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

Rachel Brough, I 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?

We each spent about half the time in driver and navigator, I was fine with that rotation.

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

I work very well with her and she does as much as me, I worked with her last week and will be working with her again next week.

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.)

In terms of program development time I would say it would be a lot easier, as an add, remove, sort method are already available for use. In terms of efficiency though I cannot say, it would depend on how the list is coded and whether calling on the set method, then having to call on the lists method would have a noticeable impact on the time it takes to run the program.

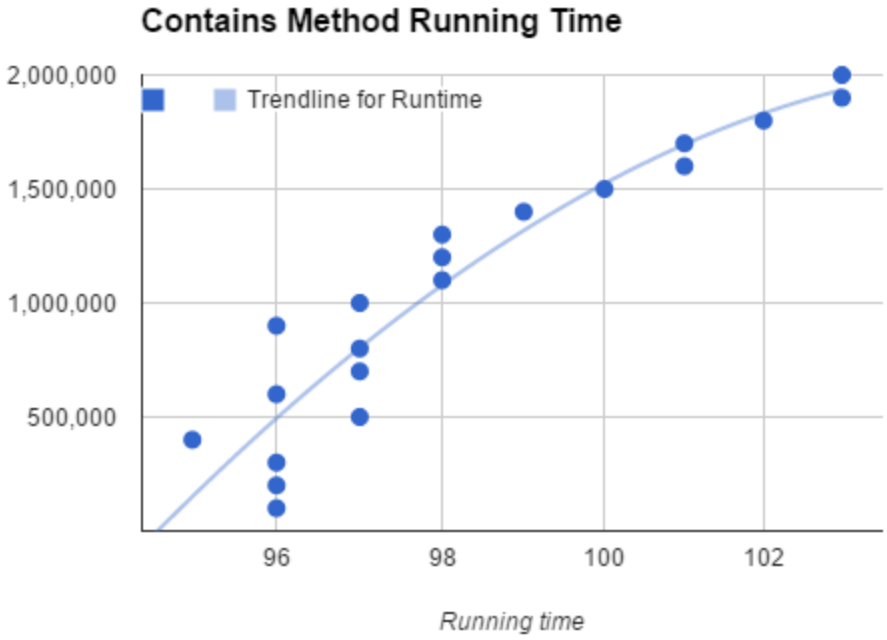
5. What do you expect the Big-O behavior of MySortedSet's contains method to be and why?

Contains runs off a binary search so I would speculate it to be an O(logN) complexity.

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?

I ran the experiment the five times per increment and took the average. I searched for the value 50,000 in an array of integers from 1 to the incremented value. Table lies below.

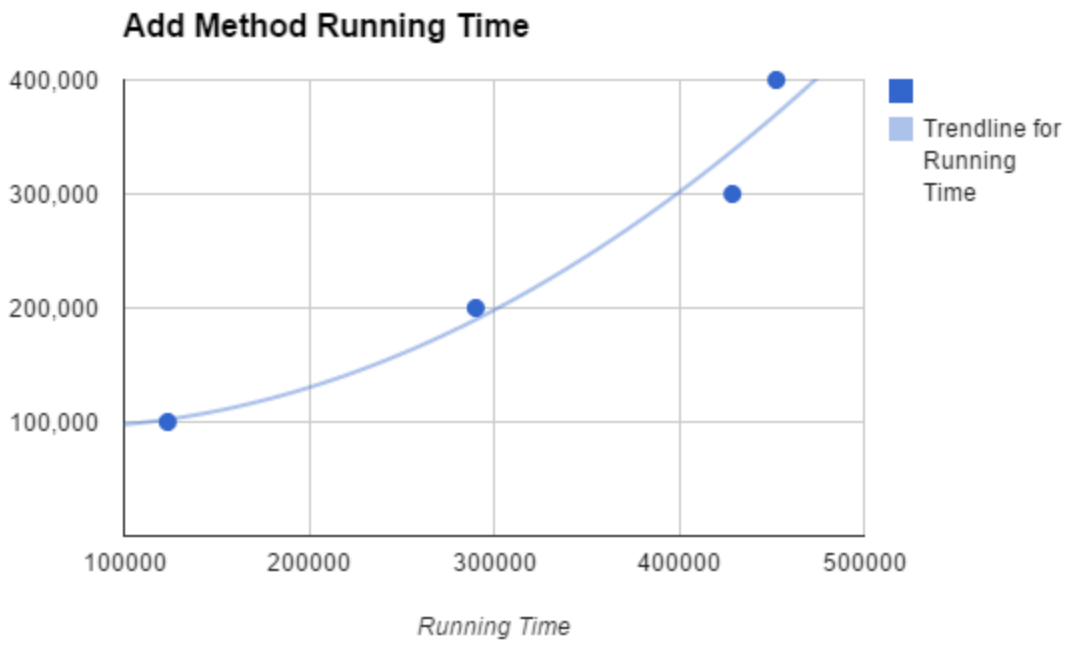
Time in Nanoseconds	Number of Items Searched
96.0	100,000
96.0	200,000
96.0	300,000
95.0	400,000
97.0	500,000
96.0	600,000
97.0	700,000
97.0	800,000
96.0	900,000
97.0	1,000,000
98.0	1,100,000
98.0	1,200,000
98.0	1,300,000
99.0	1,400,000
100.0	1,500,000
101.0	1,600,000
101.0	1,700,000
102.0	1,800,000
103.0	1,900,000
103.0	2,000,000



The graph above represents the points plotted out with a trendline drawn through. The growth seems to follow a O(LogN) complexity, what I predicted in Question 5.

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)?

In theory it should take the amount of time as the contains() method since both run the same binary search. In actuality because a for loop is run after finding the index, I assume the equation becomes O(N log N).



The Worst Case would be O(NlogN)

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

7-8