Jordan Davis
Feb 5, 2014
Assignment 3 – Cs 2420
Assessment Document

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

My Partner is Jacob Osterloh. I submitted 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 switched roles roughly every hour. I would have liked to be the driver more because it helps me stay focused and I feel I have a better grasp at what is going on. I often get lost when I'm trying to navigate and it frustrates me when the driver doesn't understand what I'm trying to say in code.

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

Jake is a good partner. We both work well together and have the same schedule, which makes it easy to find time to work on the assignments. Constructive criticism is always helpful from him and we teach each other as we learn new things.

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 been more or less efficient and why?(Consider efficiency both in running time and in program development time.)

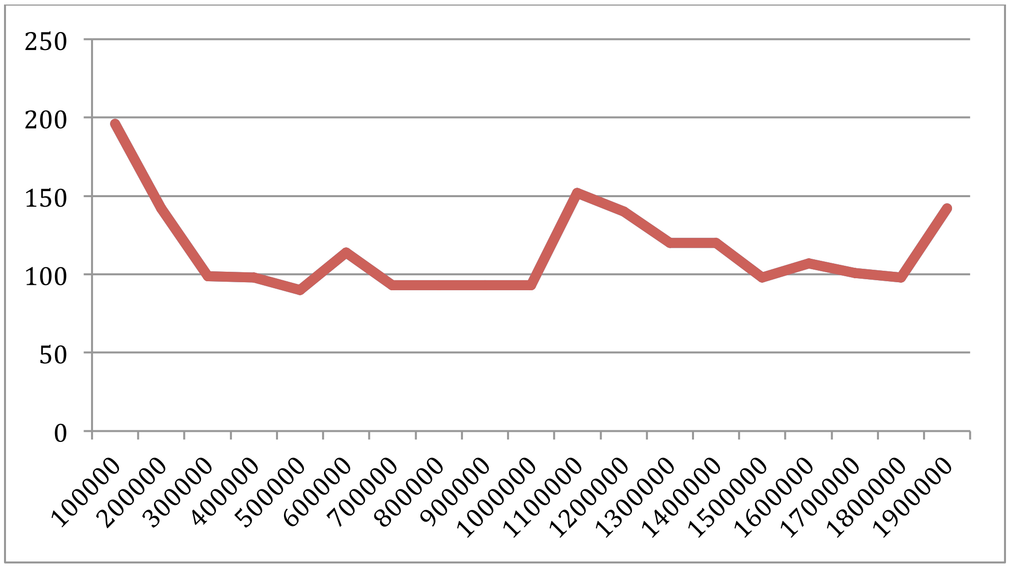
Lists would have handled resizing the set, which would have simplified the code, and program development time would have been much faster. We ran into many issues with growing the array. We might have also been able to add the objects into hash sets so that we didn't have to check for duplicates. Many things we did in this assignment could have been done by using Java Lists, but it was very educational learning how to do it from scratch. In the end, there isn't that much difference in run-time-efficiency, however development timing was definitely increased.

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

I suppose that it would be O(N) because depending on the size of the array it will take longer to search through the set using binary, however, it wont take that long thanks to the efficiency of binary searches.

6. Plot the running time of MySortedSet's contains method for sets of sizes100000 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-(O) behavior you predicted in question 5?

__**Plot of Contains Method Time Test:**__



To me it looks very similar to what I expected, however it seems to be more efficient then I supposed. Since we are using very large numbers in our set and the average is staying the same, it seems like it could possibly be O(log n) rather than O(n).

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-O)?

Our add method has many other aspects to it, such as shifting the array and resizing the array which makes it take longer than the locating search. After taking out the time for all those other aspects of the add method, the binary search in the add method efficiently finds the correct location to which the object should be added, the Big-O notation for it is O(log N), and worst case would also be O(log N).

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

Roughly 19 hours. Lots of caffeine, de-bugging, and testing.