Safia Hassan

1. Who is your programming partner? Which of you submitted the source code of your program?

Christina Foreman. Christina submitted the source code.

2. How often did you and your programming partner switch roles? Would you have preferred to switch less/more often? Why or why not?

My partner and I would switch roles about every 10-15min. I would like to switch more often maybe about every 10min so both of us are actively engaged the whole time.

3. Evaluate your programming partner. Do you plan to work with this person again?

Christina is a very committed individual. She shows up 10-15min early for all of our meetings. She is also willing to schedule last minute meetings if we need more time. I do plan on working with Christina again.

4. If you had backed the sorted set with a Java List instead of a basic array, summarize the main points in which your implementation would have differed. Do you expect that using a Java List would have more or less efficient and why? (Consider efficiency both in running time and in program development time.)

If we had backed the sorted set with a Java List then we would be extending the Collection interface which would help abstract the notion of a position. The interface, java.util would also add numerous methods to the Collection interface. Backing the set with a list would provide greater functionality. In terms of efficiency, a basic array is optimal when dealing with a fixed length as well as quick sorting algorithms. However, if the size of the set is not known, or is very large it is more efficient to use a Java List to keep program development time small and the overall code simple.
Resource: Weiss, Mark Allen. Data Structures & Problem Solving Using Java, fourth edition.

5. What do you expect the Big-O behavior of MySortedSet's contains method to be and why?

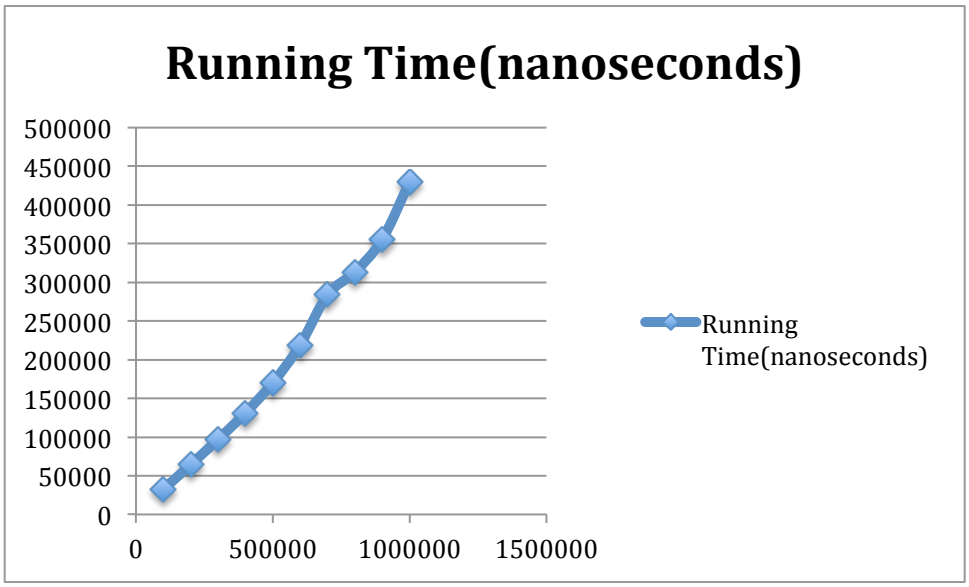
The Big-O behavior of MySortedSet's contains method is O(*n*). This is because for any array, in order to search for an element the *for* loop runs i=0+1+2+…+(n-1) times. As the dominant term in this summation is *n*, the Big-O behavior of this method is O(*n*).

****Unfortunately, my partner and I didn't read the specs carefully and we did not implement a binary search in our contains method. So instead of O(nlogn), our sequential searching algorithm has a complexity of O(n).*

Safia Hassan

6. Plot the running time of MySortedSet's contains method for sets of sizes 100000 to 2000000 by steps of 100000. Use the timing techniques demonstrated in Lab 1. Be sure to choose a large enough value of timesToLoop to get a reasonable average of running times. Include your plot in your analysis document. Does the growth rate of these running times match the Big-oh behavior you predicted in question 5?

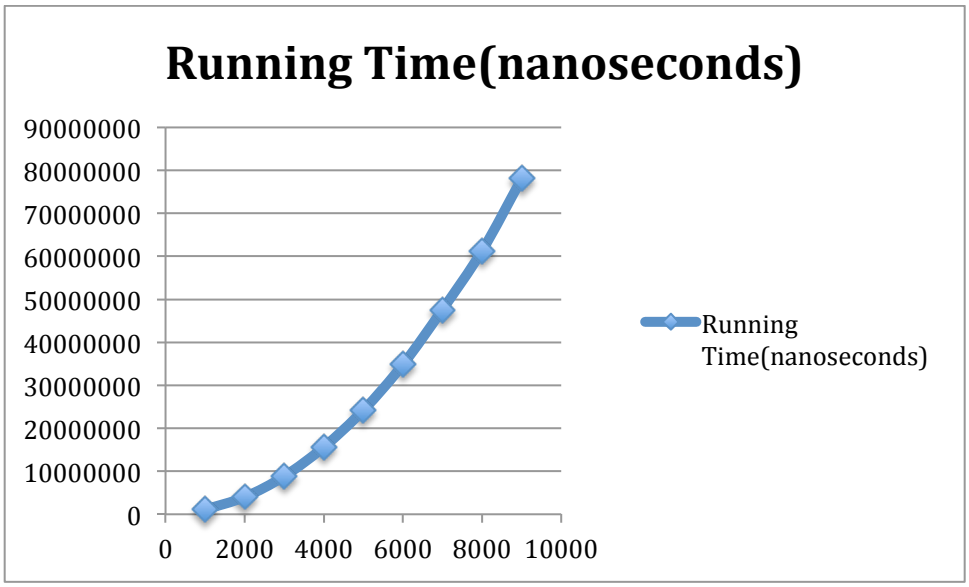
So due to the problem of not implementing a binary search, plotting the running times for such a large set for a sequential search was taking a long time(~1 hour to plot first 10 steps). I plotted the first 10 out of the 20 steps and the growth of that matches up with the Big-oh behavior predicted in question 5.



Safia Hassan

7. Consider your add method. For an element not already contained in the set, how long does it take to locate the correct position at which to insert the element? Create a plot of running times. Pay close attention to the problem size for which you are collecting running times. Beware that if you simply add N items, the size of the sorted set is always changing. A good strategy is to fill a sorted set with N items and time how long it takes to add one additional item. To do this repeatedly (i.e., timesToLoop), remove the item and add it again, being careful not to include the time required to call remove() in your total. In the worst-case, how much time does it take to locate the position to add an element (give your answer using Big-oh)?

The add method is of O(n), so to add an element not already contained in the set will have a linear growth as n increases. In order to plot the running times for the add method, I first timed n calls to add so that I could take into account the scaled growth rate. This is because the add method constantly changes the size of the set and as n increases. The timing for n calls to the add method followed a quadratic growth which implies that the add method on its own must have been a linear growth since we made n calls to add. So thus, the worst-case for adding an element is o(n).



8. How many hours did you spend on this assignment?

Approximately 11.5 hours.