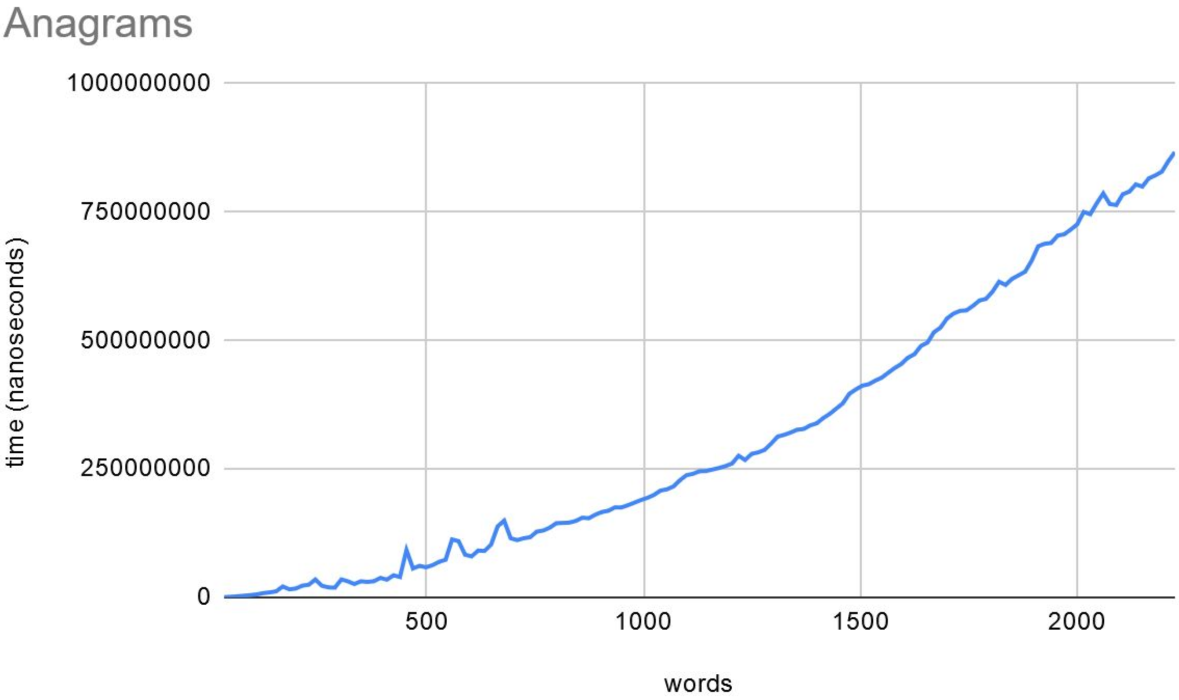
Josh Wells and Jonathan Oliveros
Prof. Swaroop Joshi
Assignment 04
Wed. 9/18/19

- Analyze the run-time performance of the areAnagrams method.

- What is the Big-O behavior and why?

  It should be O(n^2). The areAnagrams method performs the insertion sort on each word after performing an insertion sort on the letters of each word, therefore it can be written as $n_1{}^2 + n_2{}^2 + \ldots + n_p{}^2$ for $p \in Z$. Since we are only looking for the largest degree in Big-O notation, then the sum of all the n^2 does not matter and it is just O(n^2).

- Plot the running time for various problem sizes (up to you to choose problem sizes that sufficiently analyze the problem). (Note: You may want to randomly generate some strings for this or randomly choose a subset of the English Language file.)

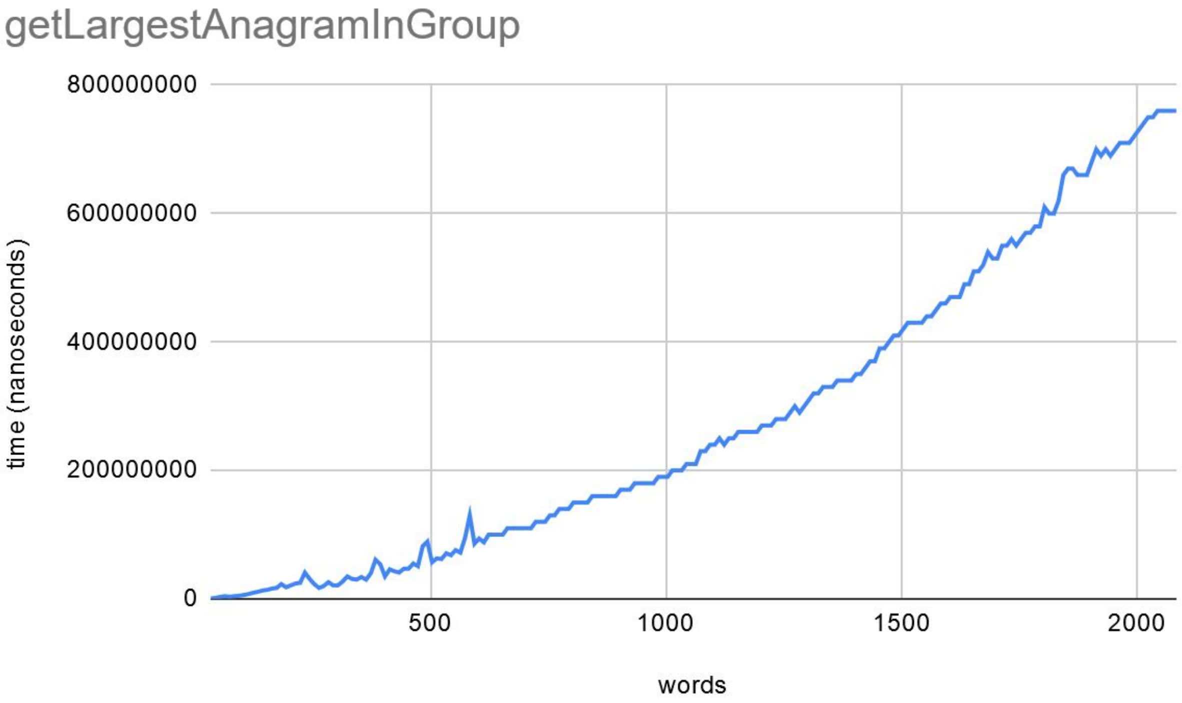


After each iteration of running Anagrams, 15 words was added to find out how much longer it takes to complete.

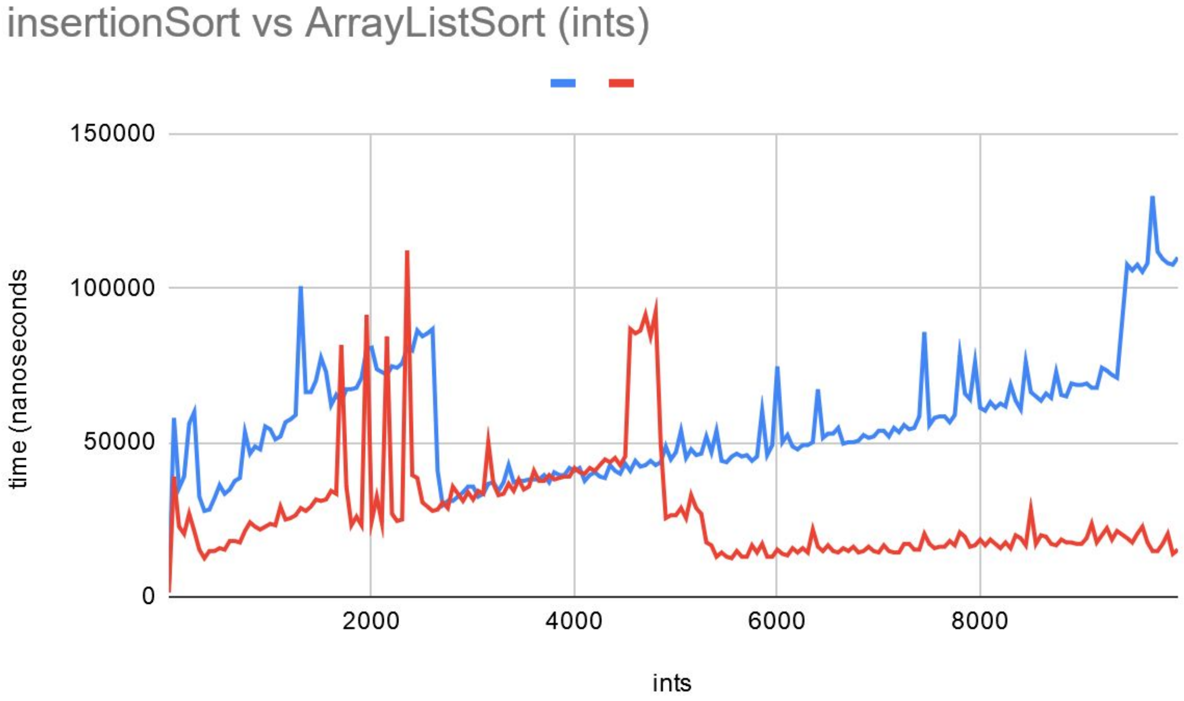
- Does the growth rate of the plotted running times match the Big-O behavior you predicted?

  The growth rate of the getLargestAnagramGroup method performed in a roughly n^2 time based on the number of words in the array. This could definitely be improved upon with a more linear algorithm. This was in line with our prediction.

- Analyze the run-time performance of the getLargestAnagramGroup method using your insertion sort algorithm. (Use the same list of guiding questions as above.) 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 (write a function to randomly build a string of characters). If you use the random word generator, use modest word lengths, such as 5-15 characters.

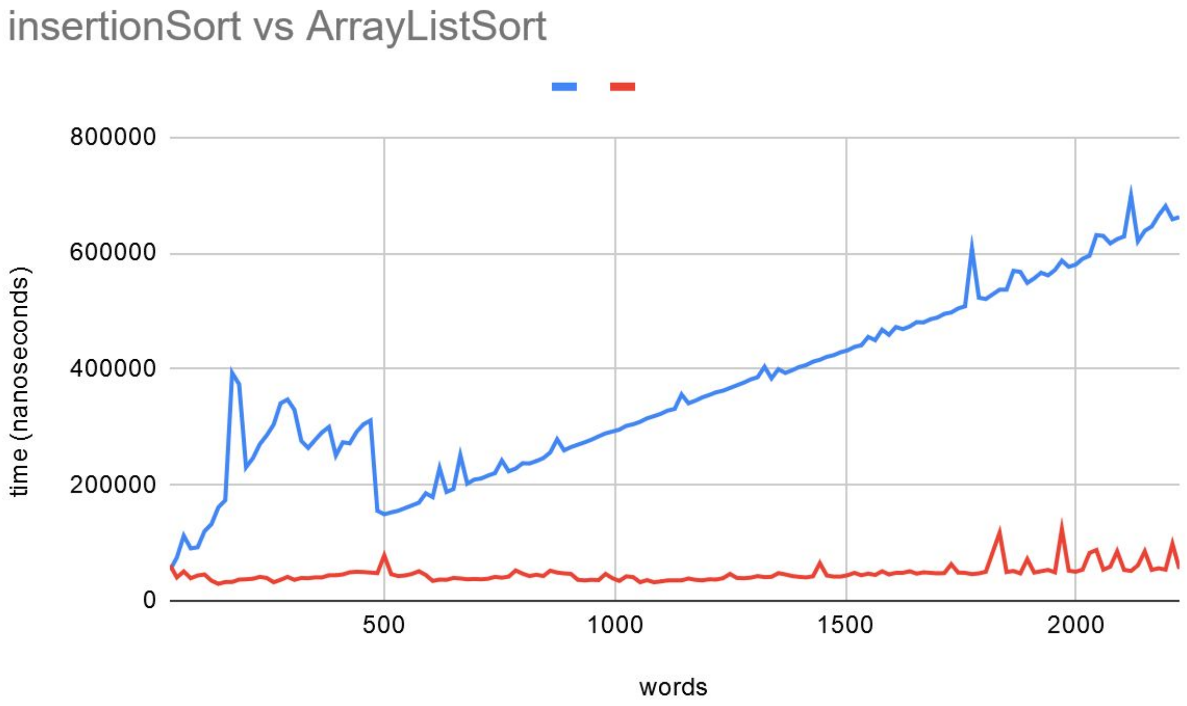


- Analyze the run-time performance of the insertionSort method using an array of strings and anArrayOfIntegers. Does the speed of computing match the speed of the getLargest Anagram Group? Explain why/why not.



Sorting integers was a much easier task than words. This is because integers are in base-10 whereas the alphabet has 26 letters, therefore there are less possibilities to choose from for integers.

- What is the run-time performance of the getLargestAnagramGroup method if we use Java collection framework’s built-in sort method instead of our own? How does it compare to using insertion sort? (Use the same list of guiding questions as above.)



Our insertion sort turns out to have O(n) behavior whereas using Java’s sort behaves more like O(1).