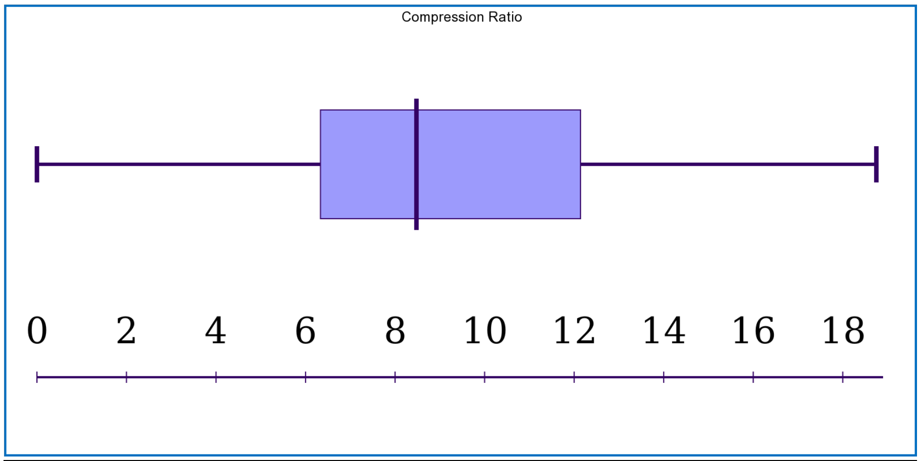
Experiment Procedure:

I conducted these experiments by testing the encode and decode programs on various test cases. I used the test files provided to encode it with both insertion and merge sort, then decoded the output file, again using insertion and merge sort. I attempted to have a variety of characteristics with the test files chosen, such as varying lengths, new lines, and special characters. The shortest input was 3 lines, with the longest being 280 lines.

For time complexity, I am assuming each line of code takes constant time and disregarding the constant value *c* in the functions

Experiments:

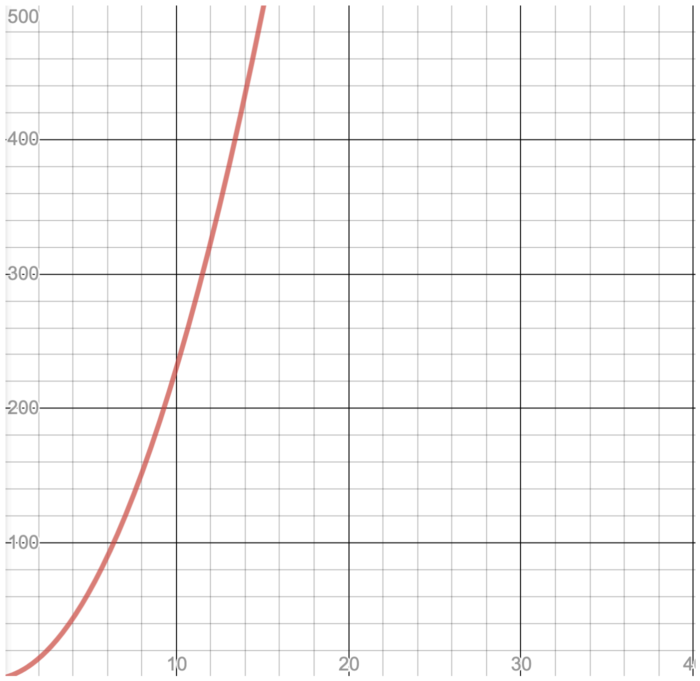
1) Average Compression Ratio:
   a. I took plot points from across the entirety of varying input files and computed it using (t-c)/t * 100, where t is the number of characters after encoding and c is the number of clusters. I used excel to plot and compute
   b.



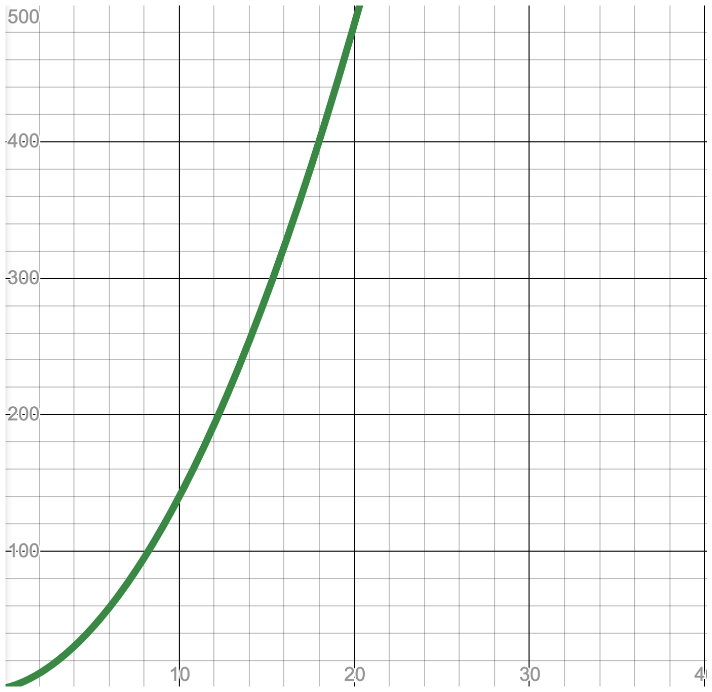
Standard deviation- 4.84
Average- 8.8
Min- 0

Max- 18

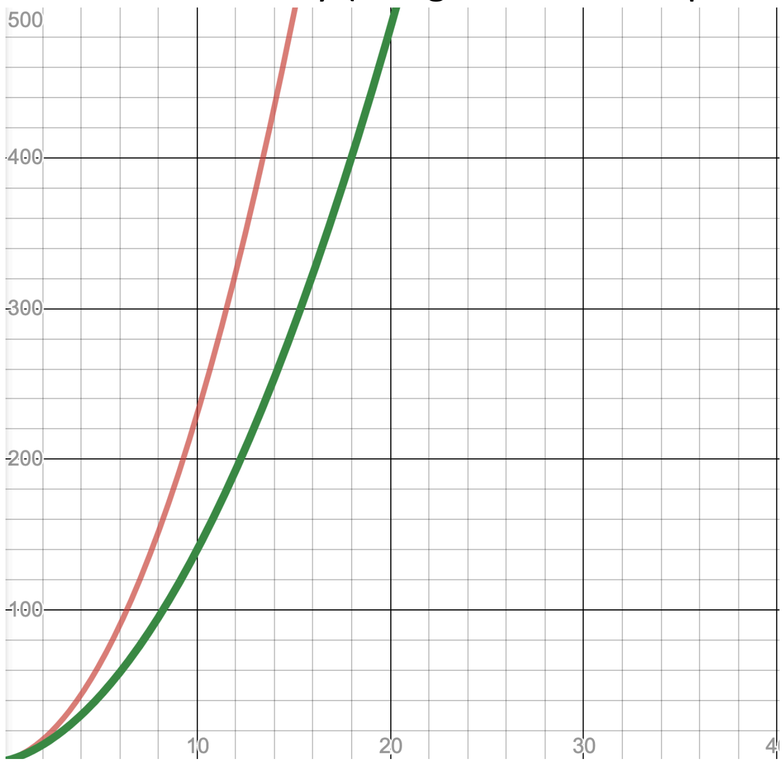
2) Encoding Time Complexity:
   a. For insertion sort, an input of size n would take n + 2n^2 (pictured in red below) or O(n^2).



   b. For merge sort, an input of size n would take 3n + n^2 + nlogn (pictured in green below) or O(n^2)

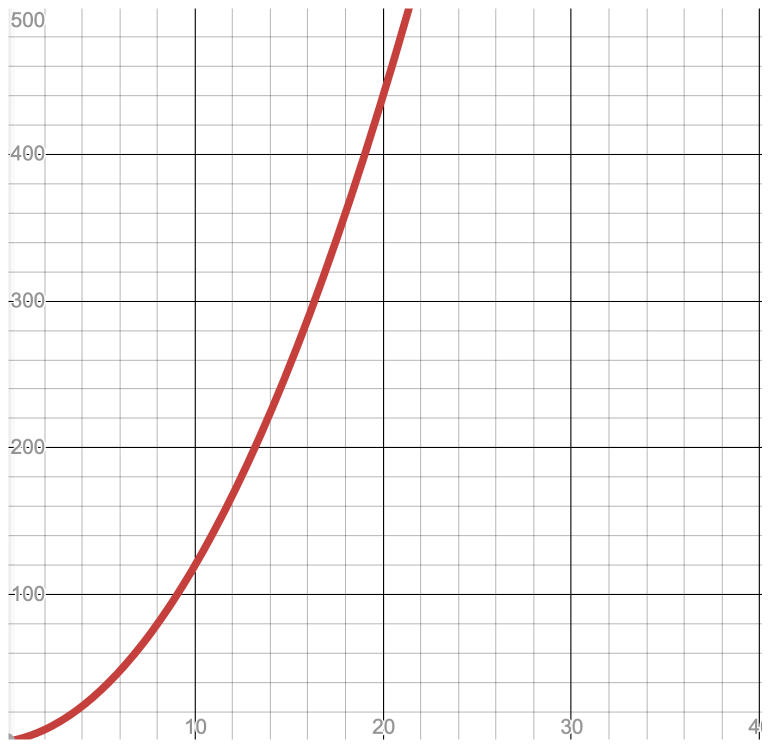


c. The merge and insertion sort did not affect my big O notation because the algorithm I incorporated to do the cyclic shifts was on the order of n^2. I most likely could have made this algorithm more efficient in order to have a better run-time efficiency (merge vs insertion pictured below)

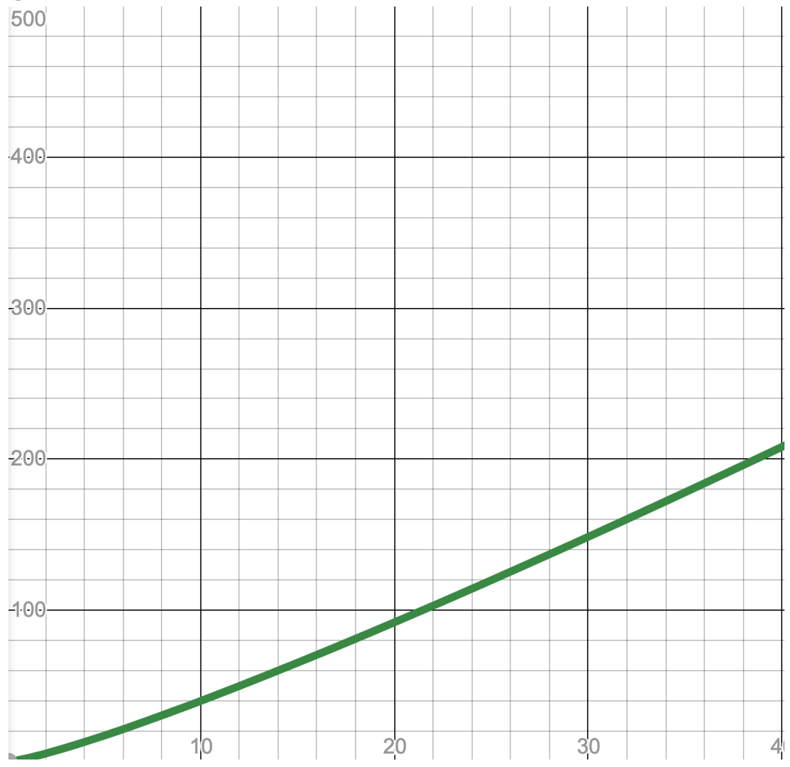


3) Decoding Time Complexity:

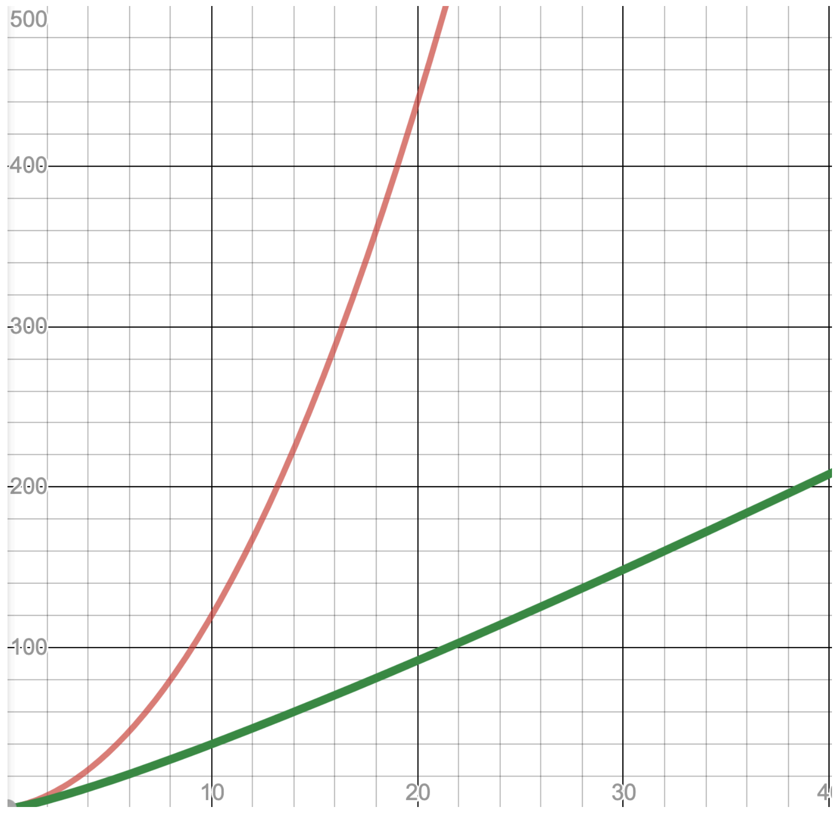
a. For insertion sort, the time complexity is n^2 + nlogn, or O(n^2) (pictured in red below)



b. For merge sort, the time complexity is 2n + 2nlogn, or O(nlogn). This is by far the most efficient of the 4(pictured in green below)



c. The merge sort was an order of magnitude faster for the decoding program, with a big O of only nlogn. (merge vs insertion pictured below)



4) If the program took in multiple lines at a time, I would partially expect the compression ratio to increase. The information being compressed is not sorted, since it is the last letter of the sorted array, not the first (which is what it is being sorted by). So, increasing the size of input doesn't directly correlate to a higher compression. However, an increased input would lead to a greater chance of higher clusters, which could increase the rate. So it is a loose correlation. I am basing this off of the data I collected in question 1, where my maximum was a large input size.