Jen Simons
u0139542
CS2420
Assignment 3
Analysis Document

1. Jon Worms is my partner for assignment 3. Jon is submitting our source code.

2. We did not switch roles as often this time although we talked a lot about different ways of implementing our methods. Often Jon had a more straight forward or cleaner way to code a method than I did (he has more experience coding) and therefore he drove more often. I may want to switch a little more often so I have more practice of actually writing the code and not only talking through it.

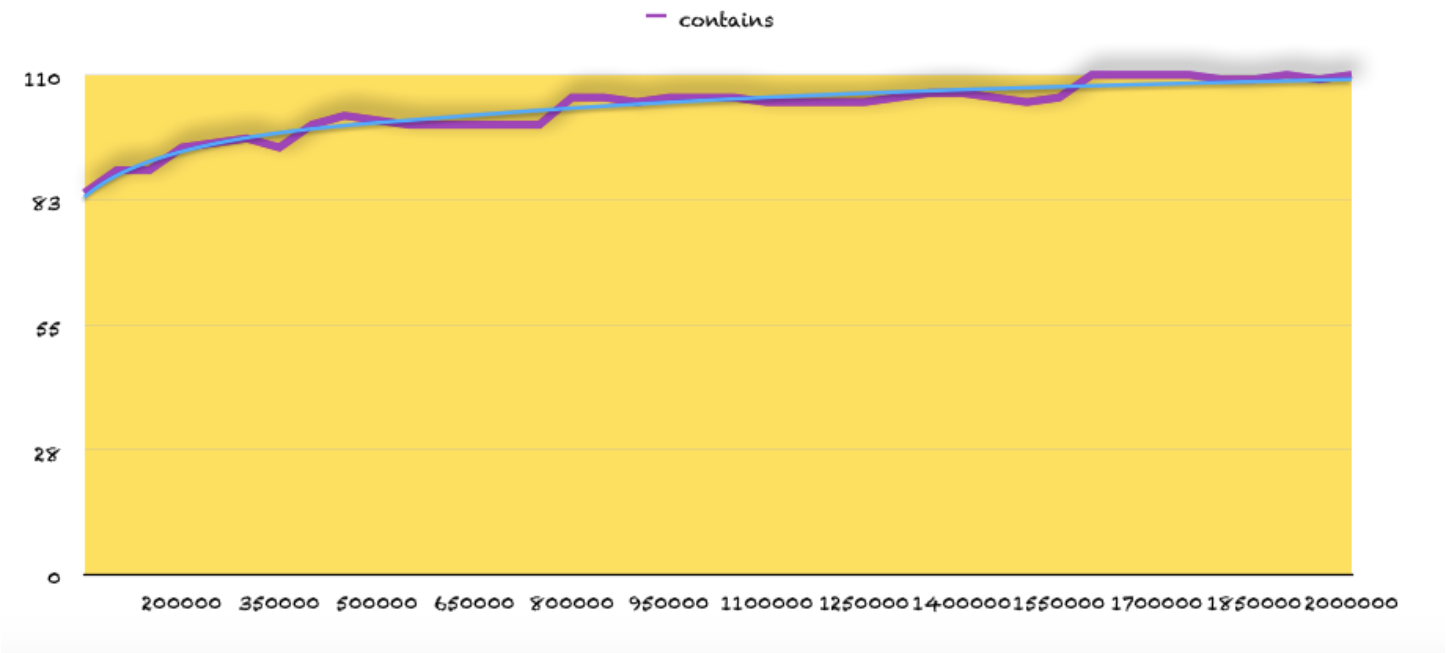
3. I do plan to work with Jon again. I feel that we communicate well and I feel that I'm learning not only from the assignments but from his previous experience, too. I appreciate that we are both focused on our assignments and highly motivated to do well.

4. If we had backed the sorted set with a Java List instead of a basic array we wouldn't have needed to account for growing the list as we did with the array since the list would do that for us. I believe a list already has it's own iterator, which means we wouldn't have had to implement those iterator methods. Also, list.size() returns the elements in the list and does not count empty/null places in the list. Array.length returns all the places in the array whether they hold values or are empty/null. Because of this we needed to keep track of the elements in our array and implement our own size() method.

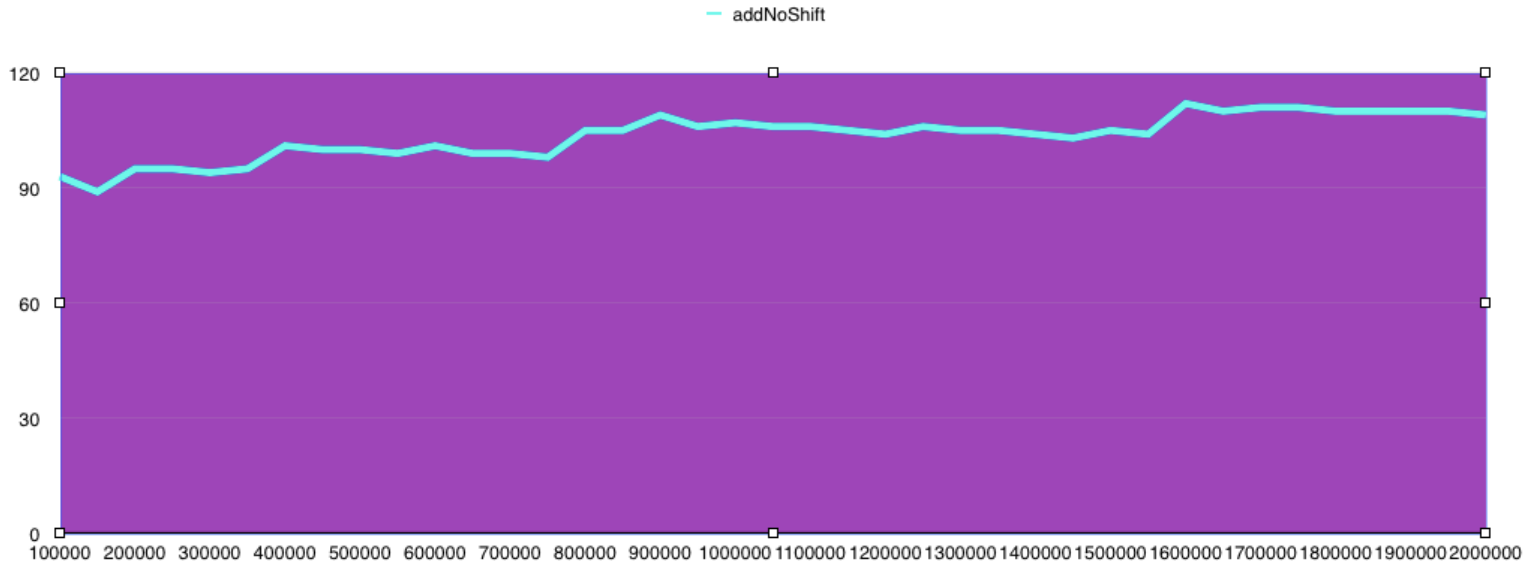
I believe using a Java list might have been more effective in program development time due to the things I mentioned in the previous paragraph. However, I'm unsure if running time would have more efficient or similar as I know that arrayList can be slower than arrays but I'm unsure how much of that slowness is related to the growing of the array.

5. I expected the Big-O behavior of MySortedSet's contains method to be O(logN). The search algorithm we used was a binary search and then we did a compare to check if the index returned by the search equaled the item we were looking for. A binary search is O(logN) and the compare is a constant, I believe. Therefore, I think the dominate term is O(logN) for our contains method.

6. Yes, the growth rate shown by our tests and the plot matches the behavior I predicted previously. The dark purple line is the plot of our timing test data, the code for which is in a JUnit test MySortedSetBenchmarks. The light blue line is a trend line for O(logN). Our timing test data for contains follows the curve of a logarithm.

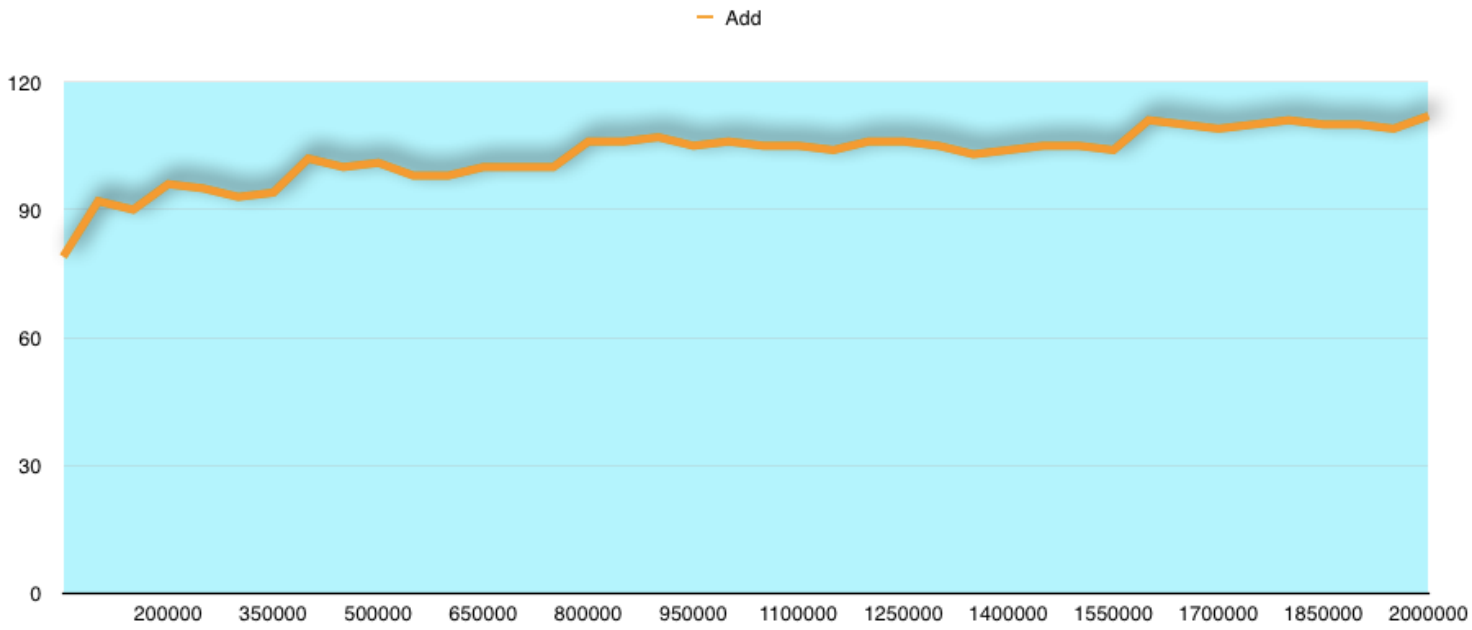


7. Our search is binary and is O(logN). Our add method uses this search to find if/ where an item is in the sortedSet. The search method returns an index of where the item to be added is or where it should be, therefore the time it takes to locate the correct position at which to insert an element not already contained in the set is the same time it takes to locate the item if it's already contained in our set, which is O(logN).



The light blue line in the above image (addNoShift) shows the times of our add method with no shift in values (i.e. the element is already in the set and is therefore not added.)

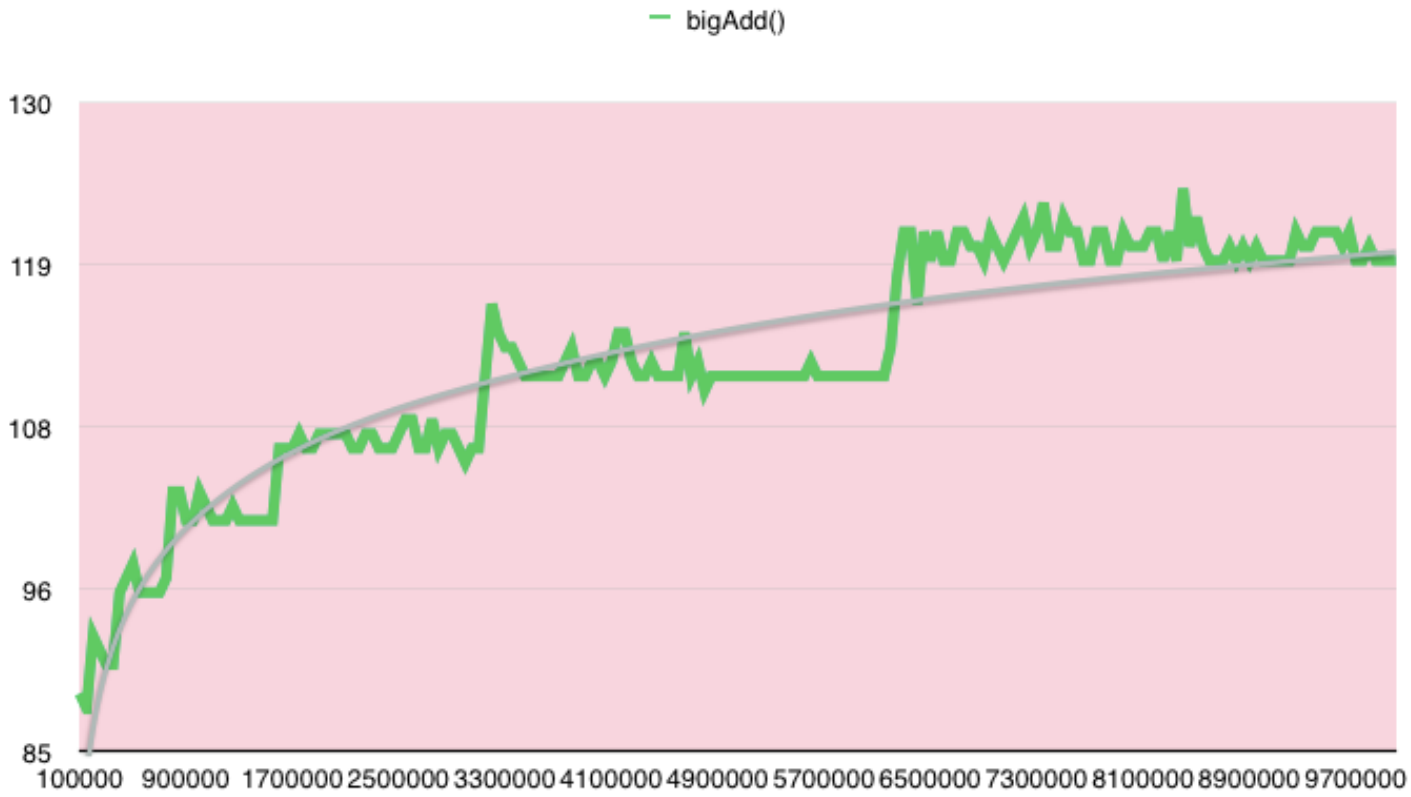
Once the search method returns the index where the item to add should be, our method compares the item at that index to the item to be added to make sure we do not add a duplicate. The compare is a constant complexity. If the item to add is not a duplicate, our method then determines whether to add the item at that index, or right after, shifts all the elements larger than the item to add to the right, then adds the item. I thought the shift of elements was O(N) because of shifts accordingly to the number of elements.



The orange line in the above image (Add) shows our timing test data for adding an element worst case scenario every time (i.e. the item must be added to the smallest index, causing all the elements in our set to be shifted.)

My estimate was that our add method was something like O(logN) + N, making it complexity O(N). This could make sense, looking at the above data images. However, it was unclear to me looking at this data if we truly had a linear complexity because the data lines could pass for O(logN).

So, we did a bigger test…



This image of our timing test of add (bigAdd) looks very much like a logarithmic graph. While I originally thought our add method was linear, looking at this information I would have to say it is O(logN). I need more practice at determining the complexity!

8. We spent around 17 hours on this assignment.