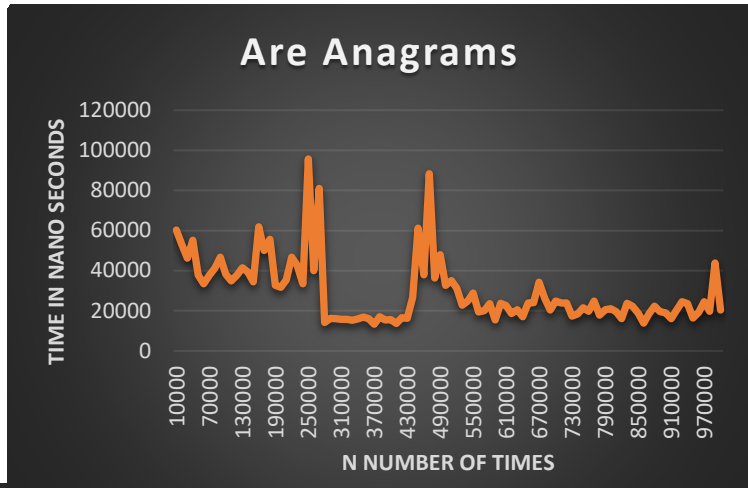
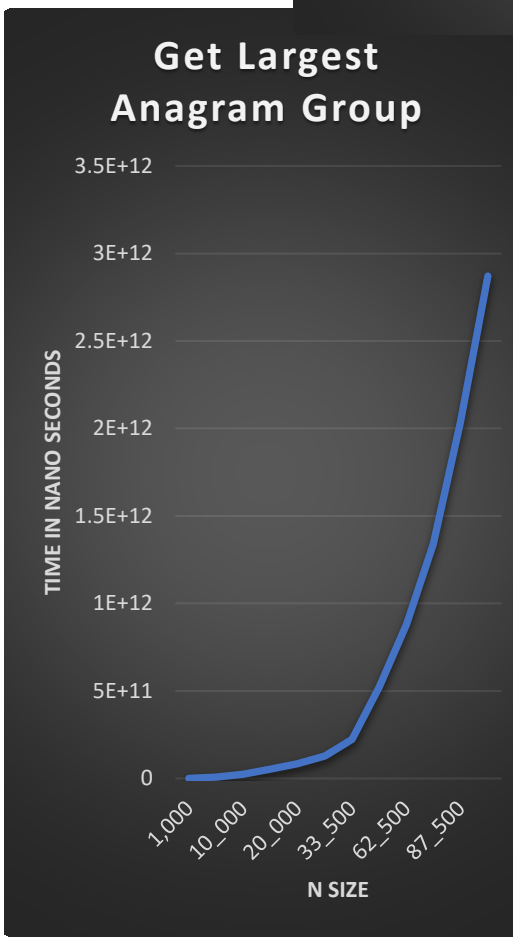
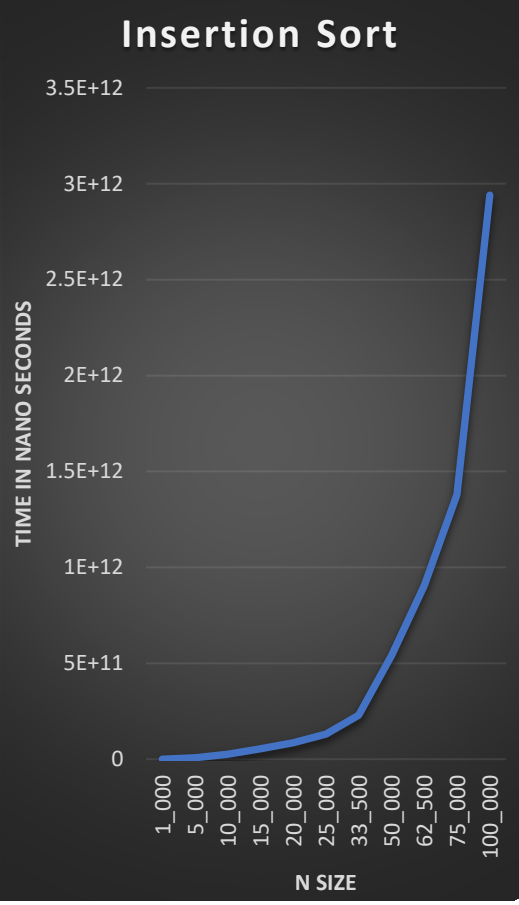
Harrison Smith and Sam Hancock
Prof Swaroop Joshi
Assignment 04 – Anagrams
Wed. September 18, 2019

1. The Run time for our areAnagrams, has many spikes, we believe this is due because we are calling sort within our areAnagrams method.
2. The Big O for areAnagrams ought be N^2 (length of word) because we are calling sort every time on the String elements, we call the areAnagrams method
3. The actual big o for our areAnagrams method appeared to model a C^-n type relationship with irregular spikes in the middle. This confuses and vexes us since as the array length gets longer, so should the execution time.
4. The Big O for getLargestAnagramGroup should be N^2 for the N's within the list being checked. This is because every time we sort, we have to touch each item for N times. Our run time for this well resembled this function, therefore preforming as expected.
5. The Big O for insertionSort should be N^2 for the N's within the list being sorted, due to checking each N every time we touch an N. This also performed as expected
6. The two Big O's for getLargestAnagramGroup and insertionSort are identically growing to N^2, because each time we check either of them, we will have to sort and check at each N for the entire list resulting in a quadratic growth. getLargestAnagramGroup will take a slightly longer amount of time to compute, because then we check at each N to find the largest group of anagrams after sorting. However, since it calls insertionSort we will still have the same runtime.
7. If we use Java's sorting method instead of our insertionSort method it preforms at about the Big O of N, this quickly beats our timing for our sorting method (see next page for graph)







Harrison Smith and Sam Hancock
Prof Swaroop Joshi
Assignment 04 – Anagrams
Wed. September 18, 2019

8.