Jon Worms
CS2420
Assignment 3

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

My partner is Jen Simons. I uploaded 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 did not switch roles very often. I can type faster than she can and am more familiar with c-based language syntax. However, Jen will never get to be as fast as I am if she is navigator the whole time. So she most definitely needs more keyboard time. In the future I'm going to try encourage this more.

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

Jen is great, she is smart and very focused. She takes the work seriously and does not mess around. (All of which are qualities that I appreciate very much.) I will work with her until it is time to rotate partners.

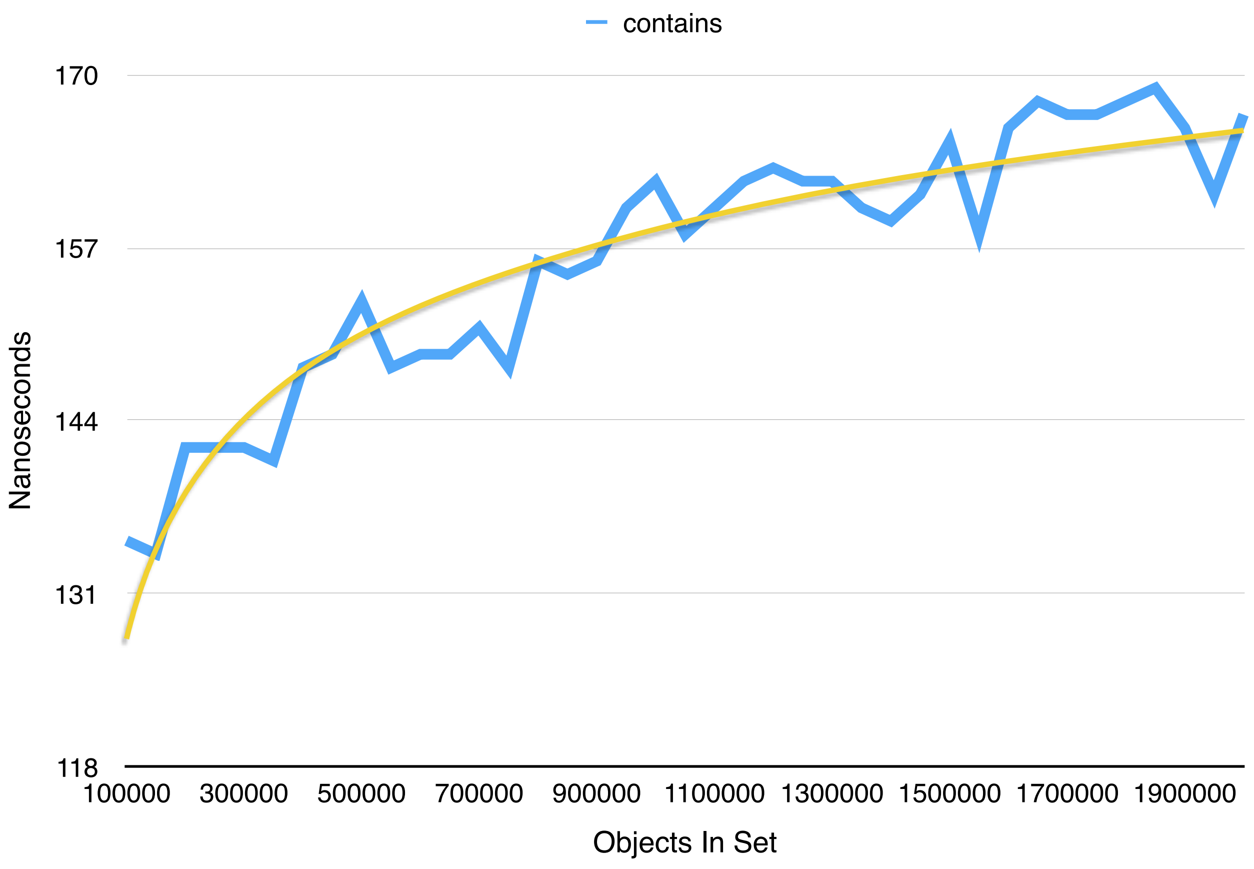
4. If you had backed the sorted set with a Java List instead of a basic array, summarize the main points in which your implementation would have differed. Do you expect that using a Java List would have more or less efficient and why? (Consider efficiency both in running time and in program development time.)

Lists have mutable capacity. So I wouldn't have to worry about growing the array. They also implement the collection interface, that would take care of any of the iterator methods I would have needed. (Though I'm not sure about the comparator) Keeping the list sorted would have been more difficult. Without direct access to each piece of data, we would have to work through the List classes methods. That would probably add a little time to the whole thing. Doing a binary search would also be a little challenging. However, that may be offset by better coding practices on Oracles side. (They wrote ArrayList right?)

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

Our contains method implements a binary search. So it halves the data with every pass through the loop. So that would be O[Log(N)] if I'm not mistaken.

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?

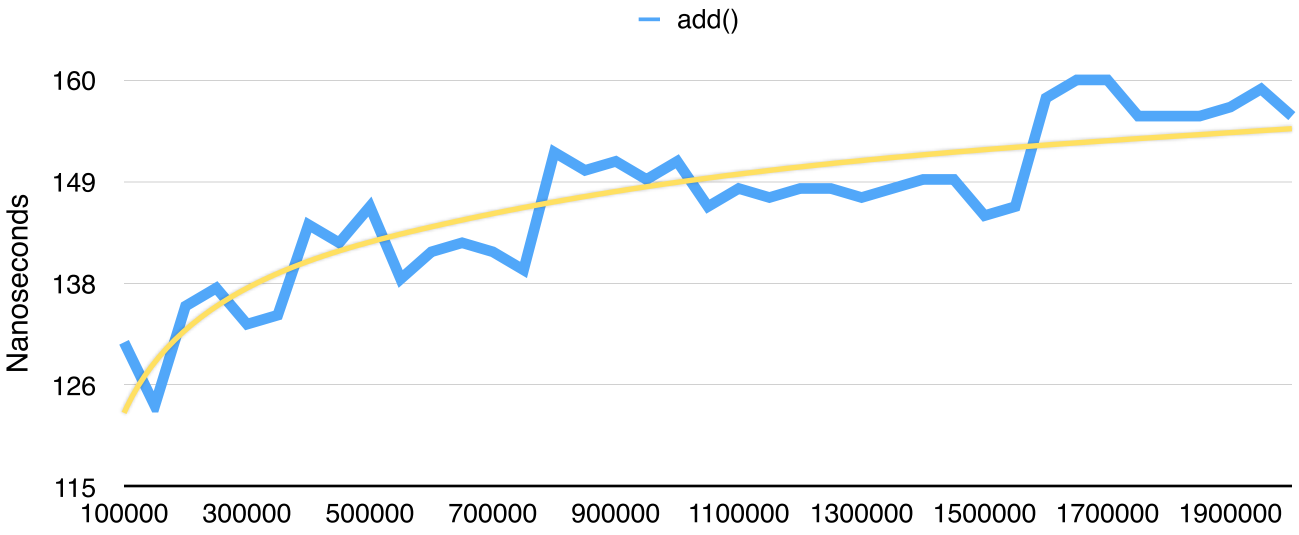


The gold colored trend line is logarithmic. We actually increased the resolution of the test by 2. (Stepping up total the data size in the set by 50,000 instead of 100,000) And we took an average of 20,000,000 samples per step. We tested all the way to 2,000,000 (though I can't seem to get it to show on the labels for the x-axis.)

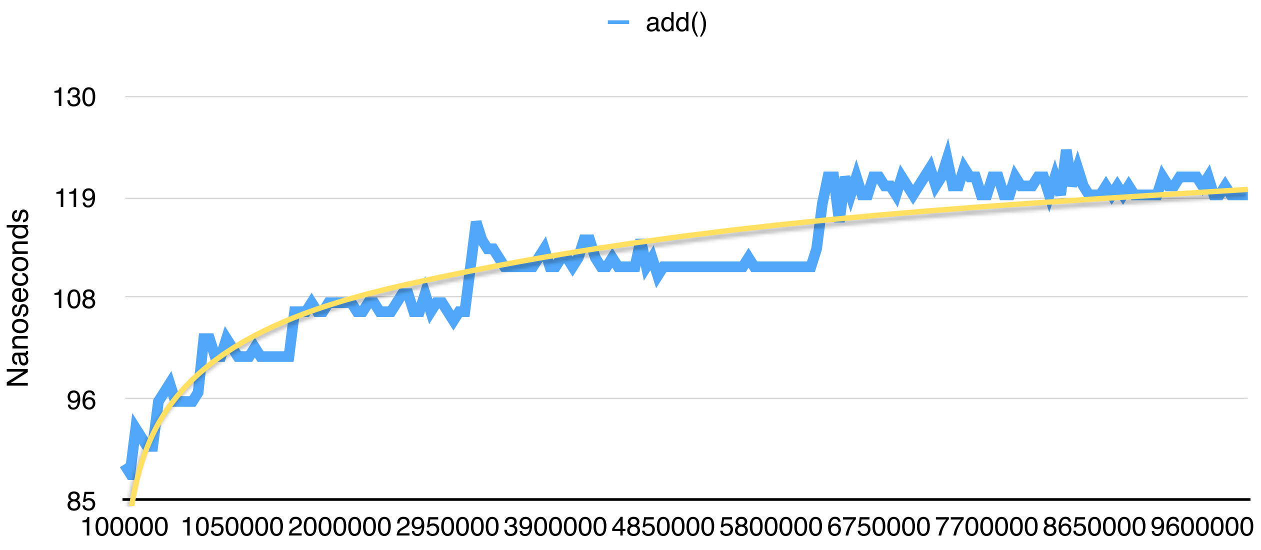
You know, I didn't really appreciate the gravity of what we were taught in lab 1 until this project. I even went as far as to implement test methods the way that I thought would be best. However, after reviewing the content in lab one. I've gained an understanding of just how powerful taking timings in that manner is. Needless to say our timing methods were re-written and now provide much better data than before. Thats the kind of stuff you can't learn from "teach yourself how to code" books. Go Utes!

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)?

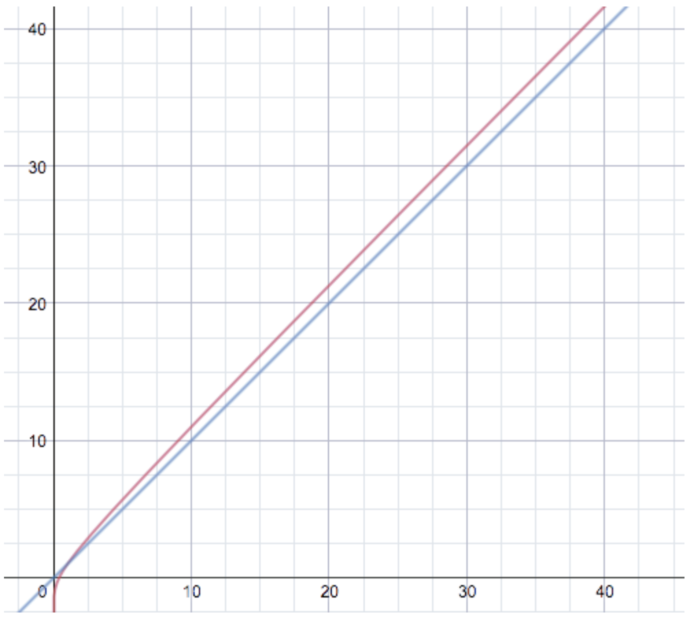
All of the methods that need to search use the same private binary search method. The search method returns the index of where the object should be. It is then up to the more specialized methods to check the return value in the data array against their arguments. So in regards to add(), to locate the correct position is binary, and therefore O[Log(N)]. However, it is not that simple when you actually need to add an item to the array:



The trend line suggest that add() follows O[Log(N)], however how could that be if we add the worst case scenario to the array each time? If the structure contains the integers 1 through 10 and we add 0, we'll have to shift each member of the array over by one. Which would be linear right? We need to hit every single piece of data after we add. That makes me believe that add() is really O[Log(N) + N]. However, the table above looks more like O[Log(N)]. So we set the size limits on our test to 10,000,000 and our averaging factor to 10,000,000 and ran another test:



It still looks like O[Log(N)] to me. Which is confusing considering we have to shift all of our data over by one. So what is happening here? If I plug y = Log(x) + x and y = x into a graphing calculator it looks like this:



The red line is y = Log(x) + x and the blue line is the linear function y = x;

So perhaps our test is doing something wrong. I hope that if you have time, you could take a look at the test and our add() method and provide some much needed clarity to both myself and my partner.

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

Its probably a valuable workplace skill to track ones hours spent on projects, but I'm honestly not sure. I'd wager we spend upwards to 15 hours total on this one.