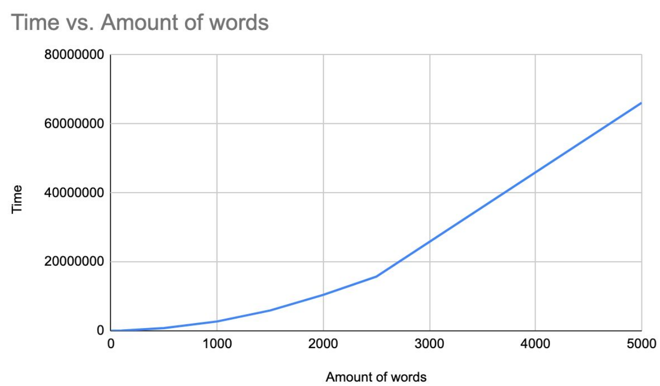
Assignment 04
09/18/2019

Alejandro Rubio
Alejandro Serrano
Prof. Swaroop Joshi

Analysis Document:

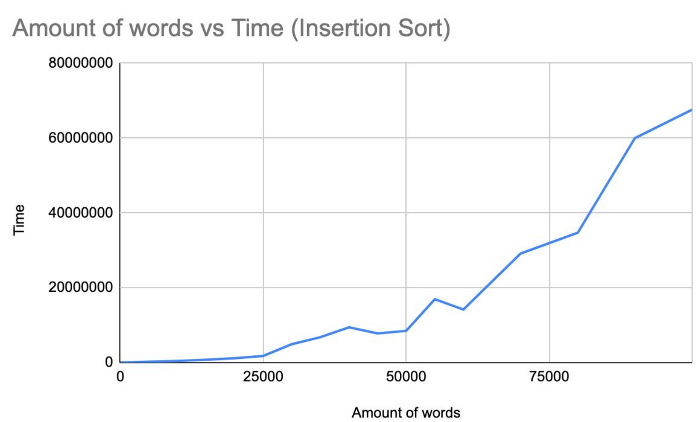
areAnagrams method

We believed that the graph is growing with a Big-O behavior of n^2. N being the amount of words on the list.



The Big-O behavior of the areAnagrams method matched our initial guess. The graph shows an n^2 complexity in respect of the amount of words tested. Our maximum amount of words tested was 5000 words. This was because if we used more than that, the program would have taken too long.

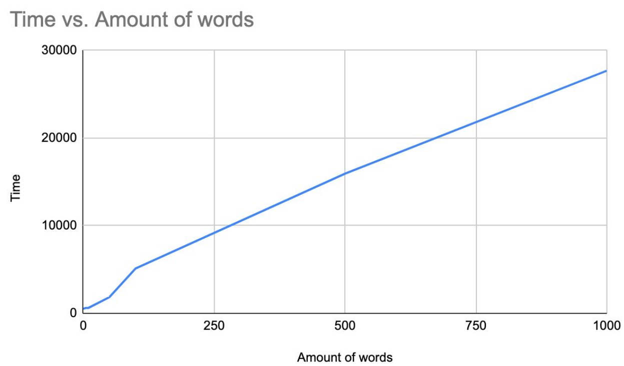
getLargestAnagramGroup method with insertion sort

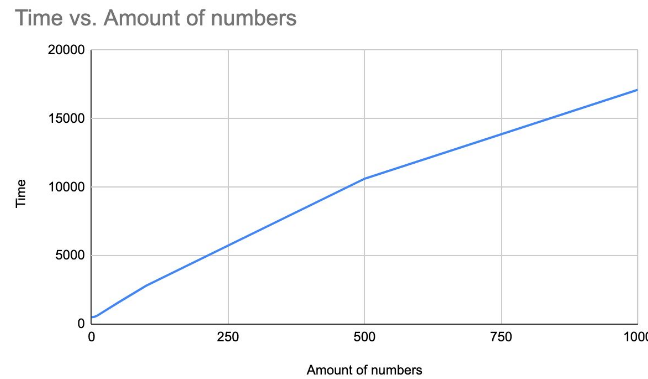


The Big-O behavior of the getLargestAnagramGroup method using insertion sort is also n^2. For this test, we used the words inside the files. We got longer times because we timed the entire program which calls all the other methods and not just one piece of code.

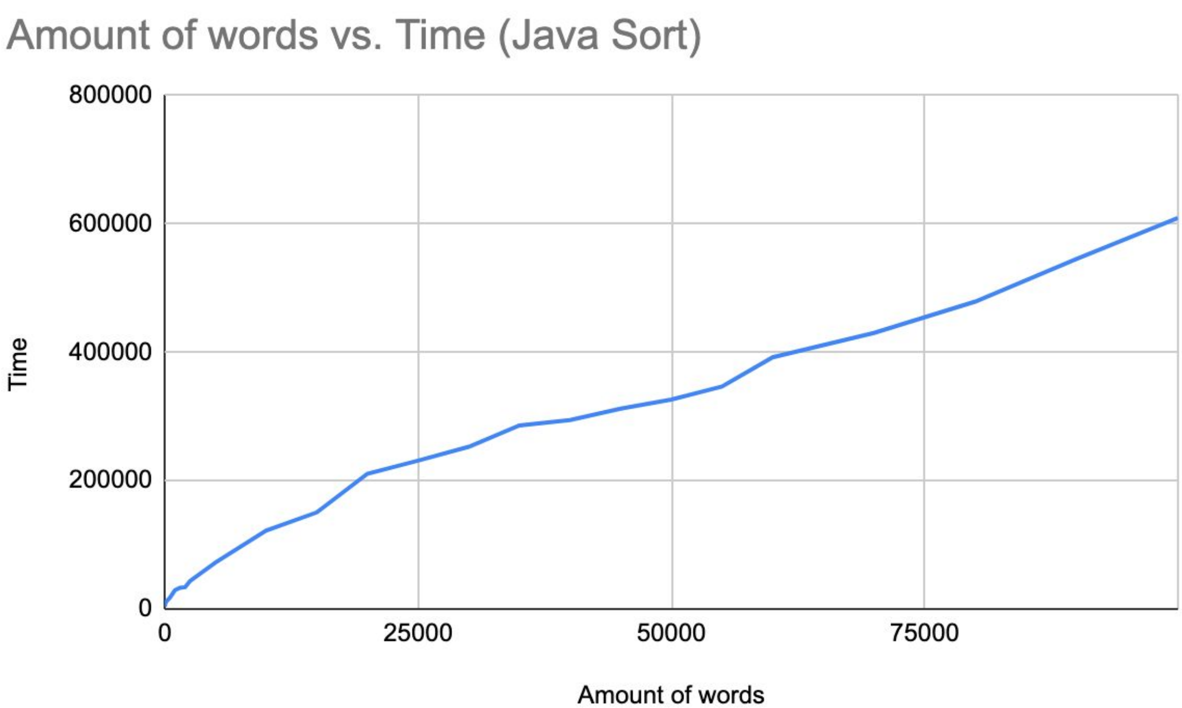
Insertion method

The Big-O behavior of the Insertion method using strings and integers was n^2. To test this, for the string we generated 1000 unsorted words and for the integers we created random integers between 0 and 2000. Even though both of them are n^2, we can see that insertion sort using numbers takes half of the time that it takes for strings. The only difference during testing was that we used the compareTo method for strings and Integer.compare for integers. We believe that sorting numbers was faster because when the algorithm is trying to sort strings there needs to be more comparisons than with integers.





getLargestAnagramGroup method with Java collection framework's built-in sort method



The Big-O behavior of the getLargestAnagramGroup with Java built-in sort method performed faster than when we used insertion sort. Both graphs have similar shapes because both tests are running the getLargestAnagramGroup algorithm. We expected this test to be a little faster because we knew that Java Sort is using merge sort.