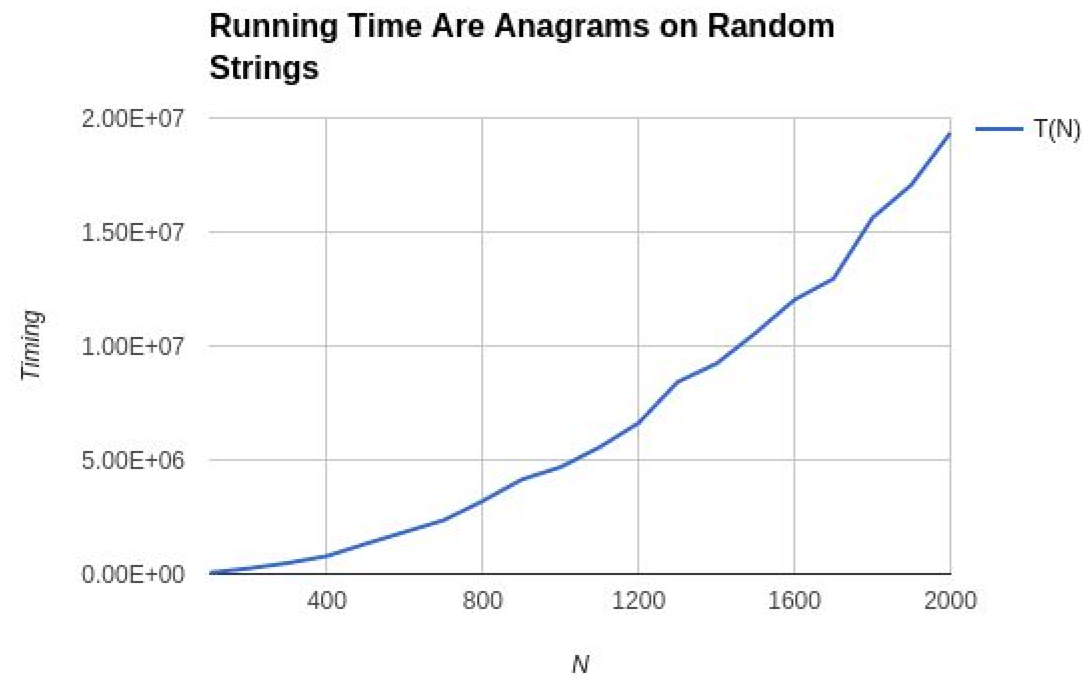
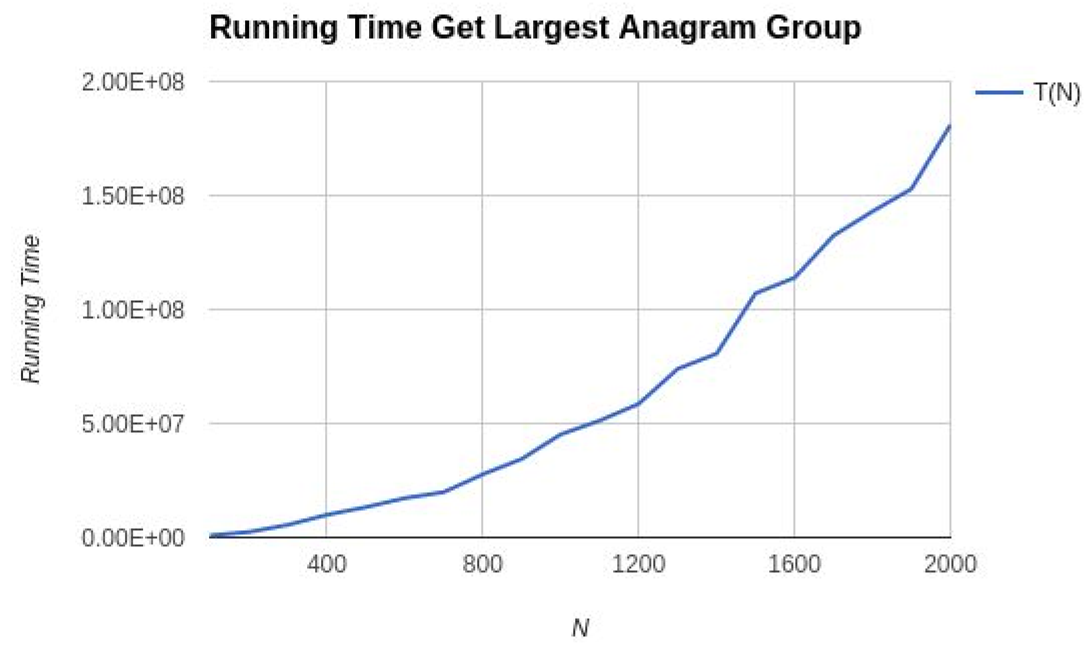
Diego Baez Lopez
U0759247
Assignment 04

My programing partner for this assignment was Brayden Carlson, I was the one to submit the code. Throughout the assignment we both learned a lot as we went, there were certain instances that I had some java knowledge from prior experience which helped streamline some looks ups. Working with one another was not bad, I was busier with work more than normal this time around so communication was limited on my end. Overall however there were no complaints on my end and I would work with Brayden again.



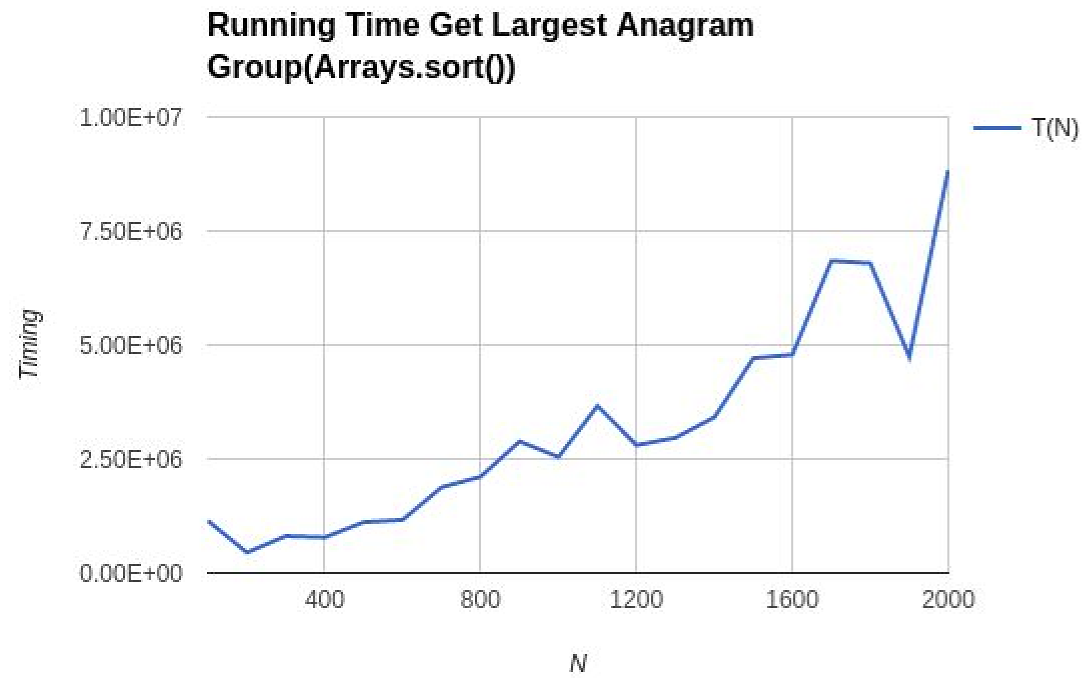
This is the running time for our areAnagrams function. The behavior is quadratic(O(n^2)) and N is 2000. The behavior is what we expected from this particular function because this function calls on sort which then calls on our insertion sort implementation. The average case and worst case for insertion sort is quadratic behavior. Using the table below which calculates the values for N^2 behavior we can see how the numbers converge on this behavior as N increases. Once the problem “warms up” the timing becomes more and more consistently quadratic. Given what has been presented to us about insertion sort this would align with what sort of behavior I predicted.

100	5.69E+00	500	5.30E+00	900	5.12E+00	1300	4.98E+00	1700	4.48E+00
200	6.15E+00	600	5.12E+00	1000	4.69E+00	1400	4.71E+00	1800	4.82E+00
300	5.35E+00	700	4.81E+00	1100	4.60E+00	1500	4.70E+00	1900	4.73E+00
400	4.91E+00	800	5.01E+00	1200	4.60E+00	1600	4.70E+00	2000	4.84E+00



This is a plot for the running time of getLargestAnagram function. The behavior is quadratic as well(O(n^2)) n is also 2k. This behavior is also expected as we know that insertion sort is quadratic on average but also because that portion of the code is the most exhaustive. The table below shows the timing calculation for N^2 and confirms this from our testing code as well. As the N increases the numbers converge and become more quadratic as N increases. Words of length zero through fifteen were used.

100	8.91E+01	500	5.32E+01	900	4.25E+01	1300	4.38E+01	1700	4.58E+01
200	6.03E+01	600	4.79E+01	1000	4.52E+01	1400	4.11E+01	1800	4.41E+01
300	6.16E+01	700	4.06E+01	1100	4.24E+01	1500	4.76E+01	1900	4.24E+01
400	6.22E+01	800	4.33E+01	1200	4.07E+01	1600	4.45E+01	2000	4.52E+01



This is the plot of the running time of getLargestAnagram using Java's Arrays.sort() function. It's big-oh behavior is O(n log n) again where n is 2k. Again here this behavior is expected because this sort uses merge sort. The behavior for merge sort is logarithmic and Java's sort method is also much more efficient than our implementation. The table below calculates the timing according to N logN calculations. As N gets larger the behavior converges and confirms the logarithmic behavior. This is better than our implementation which only relies on insertion sort.

100	1.72E+03	500	2.49E+02	900	3.27E+02	1300	2.21E+02	1700	3.75E+02
200	2.94E+02	600	2.10E+02	1000	2.55E+02	1400	2.33E+02	1800	3.49E+02
300	3.28E+02	700	2.84E+02	1100	3.30E+02	1500	2.98E+02	1900	2.29E+02
400	2.26E+02	800	2.74E+02	1200	2.28E+02	1600	2.81E+02	2000	4.03E+02

All in all this assignment took about twenty hours.