Insertion Sort: Our expectations were that it would take a reasonable amount of time to run given that the input text file was not in order and sorts of elements that are right next to each other. For half the size insertion sort ran for ~3.9 min and for the full size it ran for approximately 15 minutes so the time more than doubled when doubling the size of n. As n became larger its complexity was closer to the worst case.

Comb Sort: For comb sort, the time to sort approximately doubled when doubling the size n which we partially expected, since the comb sort method sorts in step sizes we thought it would be more efficient than insertion sort. Comb sort did not match its worst complexity of O(n^2), which we expected because it was noted as one of the more efficient sorting methods.

Bubble Sort: For the full size of n the time it took to sort more than doubled from half of n, which closely matches its O(n^2) complexity. This sorting method took the longest time, which we expected.

Merge Sort: Our expectations of merge sort were that it would not take very long which we observed when running it. When doubling the size of n the time approximately doubled and did not match the worst case complexity for larger n.

Counting Sort: Our expectations were that this would be quick, but we ran into issues with memory allocation because of the large dynamic array being created within the algorithm. To solve this issue we sorted the negative numbers in the array using quicksort and large positive numbers greater than 20,000,000 also using quicksort because counting sort typically works better for smaller numbers. Still, using counting sort and quick sort in one the algorithm did not match its worst-case complexity.

Quick Sort: Quick sort was very quick as we expected and was the fastest sorting algorithm used. For the full size of n compared to the half size of n, the time more than doubled than n but still was very quick and did not match its worst complexity of O(n^2).