Nathan Milot
CS 2420
09/21/16

Assignment 04 Analysis

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

   Axel Kerksiek, u0691509. I did.

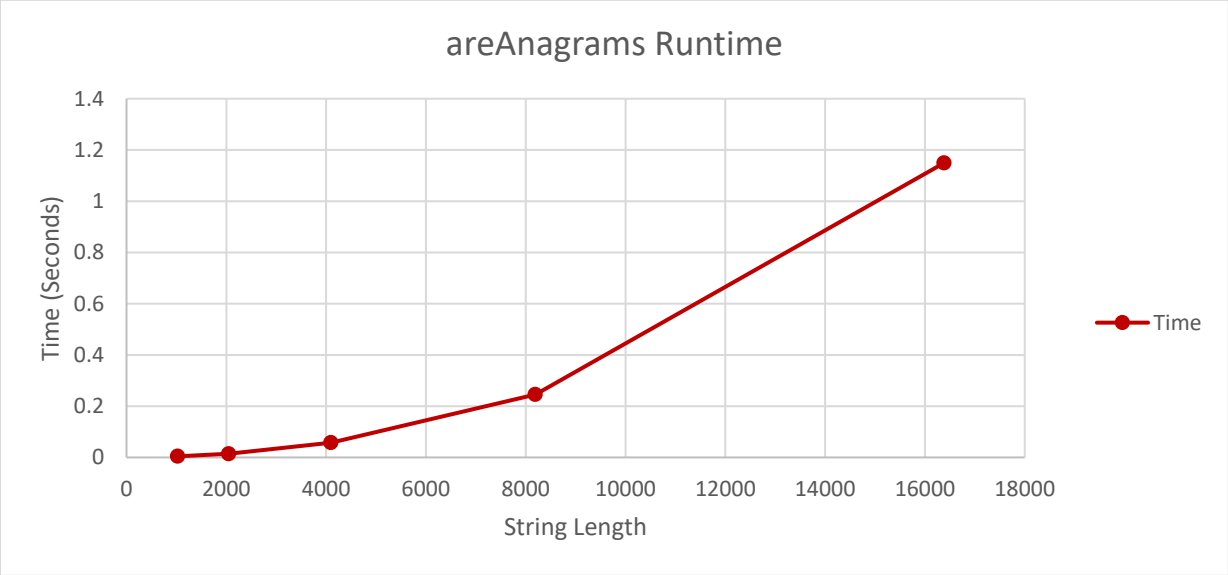
2. What did you learn from your partner? What did your partner learn from you?

   Axel thought about the problem and suggested a great way to solve the getLargestAnagramGroup method when I hadn't thought of a solution. He suggested we use a Map to simplify the tracking. I hadn't thought about that, so he taught me the way we would solve the problem. I had more specific knowledge on how to work with a Map, so I was able to teach him about that so we were able to solve the problem easily.

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

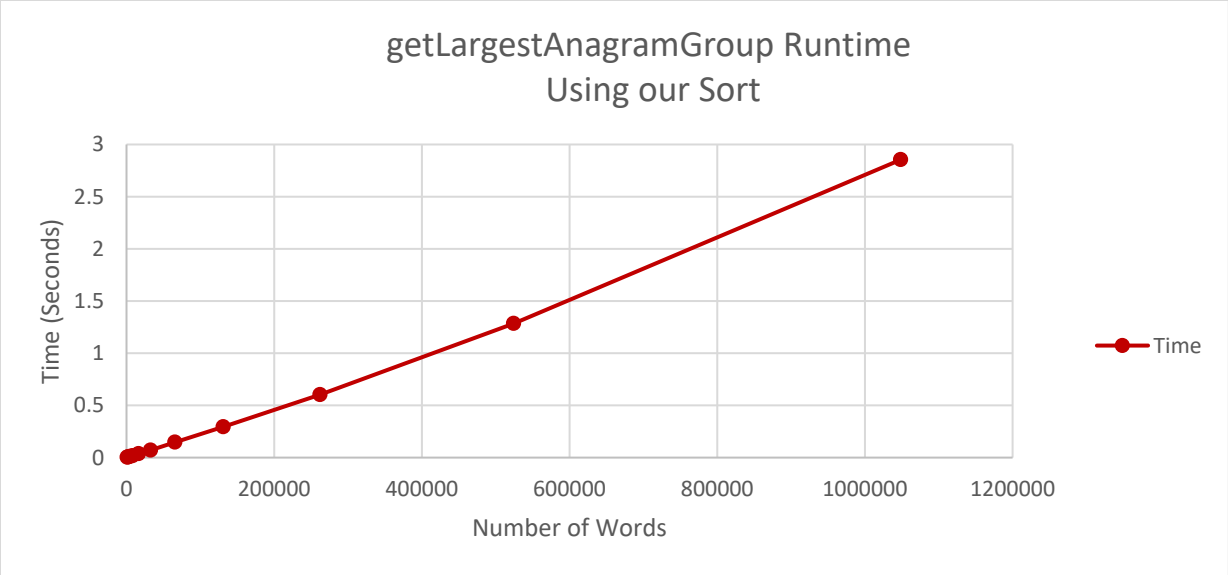
   Again, I really enjoyed working with Axel. I feel like his contributions, both while we were coding and outside time dedicated to thinking about the assignment, really helped move the project along and made getting our solutions easier. I think he did a great job and I do plan on working with him again.

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?



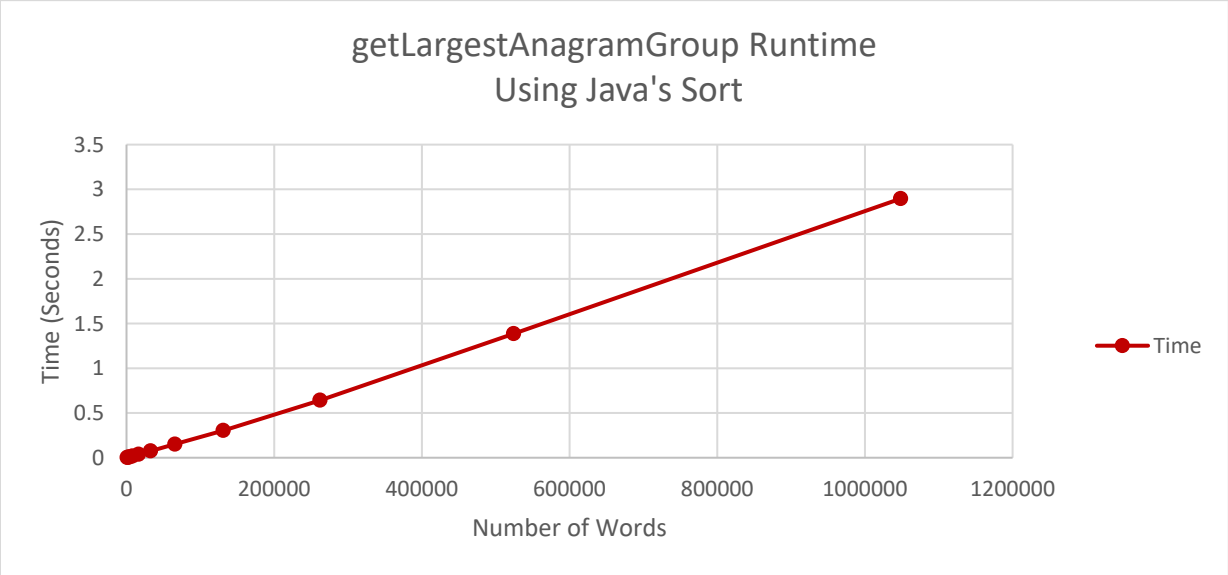
For the areAnagrams method the Big-O behavior is $O(N^2)$, for worst case, where N is the number of letters in each String. This is since the method implements an insertion sort and the Strings that were passed were random and therefore not in lexicographical order, so the sorting algorithm had to do the most work, or at least an average amount of work, to fix the inversions which leads to $O(N^2)$ behavior. Our plot shows this behavior as it has a quadratic curve.

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.



Our getLargestAnagramGroup method, using our insertion sort algorithm, performed with O(N) behavior. To get this data, we used randomly generated Strings that were seven characters long. This is the behavior I was expecting, since insertion sort works very well for short sized problems.

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.)



The getLargestAnagramGroup method, using Java's sort method, performed with O(N) behavior. Interestingly enough, our insertion sort method was actually faster in almost every instance of our testing data. Possibly as the data size gets even larger Java's sort might be more effective, but up to about a million data points ours was still more efficient.

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

About 7 to 10 hours.