Daxton Wilson
U0264580

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

Name: Ryan Cantera
UID: u0855101

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

We switched roles every 30 minutes or so. I think this worked well because it allowed us to get into a rhythm and then get a big picture view when needed.

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

My partner was incredibly helpful and complimented my strengths and weaknesses. I plan on working with him in the future.

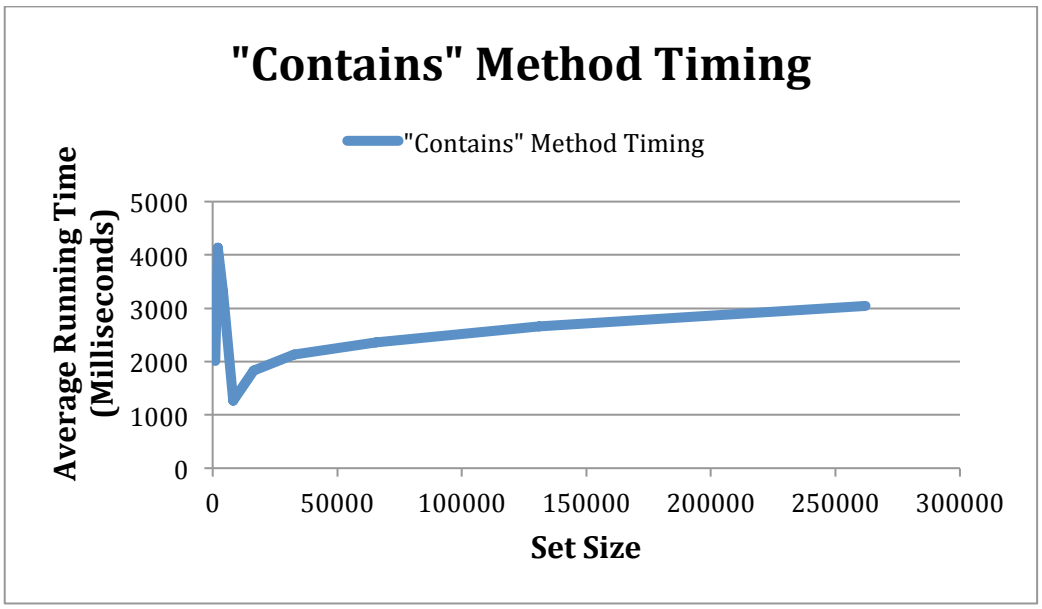
4. If you had backed the sorted set with a Java List instead of a basic array, summarize the main points in which your implementation would have differed. Do you expect that using a Java List would have more or less efficient and why? (Consider efficiency both in running time and in program development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

Programming time for Lists is significantly lower than that of arrays for the purposes of this assignment. A Java List would have made several methods much easier to implement because of getter, setter, contains, and adds methods, which already exist in Java Lists. Lists also allow for the input and change of data much easier because they are not restricted to a specific size. As far as speed efficiency, I do not think the difference would be significant, because the real determiner for our speed is the different algorithms we used and not the data structure itself. Once the code was implemented, a List is preferred because it is an easier data structure to manipulate.

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

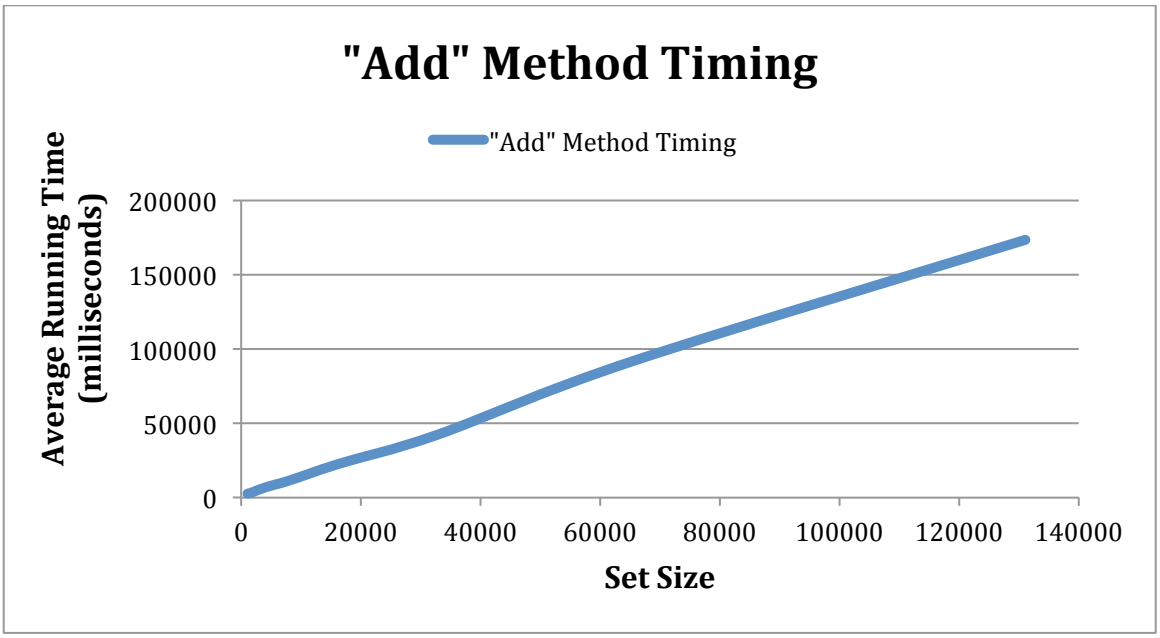
BinarySearchSet's contains method should be O(logN) because the method halves the number of elements left to check each time it searches for a new index. The number of times contains loops is dependent on N only for how many times it must guess a new index to find a specific value.

6. Plot the running time of BinarySearchSet's contains method, using the timing techniques demonstrated in Lab 2. Be sure to use a decent iteration count 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?



Yes, the growth rate matches the expected Big-oh behavior predicted in question 5. Outliers at low set sizes can be ignored due to the unreasonably small set size the "contains" method was timed at. (The running time to calculate the time was incredibly long and is likely due to the inefficiency of our methods.)

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., iteration count), 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)?



For our program, the worst-case timing for the "add" method is O(NlogN). This is because there is both a binary search element O(logN) and a for loop through the entire array once an element is added O(N) in order to shift every element in the array after the element being added. If you were to add an element at the 0 index, the entire array must be looped through: the worst-case. Best case would be O(logN) when the element is added at the very end of the array.

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

17 hours.