Mackenzie A. Simper
CS 2420
Analysis for Assignment 3

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

My programming partner is Chase Wilker. I submitted the source code for this assignment.

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

I did the majority of the programming since I think I really understood the assignment more than Chase did. It was fine with me to do most of the work, but it would've been nice to have a partner that could shoulder more of the burden.

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

I felt that Chase really did not understand the concepts such as binary search, generics, and iterator, so I had to explain it to him before we could get any work done. Even then, I did most of the programming. I know Chase means well, but I don't think he dedicated as much time to this assignment as he wanted to/should have. I'm not sure if I will work with him 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 set with a List instead of an array, we would not have had to worry about the size of the array. In my constructor, I initialized my array to have a size of 10. If the array ran out of size, I created a new array with double the size and had to copy everything over, which took time! Also, a list has the add(int index, E element) method that would allow me to add an element in the correct index without having to manually shift everything over, as I did with an array. Because of these things, I think Java List would be more efficient in running time (because the program would run less for-loops) and development time (because I could write less for-loops!).

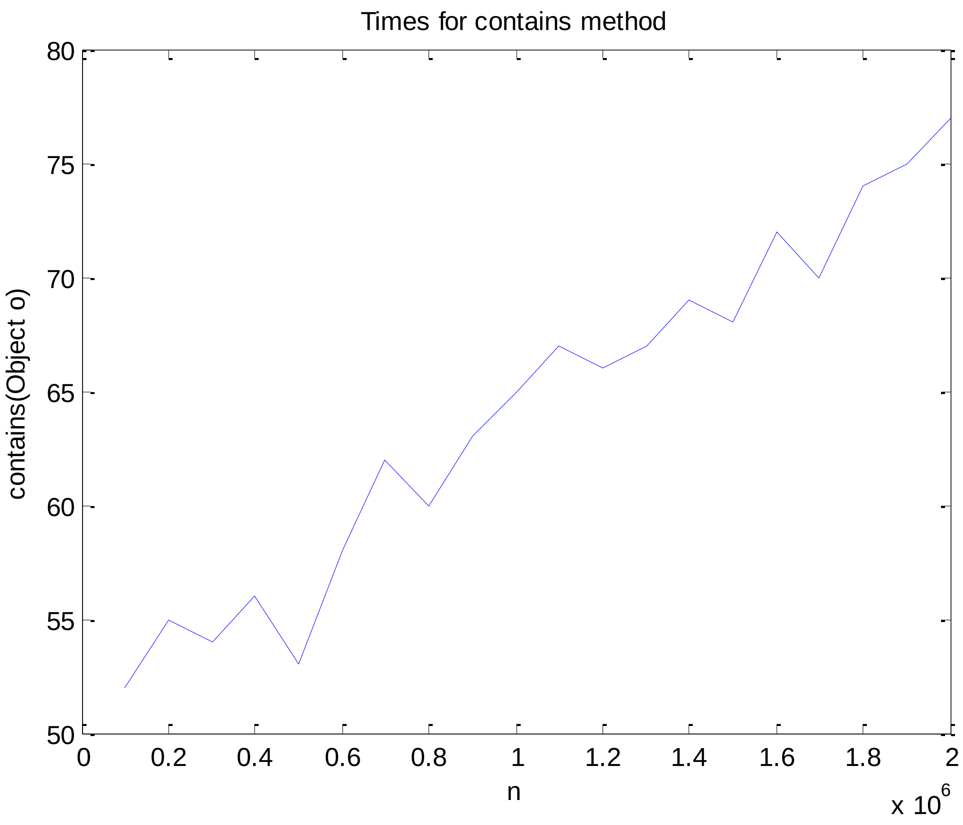
5. What do you expect the Big-O behavior of MySortedSet's contains method to be and why?

Because I used a binary search method for the contains method, I would expect it to behave like O(log n). As we talked about in class, with a binary search you just cut the array to be searched in half each time. This halving process leads to powers of two, and thus log based two.

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

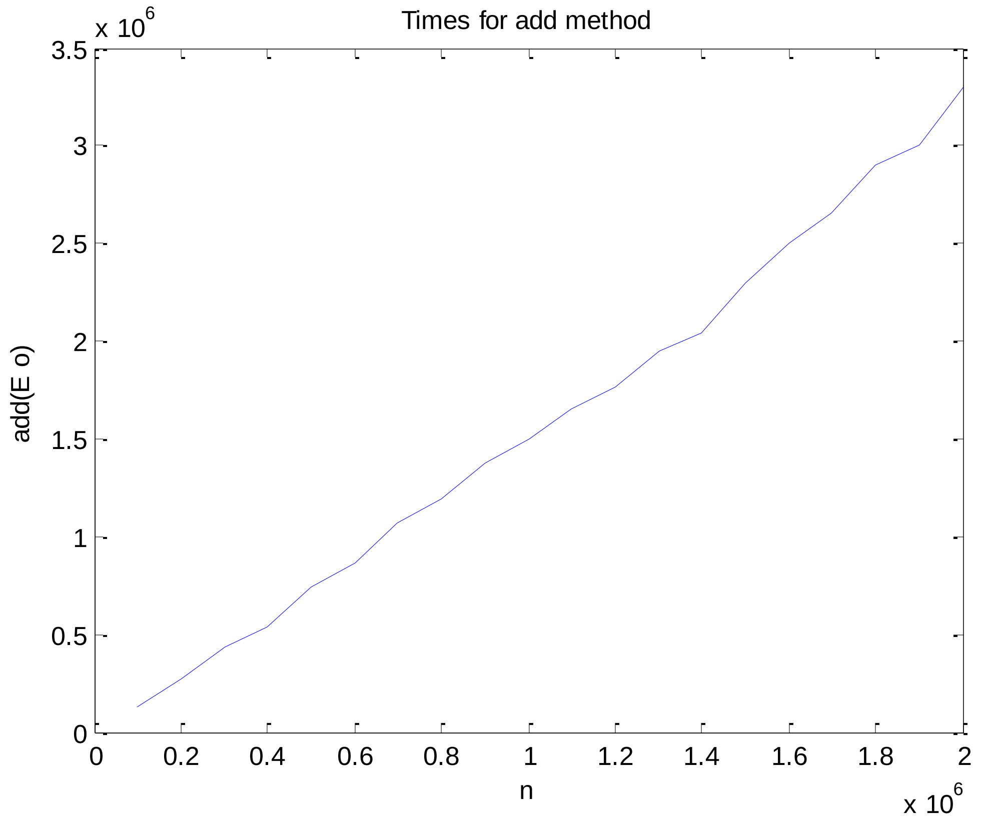
document. Does the growth rate of these running times match the Big-oh behavior you predicted in question 5?

The growth rate on the graph appears to follow roughly the same curve as log(n). The graph is a little jagged, but I think the main point is that as n increases substantially, the time it takes to call contains doesn't increase that much. Note that the x-axis is marked in 20000 increments.



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)?

I would hypothesize that that the Big-O behavior of the add method is O(n). Since we have to run a binary search to find the position to insert the object, and then everything is shifted over and the object is inserted, I would think the time would be log(n) + n, and so the dominant term is n. The worst-case scenario would be if the object was positioned at the very beginning of the list, since not only would the binary search have to run all the way through, but every single element also has to be shifted over. The graph seems to indicate this is true:



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

Approximately 10-12 hours.