Cashe Rumsey

When you are satisfied that your program is correct, write a brief analysis document. The analysis document is 30% of your Assignment 3 grade. Ensure that your analysis document addresses the following.

1. Who is your programming partner? Which of you submitted the source code of your program?
    a. ***Zach Warrell was my programming partner and he is submitting 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?
    a. ***We switched off as needed, both inputting ideas and typing them when we had them. This worked out perfectly—I'd prefer no changes.***

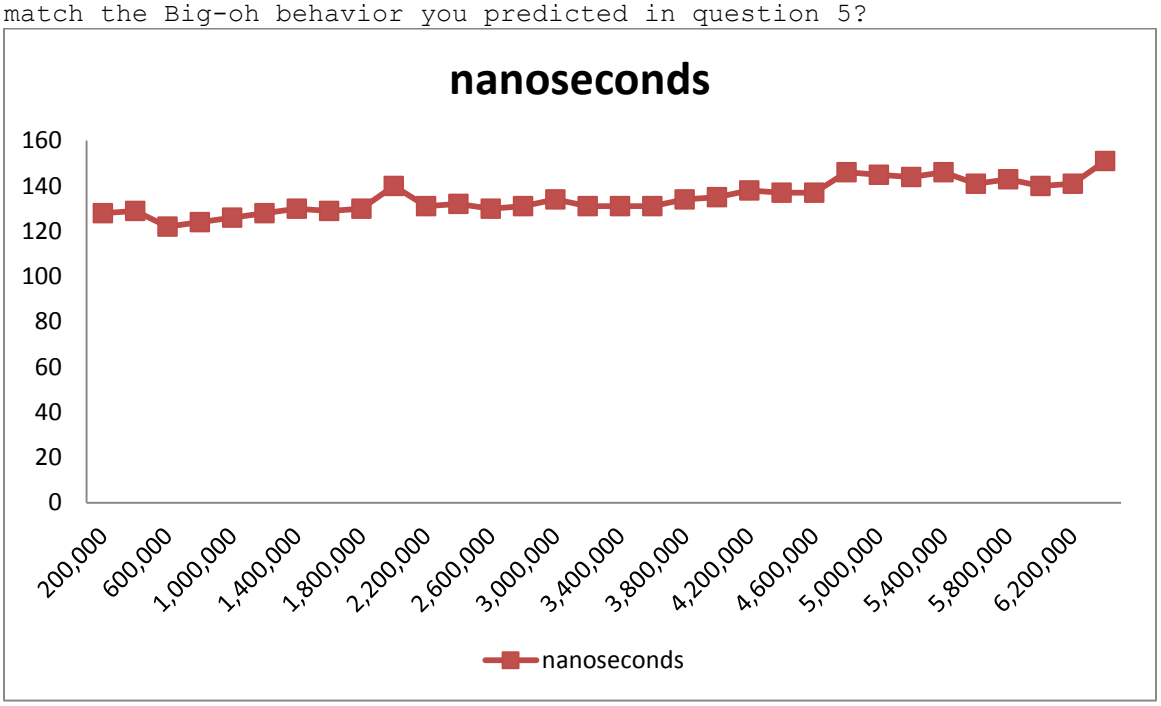
3. Evaluate your programming partner. Do you plan to work with this person again?
    a. ***Yes, I plan on working with Zach again. I feel like together we make a great programming team and complete the assignments efficiently and promptly.***

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.)
    a. ***If I was allowed to use a List, THAT WOULD BE AWESOME! Using an array, we had to implement our own searching, sorting, adding, removing, etc, algorithms that, I can guarantee, aren't as fast as Java's implemented methods with Lists. Using a List, we definitely could have been more efficient, for reasons stated above. Other reasons are: Lists are much faster when iterating sequentially (which we did a lot of), and are also much faster when inserting things. Not only would it be run-time efficient, but it would be time efficient for me. Having all the add, remove, etc, methods implemented would be just great—but terrible for learning purposes.***

5. What do you expect the Big-O behavior of MySortedSet's contains method to be and why?
    a. ***The contains() method is essentially just running the binarySearch() method that we implemented. The Big-O notation of that is O = (log(n)). Everything else in the method is constant, c, and does not make a notable difference. The reason we get (log(n)) on binarySearch() is because the function goes to the middle, if it doesn't find what it's looking for, finds a new midpoint ((high + low) / 2), resets its upper or lower boundary depending on if it needs to go higher or lower, then goes to the newly declared midpoint and searches again. It does this until it finds the location that it is looking for. All the diving by two gives us our O(log(n)). This method is a ton faster than O(n) because it doesn't have to iterate through each element individually. It also isn't constant because we have to iterate n number of times before we find the correct position, or element.***

If we do not recognize that binary search is O(log(n)), we can just reflect back to our notes in class and see that the only thing that comes between O(c) and O(n) is O(log(n)). Either way, the time it takes is "t + 0.25", which correlates to O(log(n)), where O(n) is "t*2".

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?



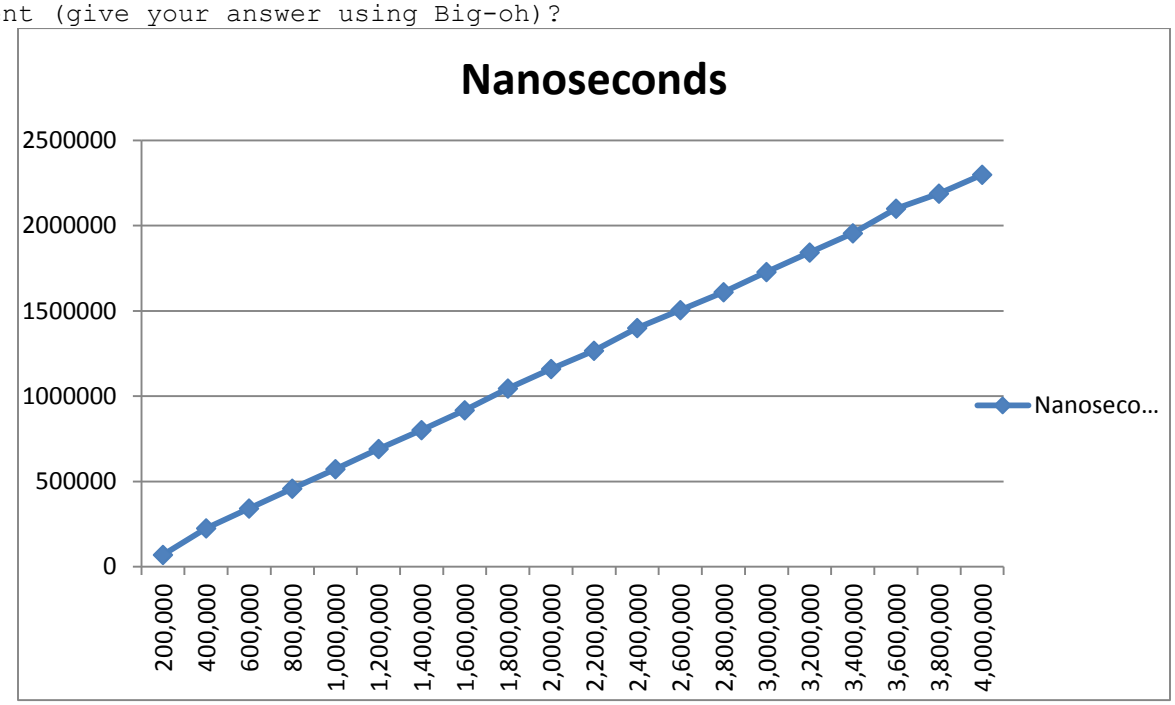
a.

b. ***The plot that we received from going up to 2,000,000 looked like a constant function, and it still revealed this going up to 6.2 million. The first and last varied by no more than 10-20 nanoseconds. While our results from timing seem very off, our binarySearch() algorithm is definitely not incorrect. We verified this with John and had him look over our timing code. We also took the code to several other people in our class, as well as two different people who work as software engineers who are in upper-level CS classes; no one was able to tell us exactly why our timing code was not working correctly, and all agreed that our binarySearch() algorithm was correct. I don't know if you can narrow down the source of the problem. If not, I think I'm going to publish my code on how to write a binarySearch() program with O(c) run-time and make millions! Anyways, our graph did not show us what we expected, which was a O(log(n))function, instead we got what looked like a O(c)function.***

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)?

a.



b. ***To add an element, the function is f(n) = n + log(n) + c. Since the leading term is the only thing that matters with large n, the function is actually just O(n). The worst case scenario is f(n) = n, where the average case would be f(n) = n / 2. The reason for f(n) = n, being the worst case is because it would mean having to iterate through every n, before getting to the desired result. Whereas, with f(n) = n / 2, that would mean only needing to iterate through half the n, before finding the desired location—which is an average.***

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

We probably spent about 7-8 hours on the assignment.

Programming partners are encouraged to collaborate on the answers to these questions. However, each partner must write and submit his/her own solutions.

Upload your document (.pdf only!) to the Assignment 3 page by 11:59pm on February 5.