Christian Hansen
U0621884

Analysis Document

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

Jared Nielson was my programming partner. He submitted the source code.

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

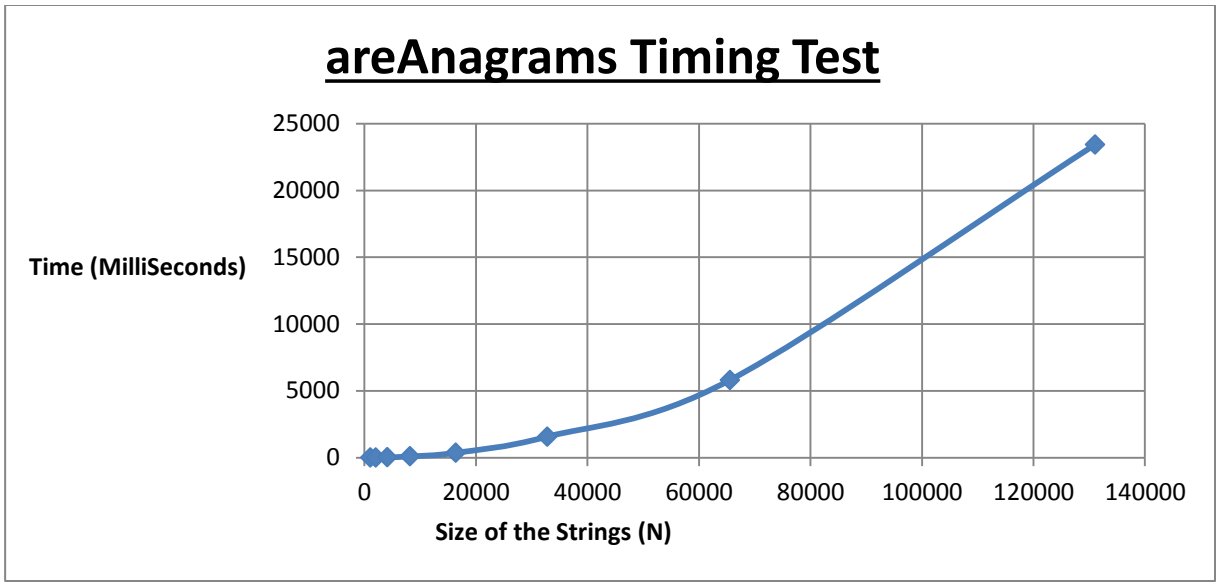
I learned from my partner how to approach difficult algorithms. As we were programming the sorting methods, He was very methodical as he thought through what each line of code did and what we needed it to do. I feel that I helped us implement tests sooner which helped us determine if our function were performing correctly. I feel he might have learned some about the importance of testing sooner from me.

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

Yes, we plan to work together again on this next assignment. We work very well together, we respect each other's opinions, and we can easily find time to meet up together to work on assignments. Jared is a sharp guy and there would be no reason to switch partners right now.

4. Analyze the run-time performance of the areAnagrams method.
-What is the Big-O behavior and why? Be sure to define N.
-Plot the running time for various problem sizes (up to you to choose problem sizes that sufficiently analyze the problem). (NOTE: The provided AnagramTester.java contains a method for generating a random string of a certain length.)
-Does the growth rate of the plotted running times match the Big-O behavior you predicted?

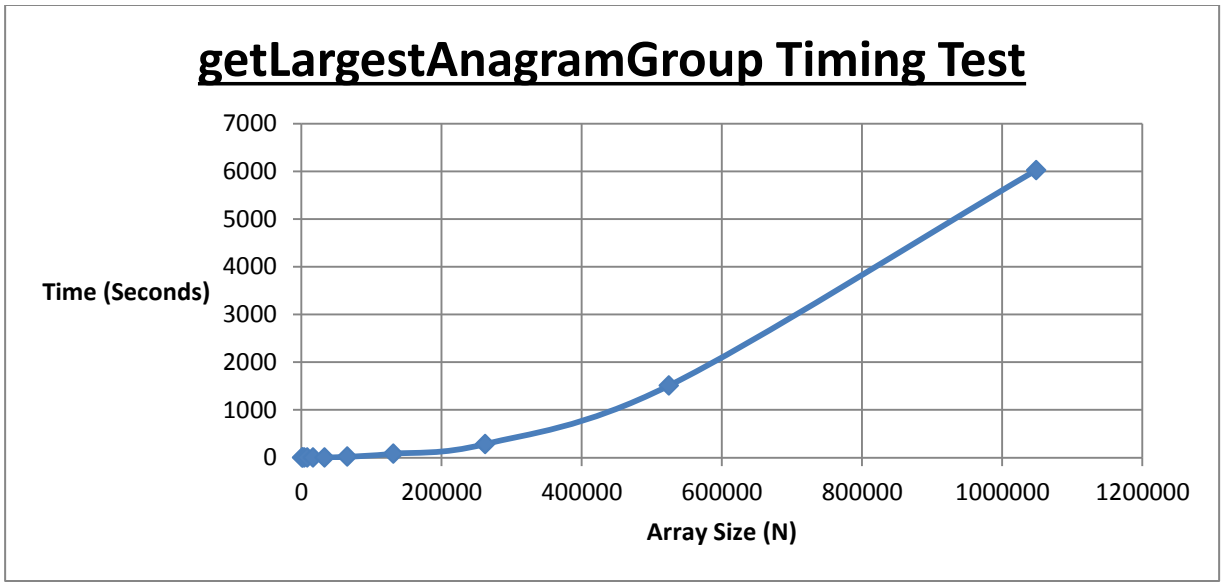
The Big-O behavior is N^2. N refers to the length of the two strings being compared. N^2 is the Big-O behavior, because on inspecting the method, there are two single for loops and there is another for loop with a for loop nested inside it. As the size of the strings increases, the only part that affects performance is the for loop with the other for loop nested inside it and that behavior is modeled by N^2. The graph for the areAnagrams timing test below shows this. As size doubles, performance quadruples which is expected by Big-O behavior N^2.



This graph displays N^2 as was expected. The rate of change between the last two points clearly shows around a quadruple increase which supports the N^2 prediction.

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.

The Bio-O behavior is N^2. To get the Largest Anagdram Group, all of the strings are sorted using the sort method. Then the array is sorted using the insertion sort method. This is the part that determines the time it takes for the code to run because as the size of the array increases to infinity, the other code is very miniscule in comparison. The graph below is a model of the data from a timing test run on getLargestAnagramGroup. It matches the predicted Big-O behavior of N^2. It is shown that as the size doubles, the time quadruples which is what is expected.



6. What is the run-time performance of the getLargestAnagramGroup method if we use Java's sort method instead (http://docs.oracle.com/javase/6/docs/api/java/util/Arrays.html)? How does it compare to using insertion sort? (Use the same list of guiding questions as in #4.)

If the Java sort method is used instead of the insertion sort method that we made, the run-time performance will improve significantly. Java uses a sorting method called Tim Sort instead of Insertion Sort. This has a run-time complexity of N(Log(N)), which is much more efficient than N^2.

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

We spent around 10 hours working on this assignment.