Andrew Keaton Bruce
u1006889
CS 2420
9/21/2016

Analysis of Assignment 04

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

My partner is Longsheng Du. I submitted the code for the program.

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

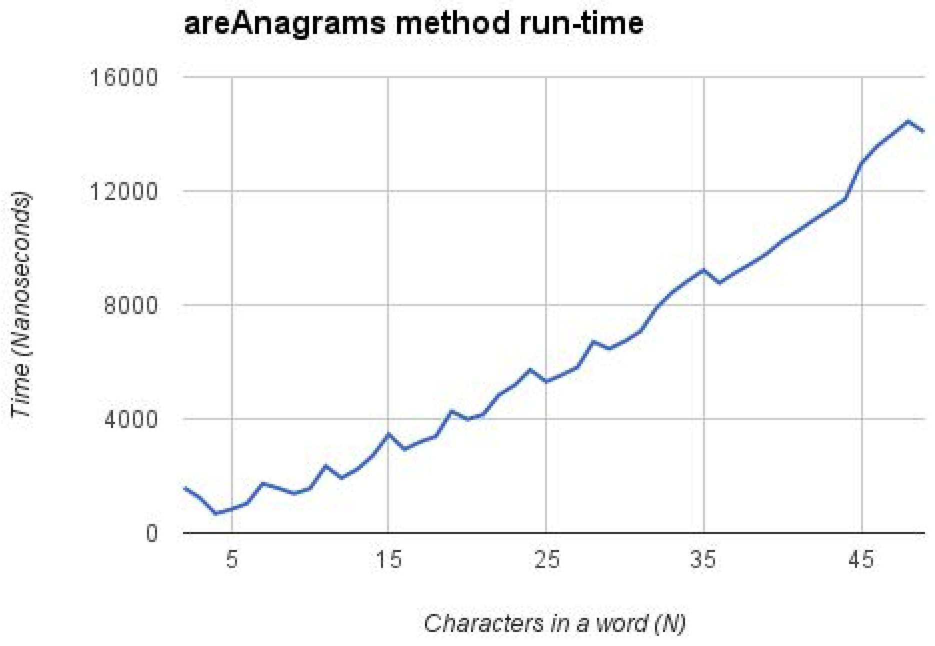
Longsheng showed me how to properly implement many of these sorting methods and I showed him good ways of naming variables and commenting.

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

Longsheng is very good and helps me a lot. I plan to continue working with him.

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?

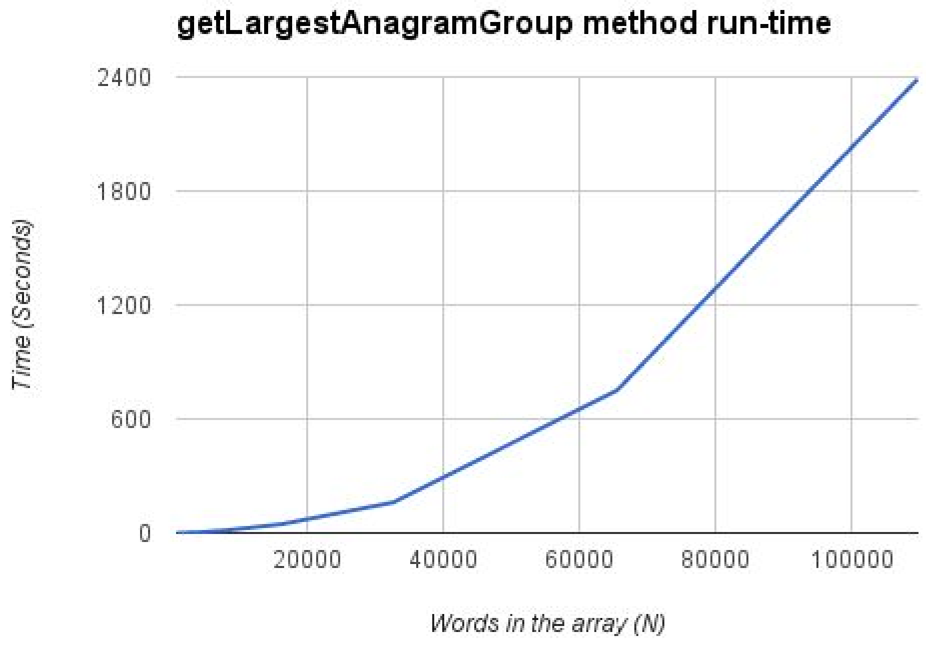
The areAnagrams method is complexity $O(N^2)$ in the worst and average case. This is because it sorts each string by their characters using insertion sort. However, because real English words are not usually longer than around 30 characters or so, we tested our areAnagrams method with strings of length less than 50. The Big-O behavior in this case looks to be linear or O(N). N in this case is the length of the string or word.



As you can see here, the time it takes to sort a string by characters does increase gradually over time. However we do not see the O(N^2) behavior because the N is so small. Here we see a fairly linear line.

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.

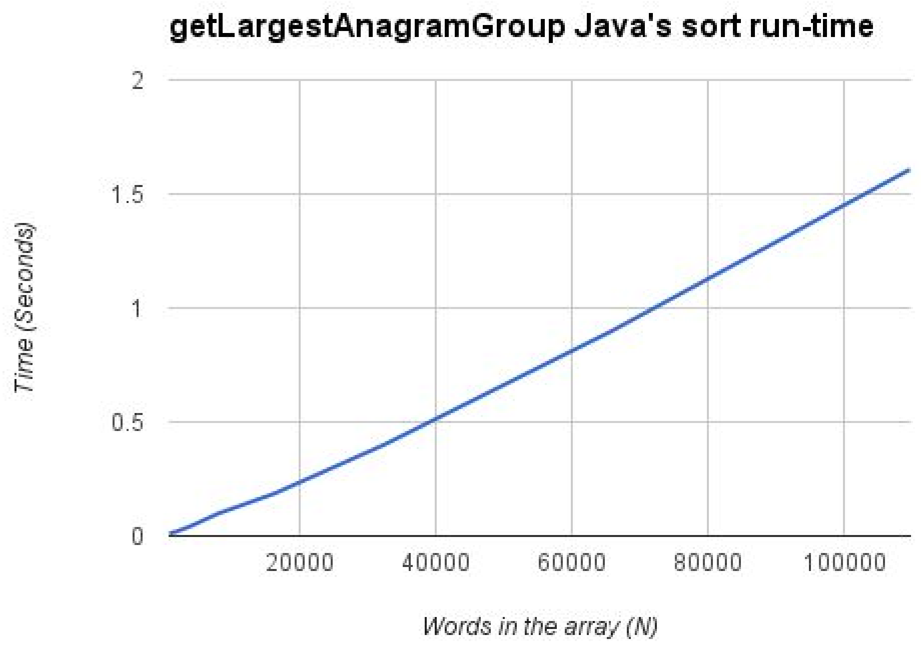
For this part of the assignment we used a HUGE list, about 109,000 words, to test the speed of getLargestAnagramGroup. Each word is just an average size word. Because of this, getLargestAnagramGroup is probably the same complexity as areAnagrams, O(N^2) in the worst and O(N) in the best case. This however only applies if the words are very small, like less than 50 characters.



This next graph does begin to show the O(N^2) behavior because of the large N.

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 runtime of a modified getLargestAnagramGroup that uses Java's Arrays.sort is far faster than our implementation of insertion sort. It completes in only a fraction of the time that we had to wait. I believe this is because Arrays.sort uses mergesort instead of insertionsort.



This graph shows how fast this method performs in comparison to our own. At the maximum size array that we used, about 109,000 words, this method only took about 2 seconds to complete. Our insertion sort took 40 minutes to complete sorting an array of that size.

7. How many hours did you spend on this assignment?
12 hours. 10 hours for the code and 2 hour for the analysis.