Braden Davis
CS 2420
Spring 2015 – Assignment 3 Analysis Doc

1. Who is your programming partner? Which of you submitted the source code of your program?
My programming partner was Eli Dalton. I uploaded 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?
We switched roles a few times - about once per day that we worked on this assignment. We did all of the typing on my computer as he is still working on getting Java 7 up and running on his computer.

3. Evaluate your programming partner. Do you plan to work with this person again?
I like working with Eli. He's very knowledgeable about the material and wants to make sure that I fully understand the code and concepts that we're working on. I'd like to work with him again, but given our schedule conflicts. I'm not sure it would be a good matchup.

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.)
I expect the efficiency to be roughly the same, but a Java List would be slightly more efficient. Lists are more flexible and don't have to be grown after a certain number of items whereas arrays must be told to grow and spend the time and resources copying the information over from the old to the new array.

5. What do you expect the Big-O behavior of MySortedSet's contains method to be and why?
I expect the behavior to be O(logN) as we're using a binary search to determine if the collections contains a given item. Worst case scenario I expect it to be O(logN), and best case scenario I expect it to be O(1). It would be O(1) if the item we were looking for was in the middle slot of the array, thus the first thing that is seen.

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?
At this time, we're still working on the timing and I'll upload this as an update to the assignment if the time hasn't passed. I am assuming that since the worst and average case scenarios are both O(logN) and the best case scenario is O(1) that the timing will be slightly skewed below O(logN). It would make sense that the actual numbers would reflect that as well, given the environmental variables on my computer during run time.

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 Big O it takes roughly O(logN) time to search through and add an element to the array. The binary search is fairly consistent as far as the timing goes. Depending on where we're adding in the element it will take slightly longer, other times slightly shorter.

8. How many hours did you spend on this assignment?
We spent roughly 14 hours on this assignment and didn't finish. Our biggest problem was understanding Comparators and how to properly implement them into the code. We understood the main points and what they were supposed to do, but implementing them was not easy. Since the entire assignment hinged on that, we hit a large impasse for quite a while until we were able to muddle through it.