Brayden Carlson
u0959889

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

Diego Baez Lopez is my partner. My partner turned in the source code.

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

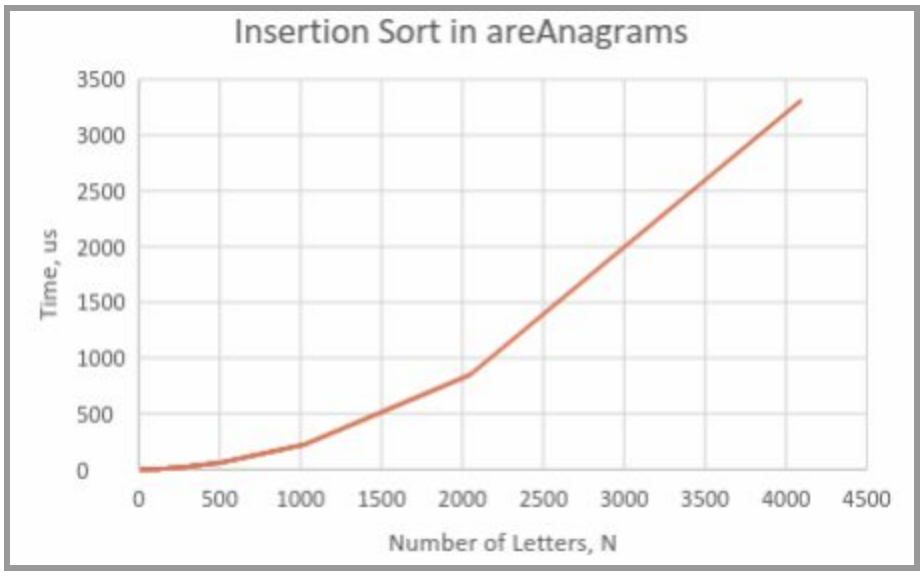
My partner showed me a few testing tricks that I will now use from now on. I taught my partner a few debugging tricks.

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

I really like my programming partner. We get our homework done fast and efficiently, and we get along well. I do plan on working with this person 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 providedAnagramTester.java contains a method for generating a random string of a certainlength.)**
**-Does the growth rate of the plotted running times match the Big-O behavior you predicted?**

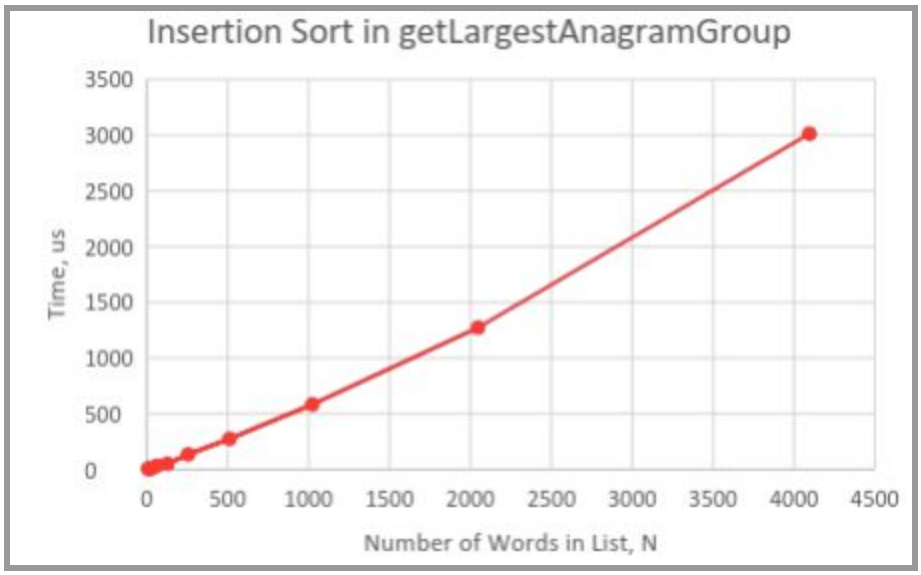
The Big-O behavior of areAnagrams should be O(n^2) where n is the number of letters in the words. It is O(n^2) because because it uses two insertion sorts, which are O(n^2). Insertion sort is O(n^2), due to it having having a for loop and an inner while loop contained inside.




The graph does appear to be 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.**

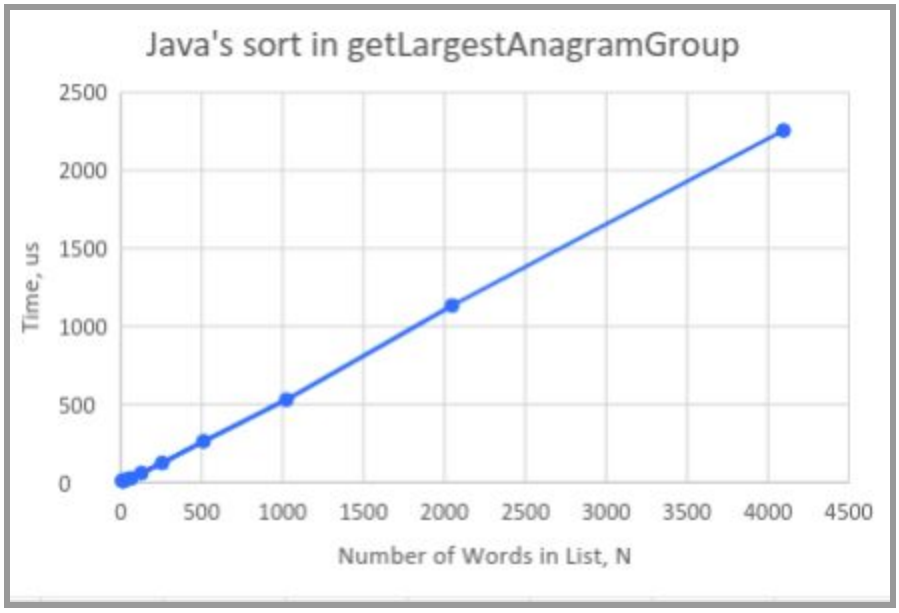
The Big-O behavior of getLargestAnagramGroup should also be O(n^2), where n is the number of words in a list. This is due to the getLargestAnagramGroup using the areAnagrams method in its implementation which is also O(n^2).




It appears to be O(n^2). As expected this graph is extremely similar in run-time to the graph of areAnagrams which is also O(n^2).

**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.)**

According to the documentation, Java's sort method should be a Big-O behavior of O(nlog(n)), where n is the number of words in the list.




This graph appears to have a lot less curve to it, indicating it's probably O(log(n)) just as the documentation says. It also appears to have run faster than our sort method.

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

8 - 10 hours.