When you are satisfied that your program is correct, write a brief analysis document. Note that this document is worth 30 points, out of 100 points total for this assignment. Ensure that your analysis addresses the following.

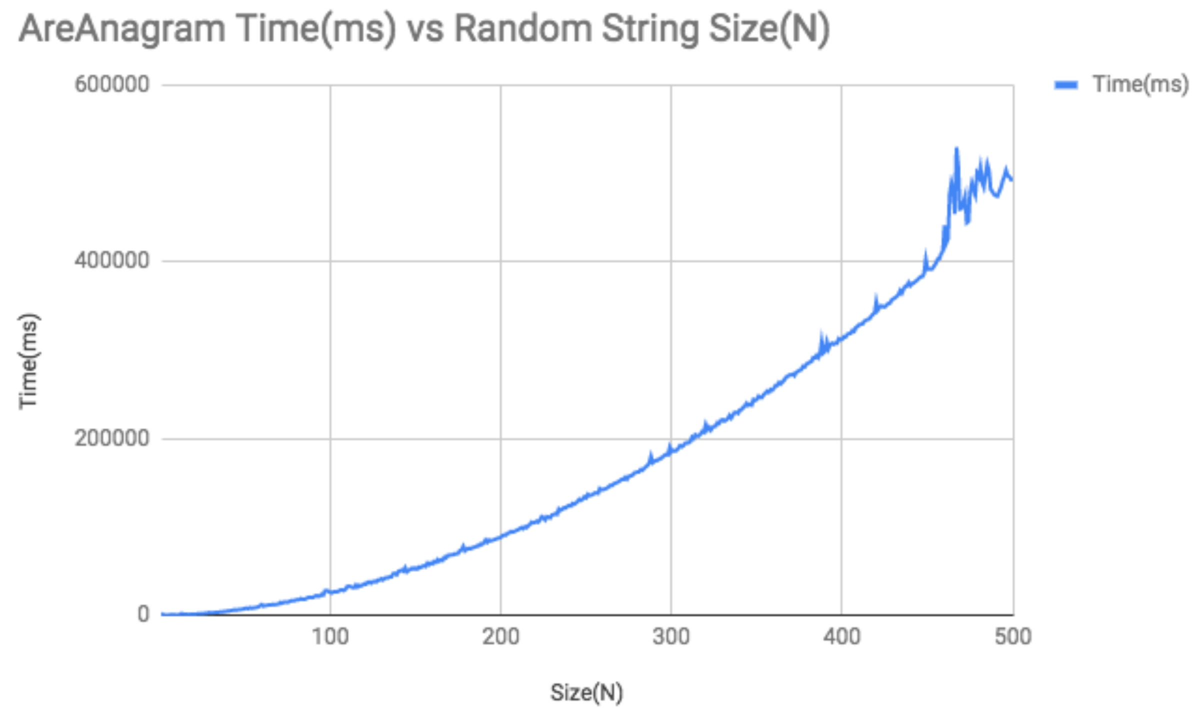
1. Analyze the run-time performance of the areAnagrams method.

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

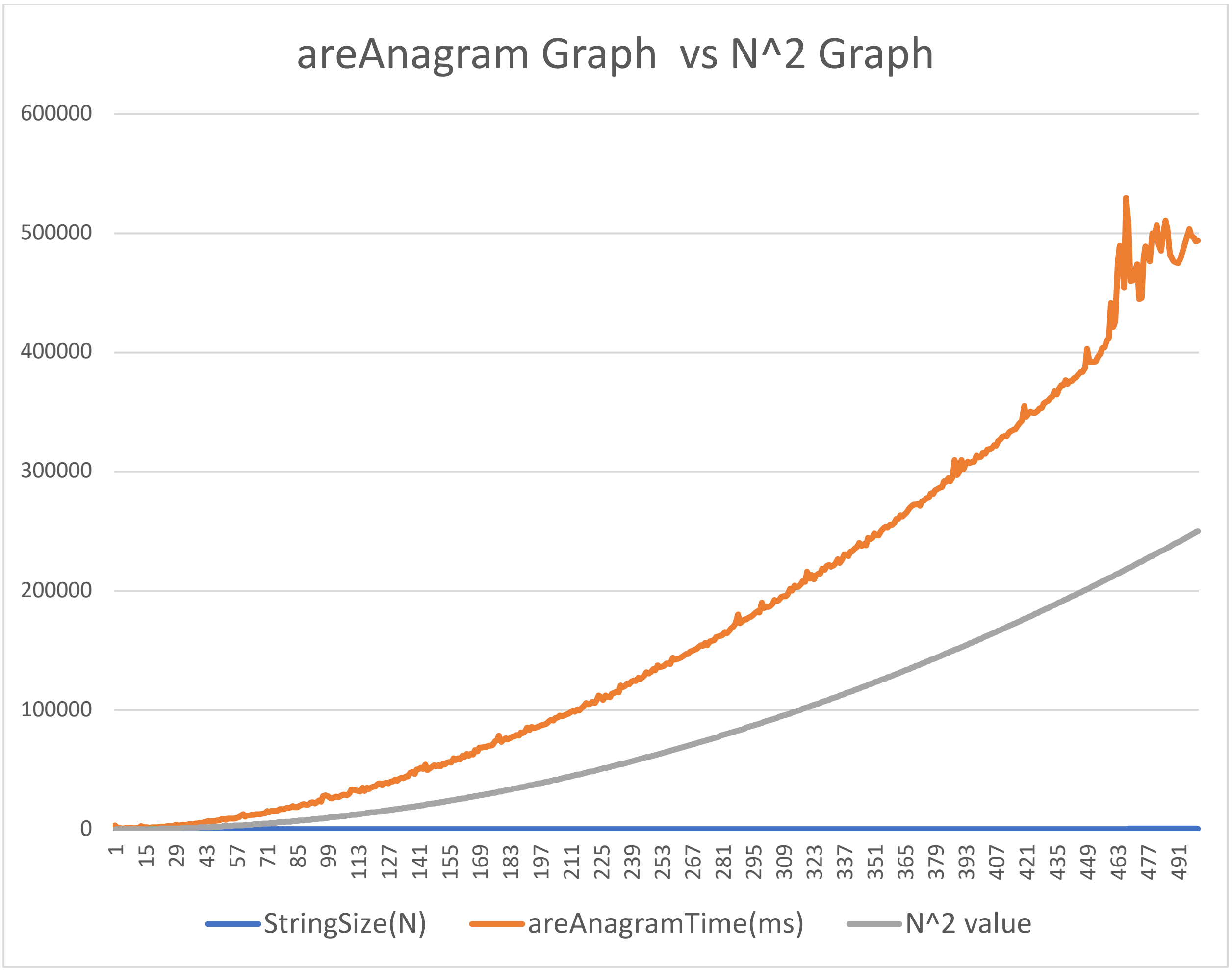
    Answer:

N refers to the input size, and here it is the length of the string input.
Big-O behavior is O(N^2). Because I called sort() method in areAnagrams to sort the two input strings. The sort() uses insertion sort of which the run time is quadratic to the input size N for worse and average cases. In terms of the actual comparing two string operation, it takes constant time to do it, based on the rule of asymptotic behavior, the overall complexity of this method is the dominating O(N^2).

    -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.)

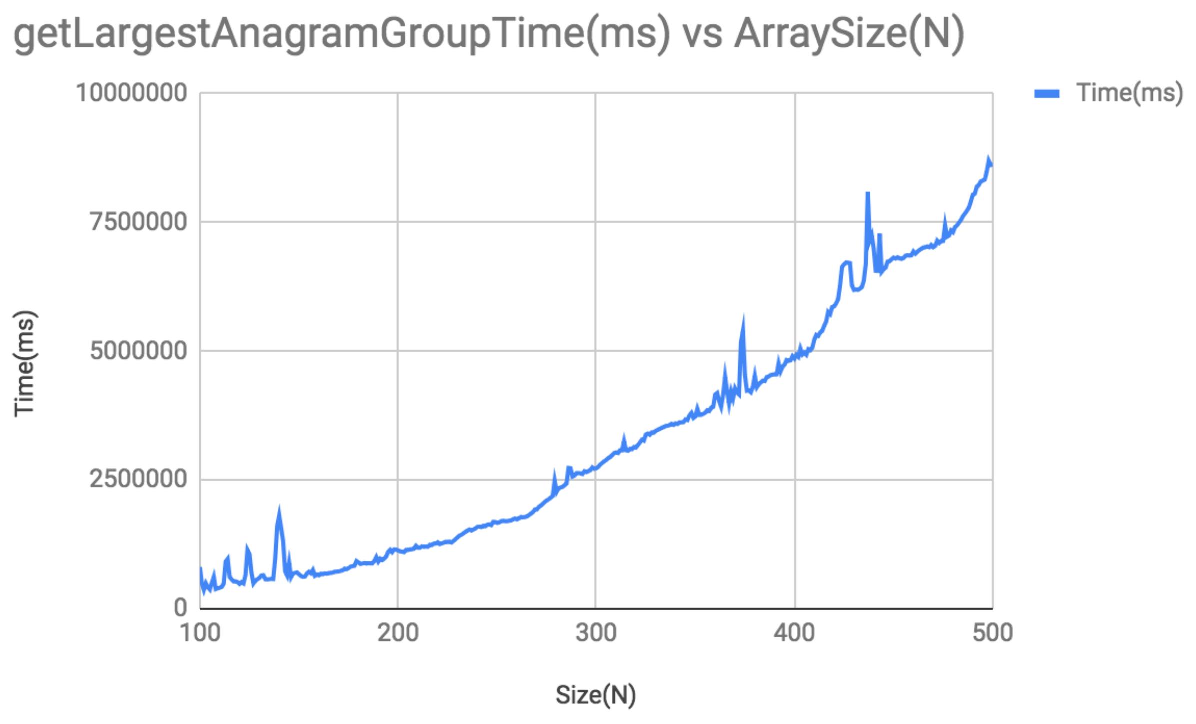


    -Does the growth rate of the plotted running times match the Big-O behavior you predicted?



Answer:
It does match my expectation of Big – O behavior of O(N^2). In the above graph, the orange line represents the data I have collected from areAnagrams timing experiment. The dot-line is what an N^2 graph looks like in the same independent variable range. Based on the graph, we can see that these two lines have similar growth rate. I think it is safe to say that the growth rate of areAnagram() method matches my expectation.

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.

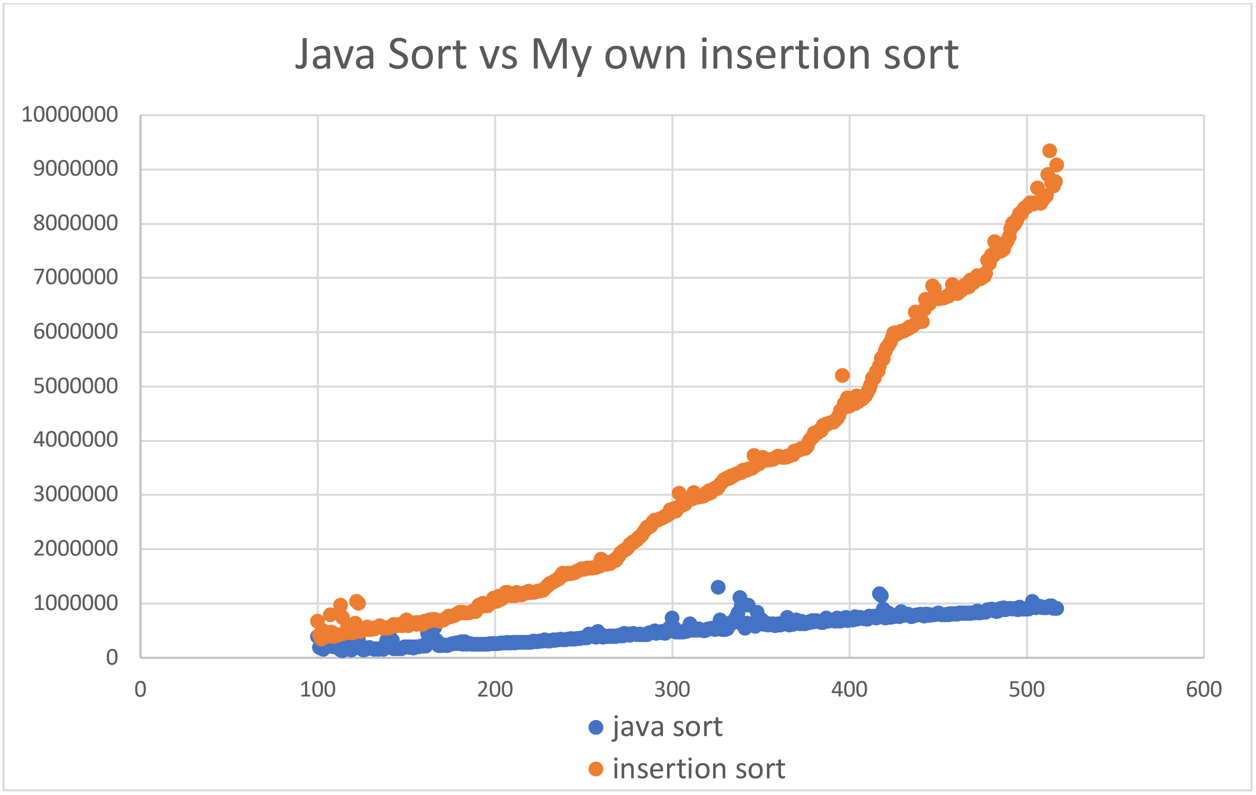


Answer:
The big-O behavior of getLargestAnagramGroup() using insertion sort method is O(N^2). N here refers to the size of the array we pass in to this method. In this method, there are two major implementations. The first is using insertion sort to sort the input array so that anagrams can be next to each other. Since the time complexity of insertion sort is O(N^2), this implementation has complexity of O(N^2). The second implementation is traversing the sorted version of array to get the largest anagram group. The way I implemented is that the outer loop initially traverses the array, the inner loop counts the adjacent anagrams and count the number then exits the inner loop, and outer loop skips the number of anagrams the inner loop just returned. So the complexity of this operation is O(N). so overall, O(N^2) dominates and the time complexity of this getLargestAnagramGroup() is O(N^2). Also the graph proves my Big-O expectation.

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.)

Answer:
According to Java Array API, the sort() uses mergesort for object that implements Comparable/Comparator interface, that the time complexity is O(NlogN) . My implementation of getting the adjacent anagram group is O(N). Based on asymptotic behavior, O(NlogN) dominates and the overall complexity of this method using Arrays.Sort() is O(NlogN). Below graph shows the comparison between my sort and java's sort method. Firstly, the blue lines matches the growth rate of a NlogN function graph. Secondly, it can be seen that

getLargestAnagram() using insertion sort is significantly less efficient than using merge sort with other parts of the code implementation remains constant.


Java Sort vs My own insertion sort
● java sort
● insertion sort