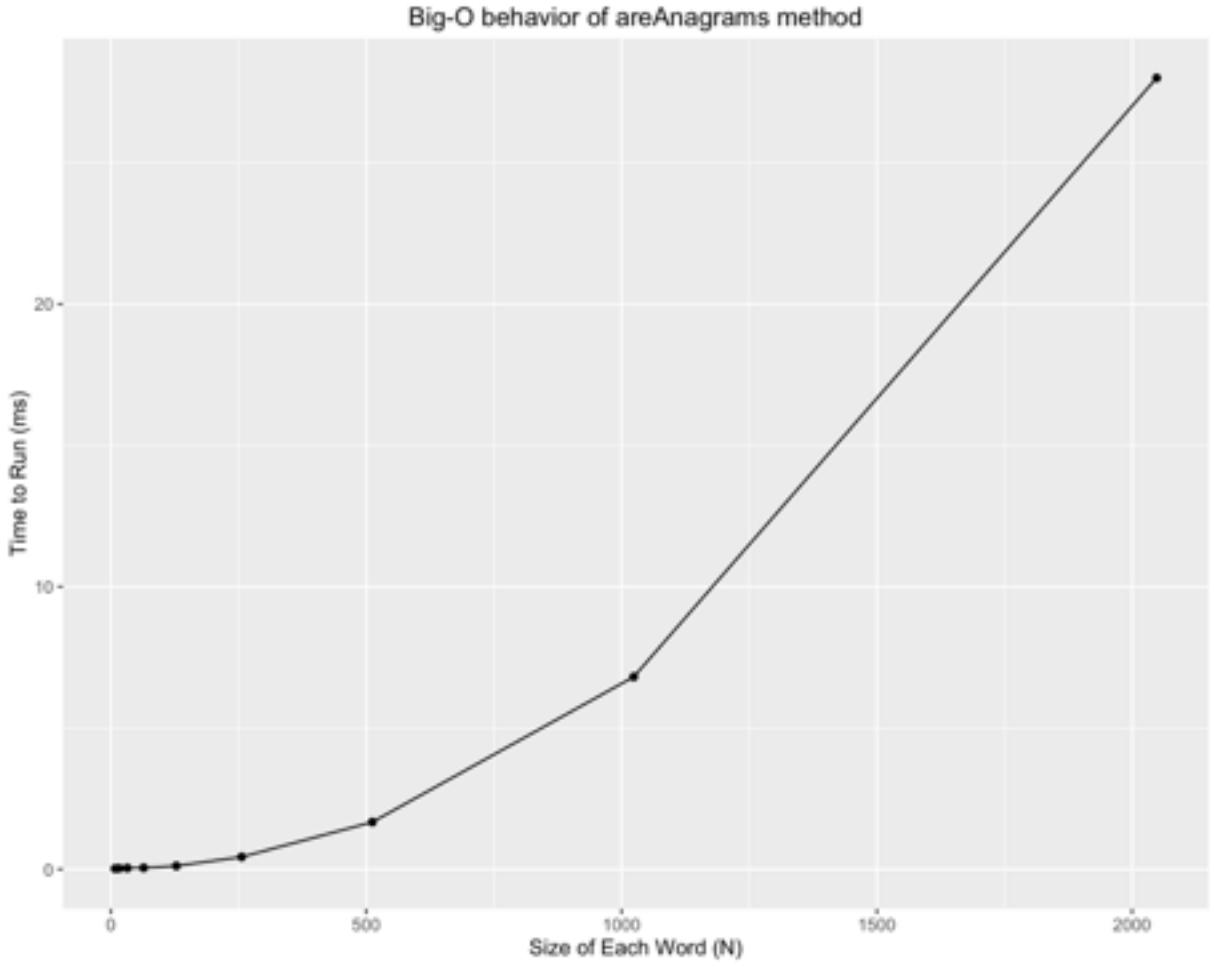
Kira Parker u1073760

1.Who is your programming partner? Which of you submitted the source code of your program?
My parter is Nathaniel Coleman. I submitted the source code for the program.

2. What did you learn from your partner? What did your partner learn from you?
I learned how to comment more effectively. Because Nathaniel was in Paris for most of the week we had to work on the assignment, we had to work separately and then come together at the end and discuss how we implemented each method and how to make it better and deal with edge cases. It was hard for us to understand some parts of my code, and I learned where I need comments and where comments are completely redundant. My partner learned more effective commenting as well, and we got better at debugging.

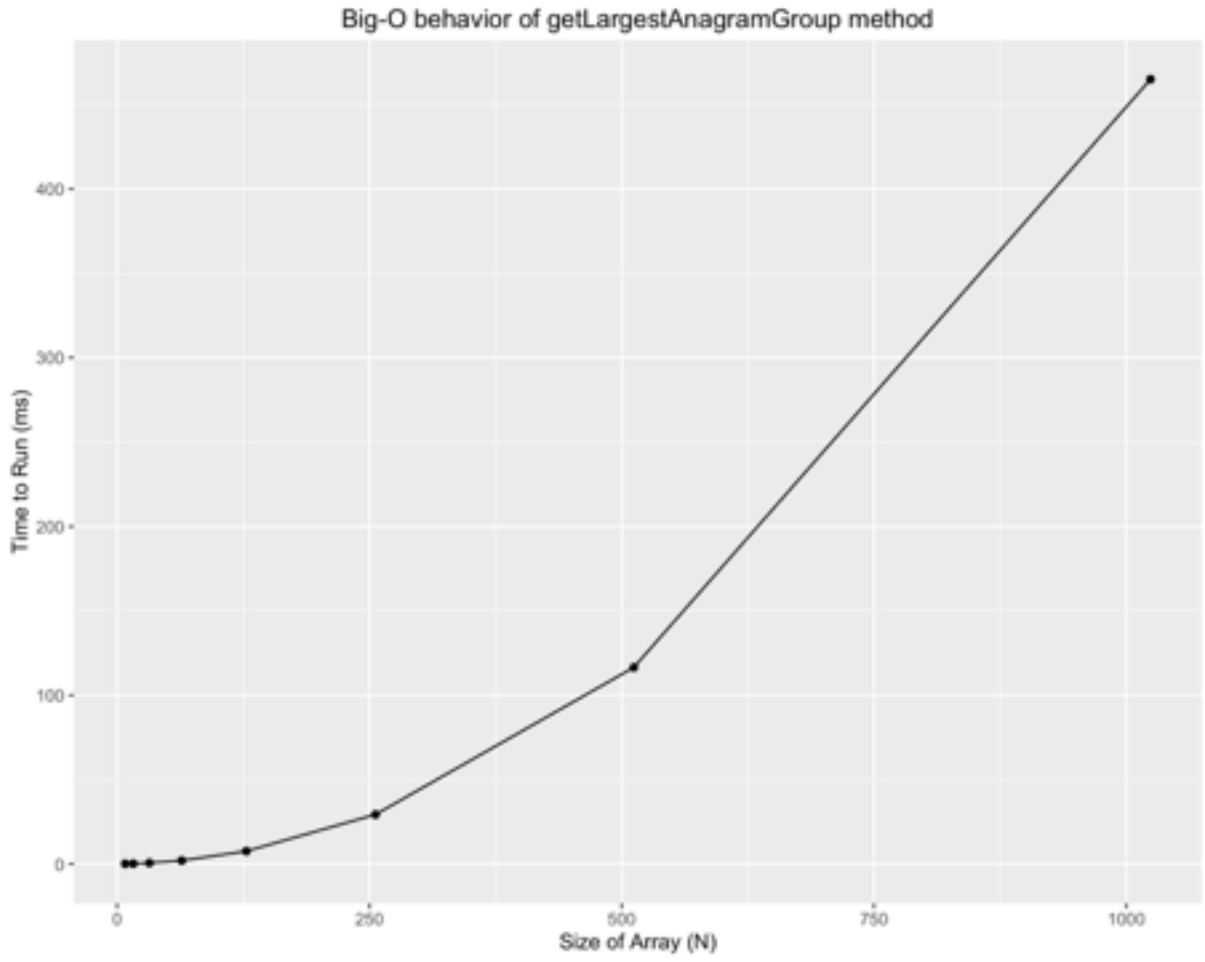
3. Evaluate your programming partner. Do you plan to work with this person again?
I would definitely work with Nathaniel again. He offered thoughtful suggestions on how to implement different methods and was very thorough in reading the assignment specifications. Our ideas worked together quite well, and where one of us didn't understand how to implement something the other would have an idea.

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).
-Does the growth rate of the plotted running times match the Big-O behavior you predicted?

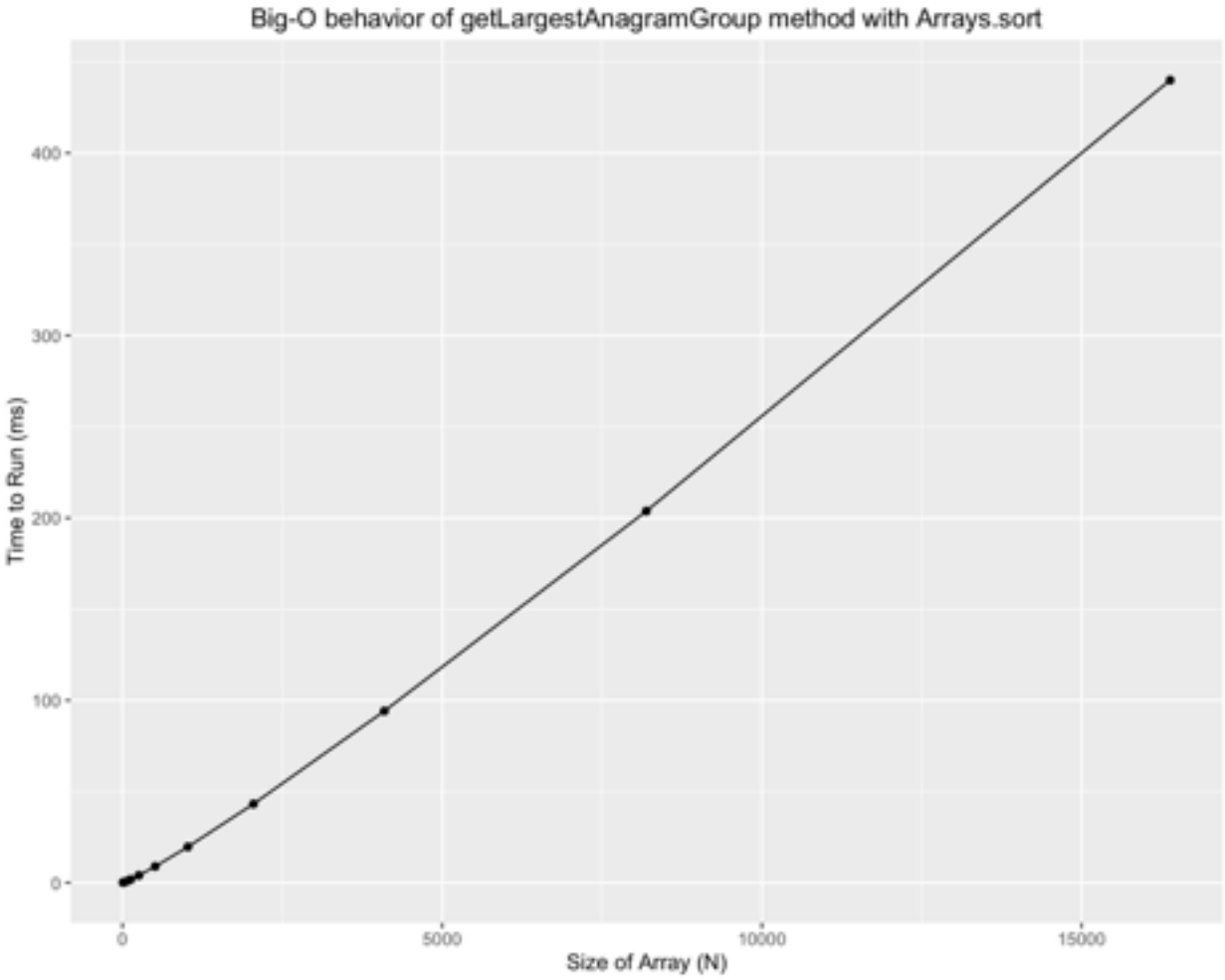


The Big-O behavior should be $O(N^2)$, where N is the length of the input Strings, because we are using insertion sort to sort the letters in each word, which is a quadratic algorithm in its worst and average case. The only other thing that happens in the method is the sorted strings are compared to see if they are equal, and that is not quadratic so the Big-O behavior should be $O(N^2)$. The growth rate of these plotted running times matches the behavior I predicted.

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 run-time performance should be quadratic because insertion sort is $O(N^2)$ and the code where we determine the largest group of anagrams in the sorted list should be O(N) since we loop through the list once, find the biggest group, and then in a separate loop copy the elements that are part of that group into a new array. The growth rate of the plotted times matches the predicted behavior.



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 Big-O behavior appears to be O(N) when we use Java's sort method instead of insertion sort, so their sorting method is faster than the one we implemented. This makes sense because on the Arrays sort method says that it offers guaranteed O(NlogN) performance.

7. How many hours did you spend on this assignment?
We spent about 8 hours on the assignment.