Matthew Christensen (u0899251)

Assignment 3 Analysis

My programming partner is Patrick McHugh (u0883718). I was the one who submitted the source code for our program.

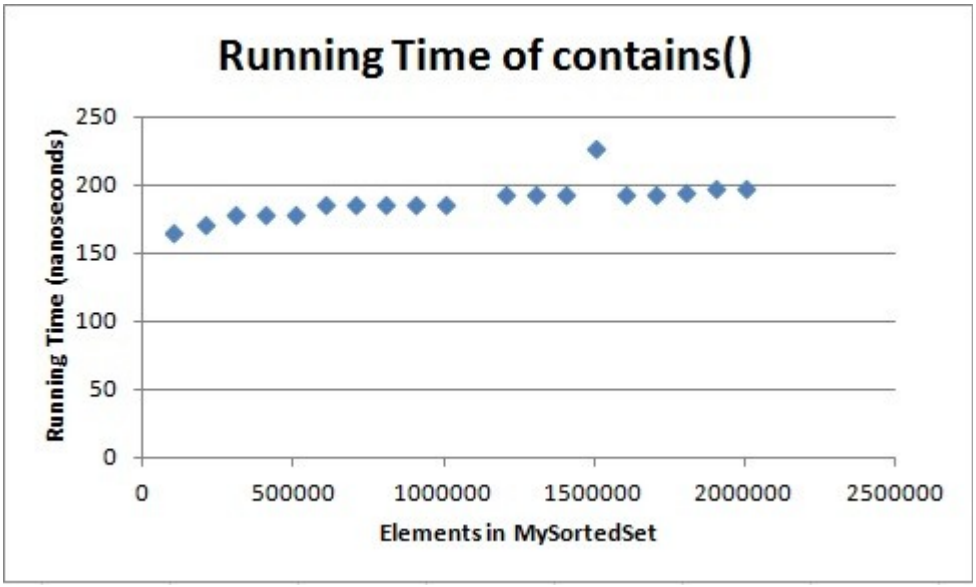
Originally, my partner and I switched roles every 30 minutes or so. Eventually we started switching rolls far less often. We became more comfortable switching every hour. I'd say we were pretty comfortable with switching every hour since that gave the Driver plenty of time to complete one big idea or so before retiring to the Navigator spot. Any longer and the Driver would have begun to lose some of their "drive."

Again, Patrick was very easy to work with in terms of scheduling and work ethic. We both decided that it would be best to begin coding for this assignment early rather than later and are very glad that we did. The assignment turned out to be much more difficult than we originally thought. Through it all, Patrick has been able to contribute during his time serving in both roles for our extended coding sessions. When the time came, he even showed the initiative to comment the code for an hour straight. I can easily see myself working with him on future projects since we seem to work so well together.

The implementation of the MySortedSet class itself wouldn't have been wildly different if we had used a List Many of the methods that we were implementing already had counterparts in the ArrayList class that we would only have to modify slightly. The main points are that the add() method wouldn't need to check whether or not the set had gotten too big for the backing data structure, and that we could probably use the ArrayList's binarySort algorithm instead of having to create our own (so development would have been sped up pretty significantly). In terms of runing time, I don't think that using an existing List would really speed things up at all, since those implementations still have to check the same conditions that our class did. Perhaps their methods are a bit more optimized, but beyond that

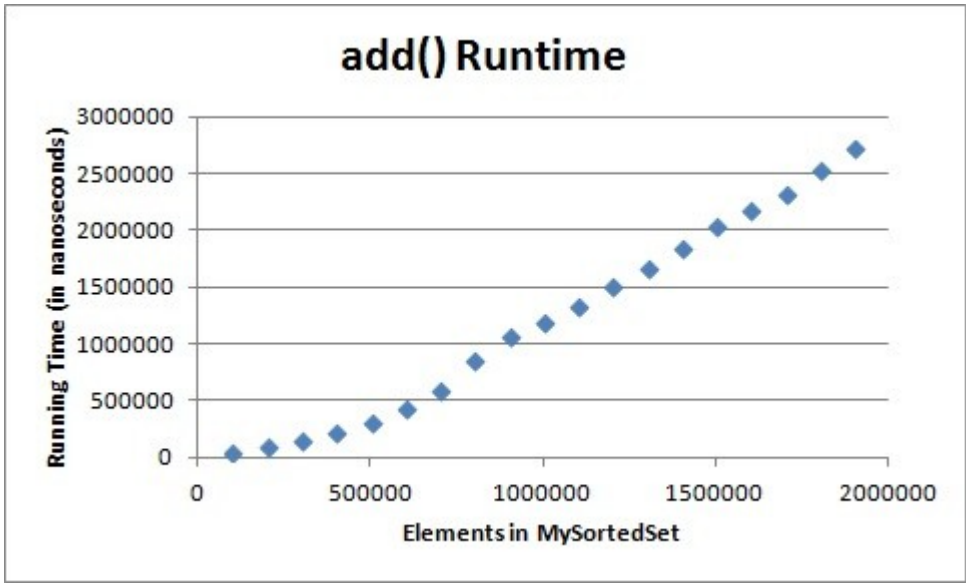
there isn't much that would change.

I expect the behavior over time to look like O(Log(N)), since doubling the number of elements will only increment the running time by one comparison. Since all of the elements are sorted, this is a fairly easy and efficient way of checking for specific elements.



Here is a chart documenting the average running times of the contains method for different amounts of items in MySortedSet. Notice that the curve does appear to be logarithmic (with the exception of the outlier towards the end). This matches my expectations for the behavior of the method.

Since the add method needs to run the contains method, the location of the spot to add an element takes the same time that it would for contains: O(Log(N)) . The more time consuming part of the code is where it needs to take all of the elements after the newly added element and shift them one space over to the right. In the worst case, when the element is at the very beginning of the list, every element needs to be relocated. In this case, the complexity of the operation would be O(N), since the amount of time to perform it is basically proportional to the amount of elements in the set.



This is supported by our test data. For the tests, we timed how long it took to add a value to the beginning of the list (averaged over 1000 tests). The graph displays a linear relationship between the number of elements. Out of interest, we also tried to test adding elements to the end of the list, and saw a distribution virtually identical to the results of the contains() method.

When all was said and done, we spent about 8 hours total on the actual coding (which includes the 2 hours we spent commenting). The most difficult part was figuring out how to think generically enough. Most of the logic was (relatively) straightforward, and testing went pretty smoothly.