Andy Dao | uID: u0692334 | CS 2420

ANALYSIS DOCUMENT ASSIGNMENT 4

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

   Moses Manning was my partner. I will be the one to turn in the source code.

2. **What did you learn from your partner? What did your partner learn from you?**

   I learned more of how insertion sort is supposed to be implemented. I didn't exactly grasp insertion sort 100% until he ran it through with me pseudo code and watching him implement most (if not all) of the insertion sort algorithm.

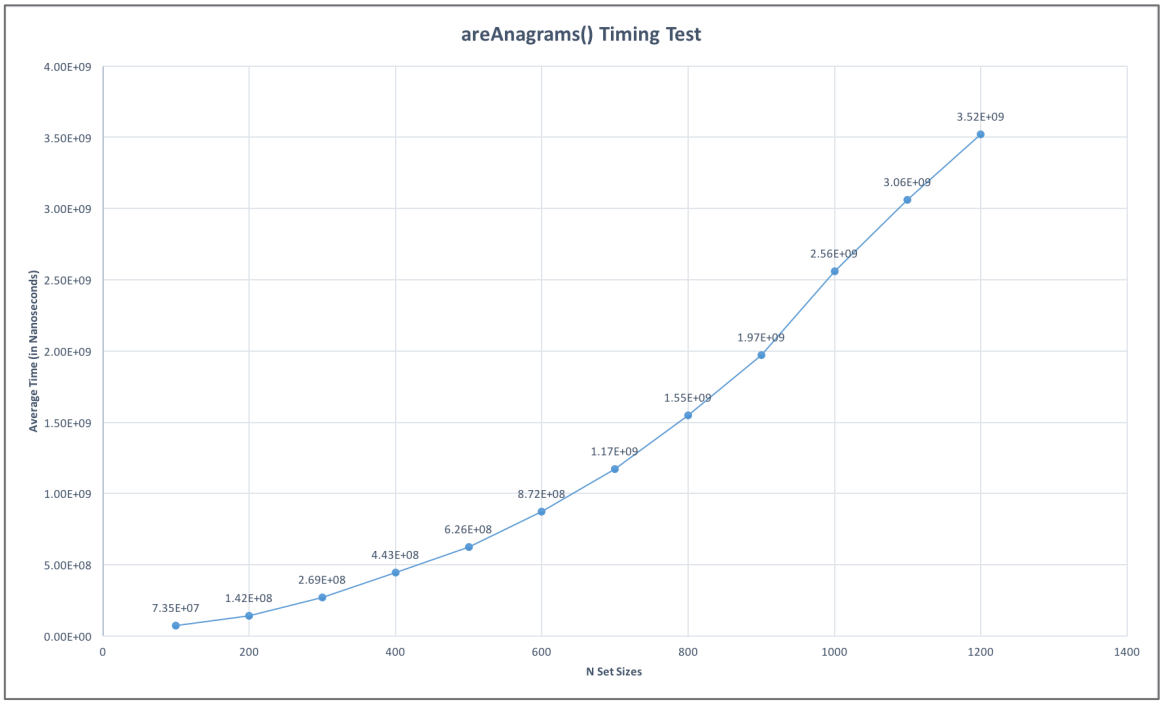
   What my partner learned from me was double checking code and implementation. I was very persistent on that it should be running O(n^2) for our run times, and when it wasn't coming out that way, we kept going back to check and change what was going on and we found out the solution. So in short, just being persistent and double checking the code multiple times for errors.

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

   Moses as always comes reliable. Communication is more than enough and very good. I had a bit of schedule conflict due to some family emergency happening during this week's assignment and he was very understanding. We remained caught up with each other mostly through github this time, which in itself is a learning experience but we've had experience with it previously. We did meet up though, so that wasn't out of the question (as the rules for partner assignments).

   He's a great driver. He's really good at explaining his thought processes and why he's implementing the way he's implementing it. I do plan on working with Moses again in the future.

4. **Analyze the run-time performance of the areAnagrams method.**
   -What is the Big-O behavior and why? Be sure to define N.
   It looks to be about O(n^2) for the amount of set sizes I plotted. I had made a strict timing restriction where if the plot for a set size finally got over 30 seconds to time, then I stopped and plotted what I had. I started my N to be 100 and had it double its' size each time. The reason why it acts like O(n^2) is because our implementation for our sort method uses insertion sort, which has a Big-O Complexity of O(n^2) average case. So thus, this is why our graph/data seems to be O(n^2) as well.

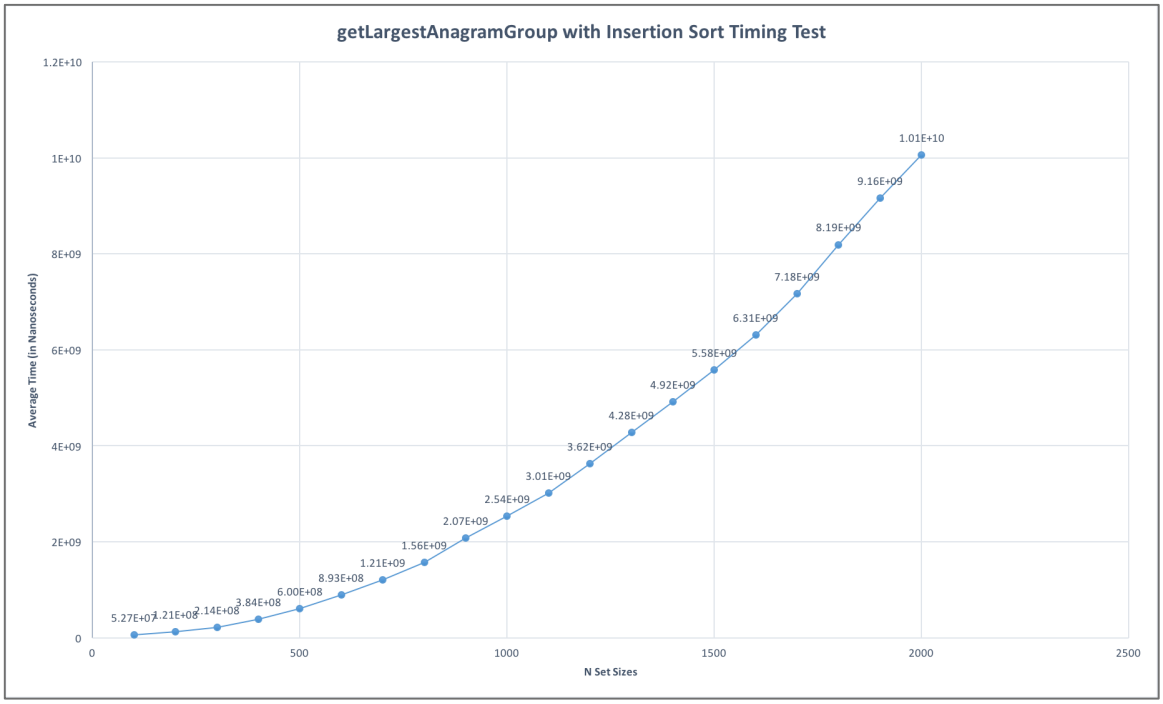


   -Does the growth rate of the plotted running times match the Big-O behavior you predicted?
   Yes. Though I explained my reasoning why just previously, it's because we use insertion sort to sort which has a runtime of O(n^2).

5. **Analyze the run-time performance of the getLargestAnagramGroup method using your insertion sort algorithm. (Use the same list of guiding questions as in #4.) Note that in this case, N is the number of words, not the length of words. Finding the largest group of anagrams involves sorting the entire list of words based on some criteria (not the natural ordering). To get varying input size, consider using the very large list of words linked on the assignment page, save it as a file, and take out words as necessary to**

get different problem sizes, or use a random word generator, provided in AnagramTester.java. If you use the random word generator, use a modest word length, such as 5-15 characters.

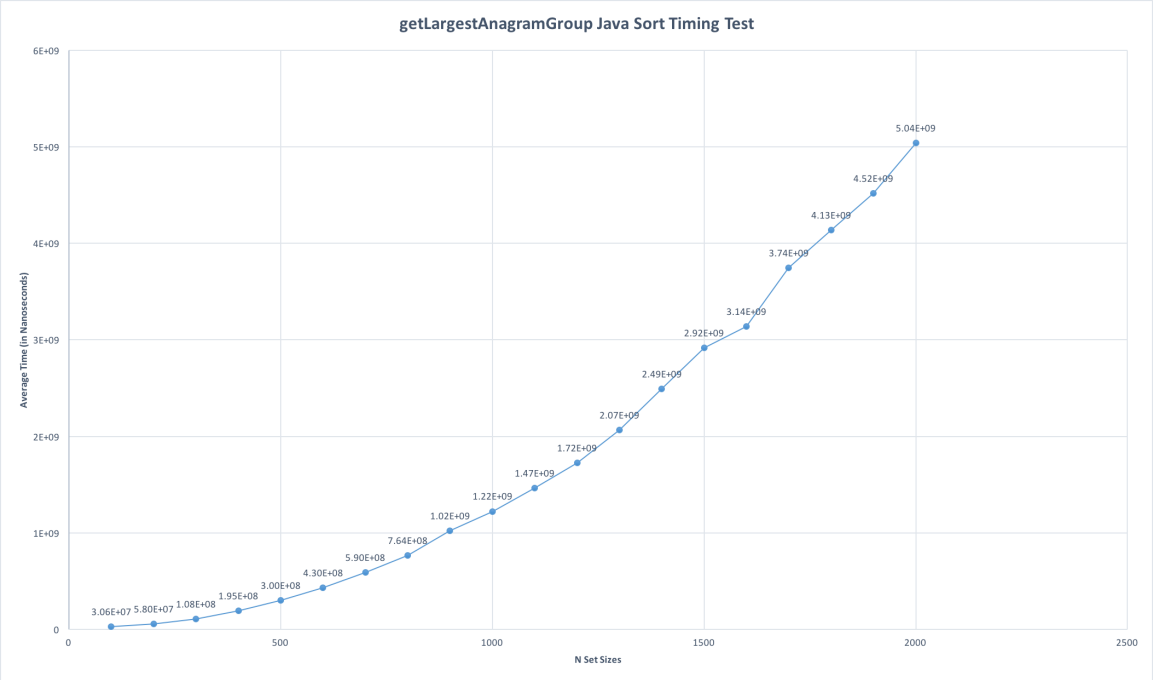
Here is our result for our getLargestAnagramGroup() method using our insertion sort algorithm:



Our Big-O behavior is expected to be O(n^2) as well since we are still in the end, using our insertion sort method which has a run time of O(n^2). Since getting the largest anagram group involves insertion sort, we concluded that it would just run O(n^2) at the end anyways and we were correct about that.

6. **What is the run-time performance of the getLargestAnagramGroup method if we use Java's sort method instead)?**

   To start off, here is the plot graph for using java's sort method:



It looks to be O(n log n), as it says in the Java API for java's sort method. Since java 6, it basically uses a merge sort algorithm to perform it's sorting. That is the reason why it runs at a run time of O(n log n) due to a divide and conquer implementation coincide with our getLargestAnagramGroup implementation. It does match it what we expected because since looking online on the Java API, it states that it runs at a O(n log n) runtime.

7. **How many hours did you spend on this assignment?**
   15 hours