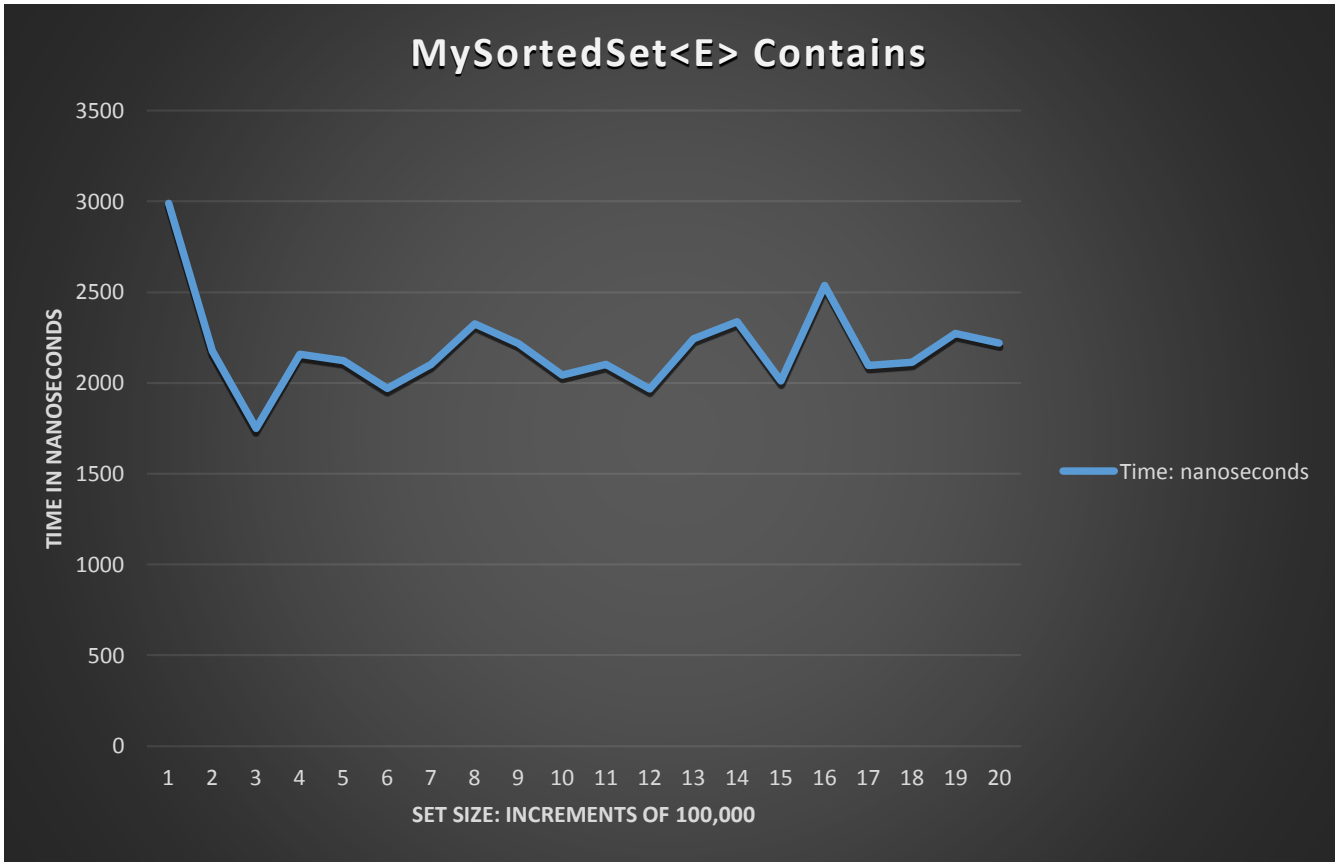
For assignment three, I worked with the same partner, Alfred Neufeld. We switched rolls when we started the timing experiments. I would have preferred to drive during the timing because I feel like I knew a more efficient way to do the timing. Freddy was once again an awesome partner to work with. For the next assignment I will be working with someone new just so I get the opportunity to change things up. I might work with Freddy again on future assignments.

If we were to have backed our sorted set with a Java List instead I feel that implementation would have been trivial. In order for the set to remain sorted, after every add() or remove(), our list would have to be sent to the Collections sort method. I'm not up-to-par with which sorting implementation collections uses, but even in the best possible case the sort would be on the order of O(NLgN). I feel that the efficiency would be determined by the way in which Javas Collection does sorting and how Java's List does contains. After doing further research I've found out that Java's List contains() method is on the order of O(N). The order of add() and remove() would be O(N^2LgN) and the contains() would be O(N) if we were to have used a Java List. I feel like the way we implemented our sorted set by using an array provides a better efficiency than using a Java List as our backing structure. The mentioned remarks are strictly related to run-time efficiency. As far as program development time is concerned, using a Java List would have taken give or take five minutes to implement. Thank goodness for knowledge of data structures and algorithms!

The expected running time for MySortedSet's contains() method is O(LgN). This is because we implemented a binary search algorithm in our contains() method. I am not as familiar with logarithms

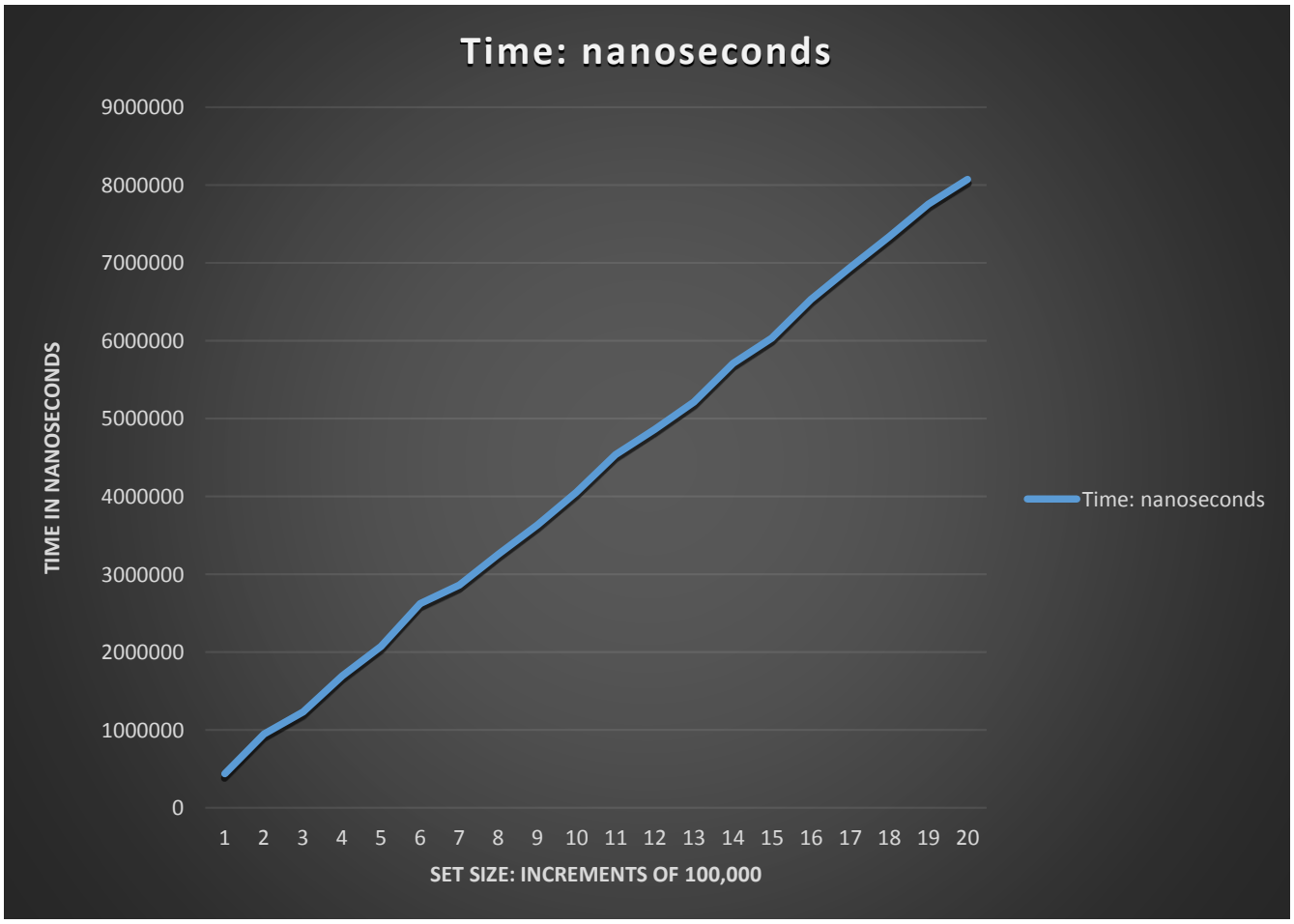
as I should be, but according to the Big-O cheat sheet online, a binary search (if implemented correctly) is on the order of O(LgN). I feel that we implemented our binary search correctly. N here is referring to the size of the set when contains is called..

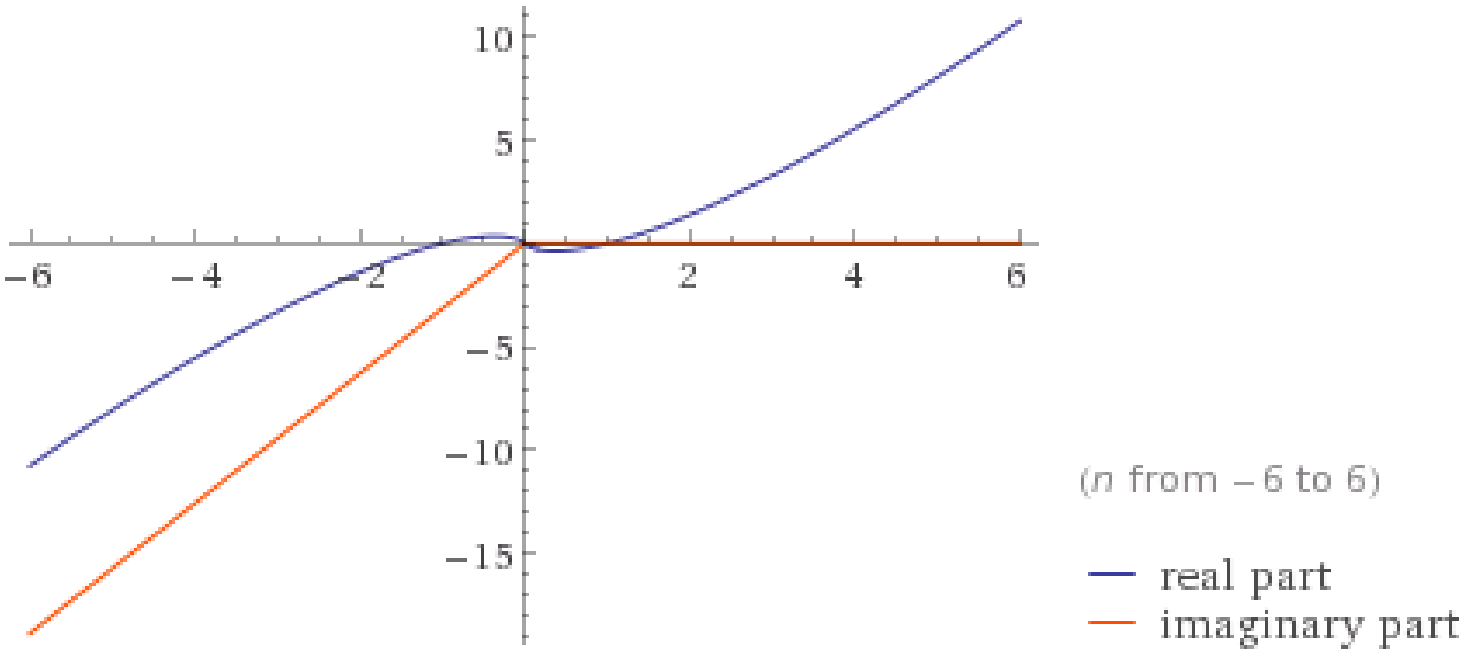
The following portion of analysis will include a plot of the running times for the MySortedSet's contain() method for sets of sizes 100,000 to 2,000,000 by steps of 100,000. Timing techniques which were demonstrated in Lab1 were used. The number of times we tested each set was 1,000,000. Using this large of a number for times to test each set provided for more clear data. The growth rate of the graph looks as expected. There is very slow growth which matches perfectly to logarithmic performance. The bumps in the graph are most likely caused from miscellaneous processes running in the background.



The following portion of analysis will include a plot of the running times for the MySortedSet's add() method for an element not already contained in the set. By observation of this graph and taking

into account the running time of contains(), which is called during add(), we can deduce a running time for add() by itself, as well as contains(). In order to get accurate results, we filled different instances of our MySortedSet with integers from sizes 100,000 to 2,000,000 in steps of 100,000. After filling each set, we removed an element at random that was previously contained in the set, then proceeded to add the element back into the set, only timing the single add. Our graph for the single call to add appears linear. However, if we take into account that contains() is being called during add(), we can note that add() is on the order of O(NLgN). The 'actual' add by itself without the call to contains() would be linear, but we must take into account that for our set to remain a set, we must call contains() to ensure no duplicates. N here is referring to the size of the set at the time of the add.





(Produced on Wolfram Alpha)

My partner and I spent about ten or so hours on this assignment.