Assignment 4

I (Chase Yates) and Liam Astin worked on this assignment. We swapped in the middle of the assignment between driver and navigator. I submitted the code on Gradescope.

Question 1

We did better pair programming on this assignment than previous and got most of the work done in just over 2 hours. We swapped partway through between driver and navigator which probably made things more efficient. I might switch to someone new either on the next assignment or the one after that just to get a different experience and the mandatory semester switch out of the way, not because of Liam, however.

Question 2

The reason the `String sort(String string)` method does not have the signature `void sort(String string)` is because we want to, when comparing anagrams, keep our original String to return at the end, meanwhile getting a sorted copy to work with internally.

The reason for the `void insertionSort(T[], Comparator<? super T>)` not having the signature `T[] insertionSort(T[], Comparator<? super T>)` is because the order of the word array does not really matter for our purposes, and it would be more memory and time efficient to simply sort the array in place, rather than producing a sorted copy. The array provided is already essentially for internal use anyway.

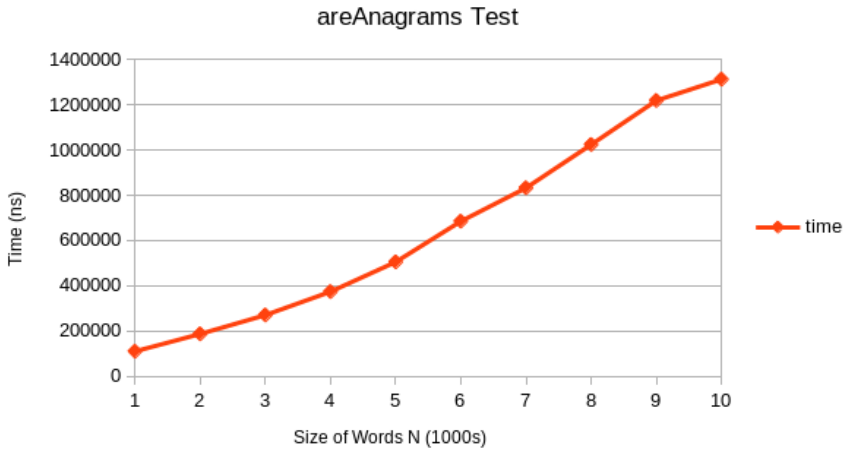
Question 3

Pre-plot Analysis

The Big O of the `areAnagrams` method is somewhat complex and there are several cases to consider. For this method, N is the size of the two given Strings The first case is that the lengths of the provided Strings are different, in which we immediately return false, as the two cannot be anagrams. This is O(1). The second best case is that the two Strings are empty. This is also O(1). If the two Strings are alphabetized, then the sorting method called on them, being insertion sort, is O(N), as this is the best case for insertion sort, iterating through every element and not moving them (the elements are characters in this instance). In the average/worst case, however, there is more to consider. In the method, we check if the lengths are the same, this is O(1), then we sort both Strings with insertion sort, being O(N^2) on average/worst cases. Then we compare the two Strings for equality, being O(N) as

we must check each character (I presume, not being totally sure about how Java runs its `.equals` method on Strings). The comes out to 2N^2 + N + 1, which is O(N^2).

Plot and Analysis



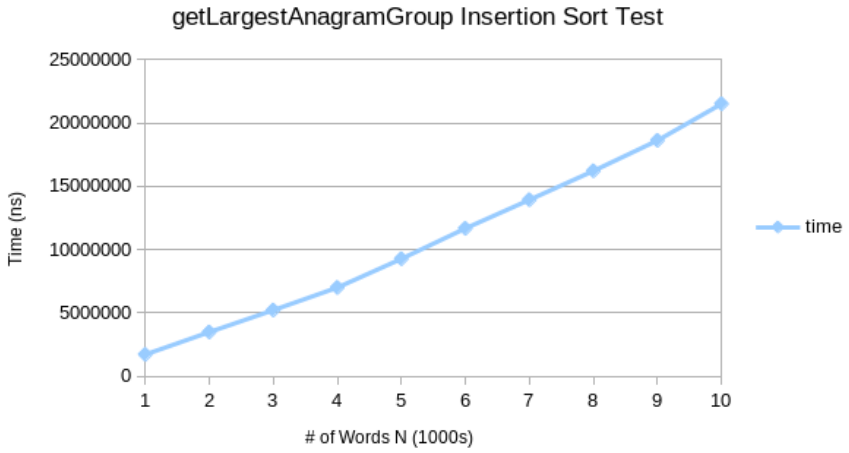
This plot of areAnagrams being run on random strings of size N shows a fairly clear quadratic curve, except for the last data point, lending credence to my theory of the algorithm being O(N^2).

Question 4

Pre-plot Analysis

Given N is the size of the array of Strings given, the `getLargestAnagramGroup` method is O(N^2) on average. In the method, we run an O(N^2) insertion sort on the list, then iterate over that list (O(N)), finding the largest group of anagrams, and do some constant time work at the end. There is a method call to `areAnagrams` within our O(N) loop, however this method is dependent on String length, not the N we are dealing with. The most dominant factor, and the average/worst case of this method, is O(N^2). In the best case, insertion sort is O(N), so the method is O(N).

Plot and Analysis



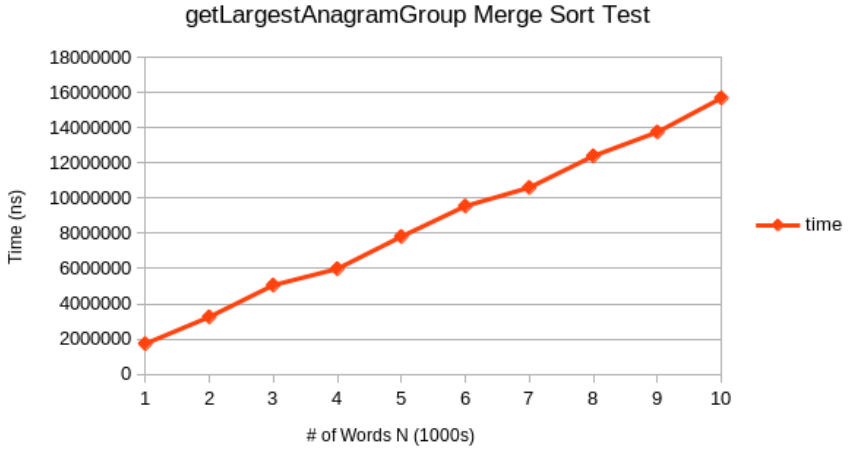
Although this graph is only quadratic if you squint at it, I imagine the low curvature of what should look quadratic is due to the nature of insertion sort being more efficient the more sorted something is. As the inputs were random, often times things will be somewhat sorted and other times definitely not sorted, which may give evidence for the low curvature on this graph. I would still argue that the presence of curvature shows O(N^2) is probably accurate.

Question 5

Pre-plot Analysis

Similar to Question 4, the `getLargestAnagramGroup` method using Java's sort method would look much different. Java's method uses a modified merge sort, which according to Java's own documentation, is O(NlogN). This would still be the new most dominant factor in our method, so I predict our method will be O(NlogN).

Plot and Analysis



This graph is much more linear, with some bumps along the way. This graph should be O(NlogN) however, which certainly looks fairly accurate to this graph. I imagine with much more testing time it would slowly look more and more like the predicted function, however this graph suffices to prove my prediction.