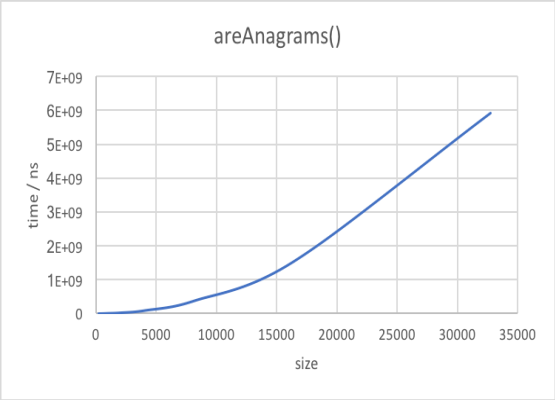
1. Analyze the run-time performance of the areAnagrams method.

-What is the Big-O behavior and why? Be sure to define N.

$O(N^2)$ as it calls the insertion sort, N is the length of the string

-Plot the running time for various problem sizes (up to you to choose problem sizes that sufficiently analyze the problem). (NOTE: Use the method you wrote for generating a random string of a certain length.)

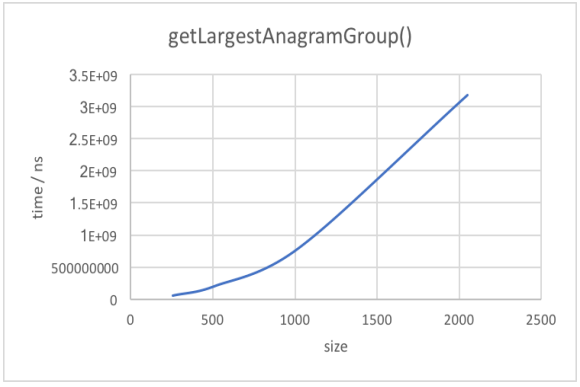
The plot is shown below.



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

Yes it matches as it is like the graph of $y = x^2$

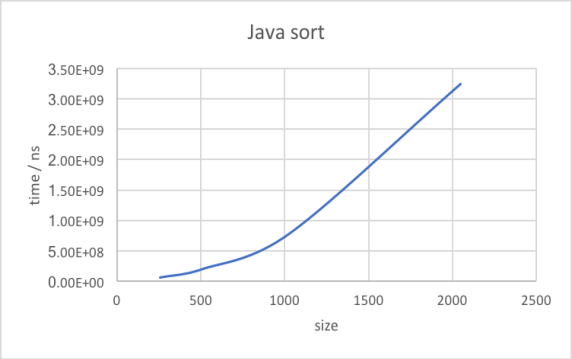
2. Analyze the run-time performance of the getLargestAnagramGroup method using your insertion sort algorithm. (Use the same list of guiding questions as in #1.) Note that in this case, N is the number of words, not the length of words. Finding the largest group of anagrams involves sorting the entire 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 your random word generator. If you use the random word generator, use a modest word length, such as 5-15 characters.



$O(N^2)$ because it uses the insertionSort().

3. What is the run-time performance of the getLargestAnagramGroup method if we use Java's sort method instead? How does it compare to using insertion sort? (Use the same list of guiding questions as in #1.)

The plot is shown below.



They are almost same complexity but the Java sort takes a little more time than insertionSort().