Michael Morris
Assignment 3 Analysis Document

1. My programming partner is Paulmartin Gentile. I submitted the source code for our program.

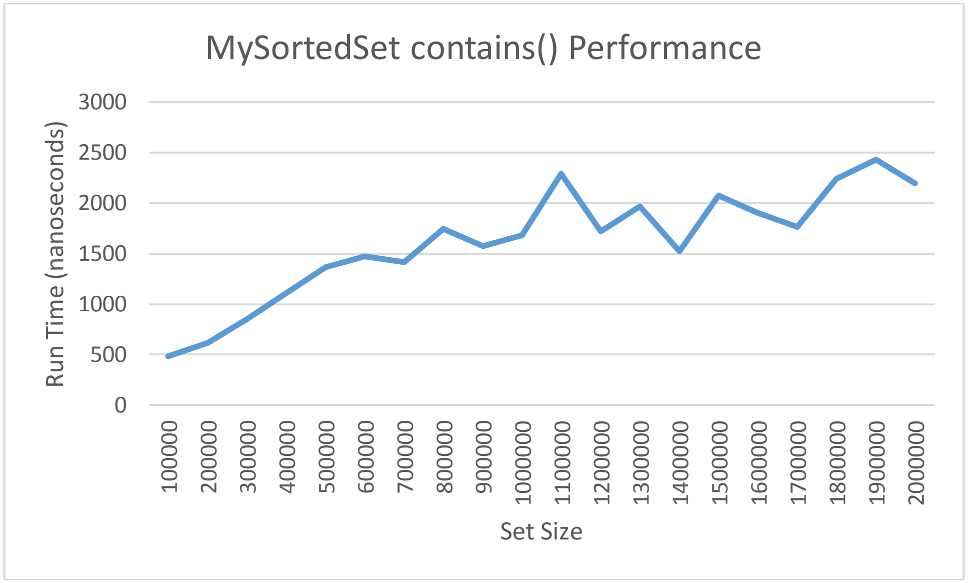
2. My partner and I switched roles frequently. While one of us typed, the other would have an idea and take over. This worked well. I generally like piloting so I did not mind switching frequently because it was not long before I got control again. I work best when typing and physically manipulating the problem, but it was also good to just think for periods and not worry about producing some result.

3. I get along well with my partner. This is the second assignment we have worked on, so we already are comfortable with one another and the expectations each of us have. We both think quickly and have similar design tendencies. I do plan on working with this partner again.

4. A Java List already expands as necessary, but my functions for preventing duplicates and null elements would remain the same. A List has contains, remove, and toArray methods, so that would simplify the program and make my implementations unnecessary. Because I can call the List's contains method, I probably would not need to keep my List sorted (which I would do the same way I do for the array), but creating a sorted Iterator would be more difficult. Overall, therefore, the List would not be much different from my array and probably behaves similarly to what my array and its functions do, so I doubt the running time would differ much (I would keep the Listed sorted). In terms of program development time, a few bits of housekeeping code would be unnecessary, so development would probably be slightly faster.

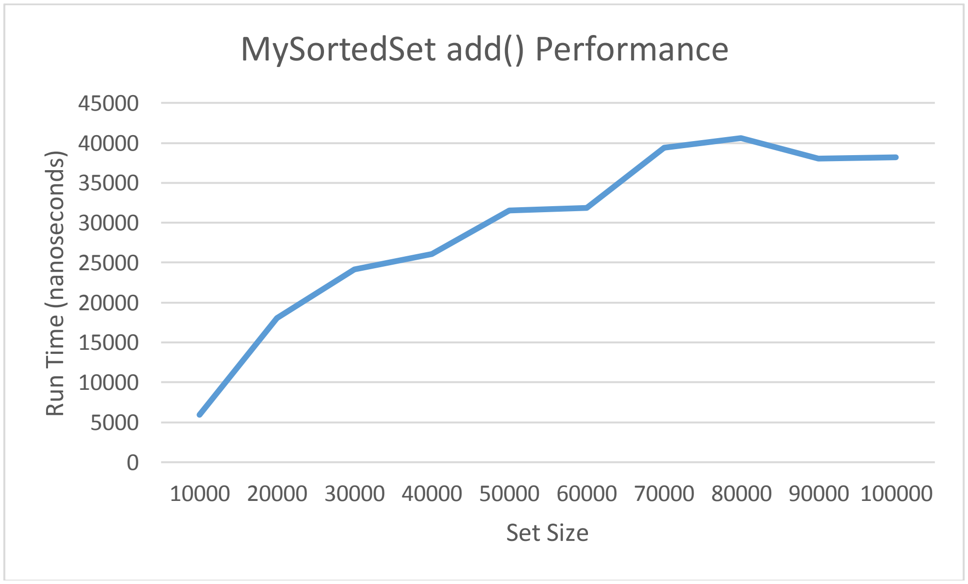
5. MySortedSet's contains method is binary search for the parameter object. If the object is not of type E or is null, that would be discovered right away. Otherwise, the search continues until the object is found or is not found. Worst case scenario (object is never found), the search does one straightforward comparison and halves the array appropriately, repeating as necessary. I anticipate the Big-O behavior would be similar to O(LogN), because if N is doubled, only one additional halving operation is necessary.

6.



Here is my analysis plot. Note that the operation was performed 100,000 times at each interval. The growth rate appears to match O(LogN). These tests began to take several minutes (15+ toward the end) to complete because of my sample size and clearly external performance variables became significant during the long processing times. I tried to control as much as possible, but I presume variable background services, etc. came into play. Ultimately, however, a 20x increase in set size produced approximately a 5x increase in operation run time.

7.



Here is my analysis plot. My add method appears to run at approximately O(N + LogN), or in the absolute worst case O(3N + logN), so about O(N). A binary search finds the target index, and then all the data starting at the index moves one to the right. Worst case would be the index is

0 and the array needs to be doubled during the move. Note, the operation was performed 20,000 times at each interval. Again, despite control, performance was variable even at each interval (repeated test consistently decreased in time, probably for a variety of background CPU, power management, etc. reasons).

8. I spent approximately 9 hours on this assignment.