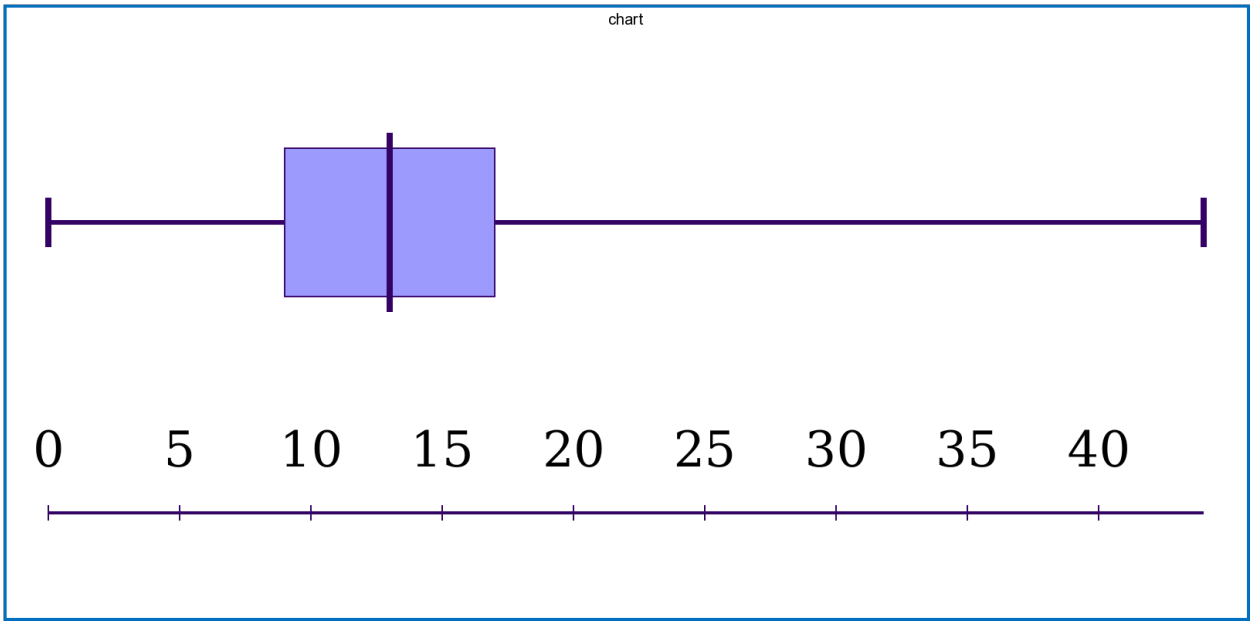
Experimentation

The next few pages will be the graphs measuring the compression ratio of encode, and measuring the time each quick/insertion for encode/decode. I have decided to use the file: wizard-of-oz.txt given by the professor for testing our encode and decode outputs. I use this txt file because it has 3473 total lines that have data(not including empty lines). I feel the more data we analyze, the better understanding we have when determining the average compression and time used to run the program with wizard-of-oz as an input.

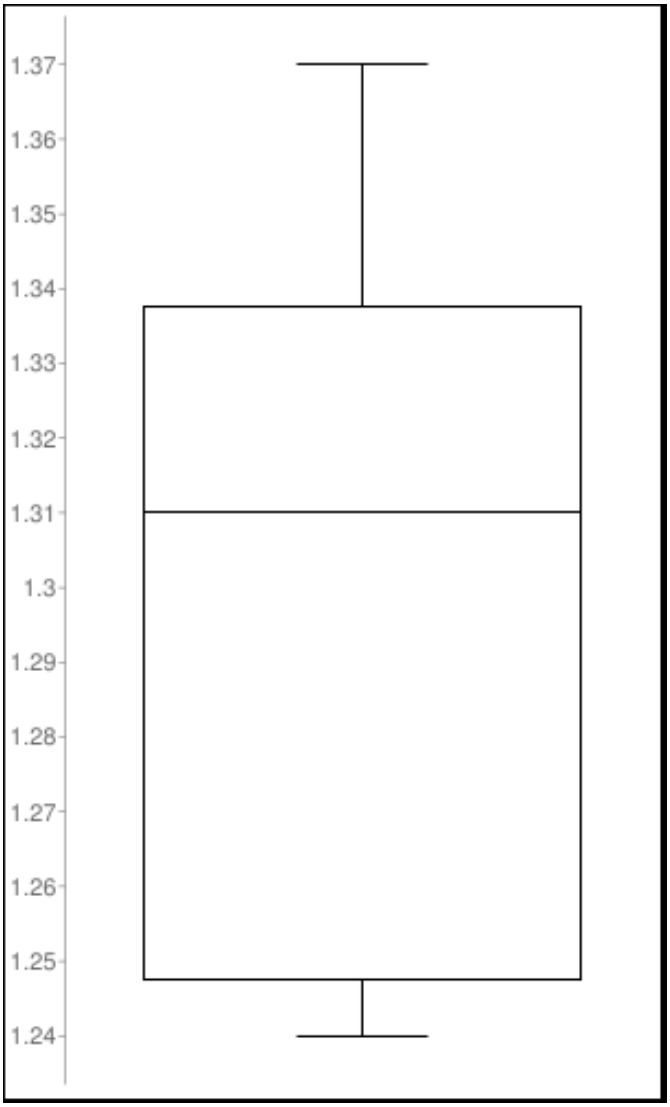
Compression ratio(CR):



Our total data is 3473 separate ratios. Out of the 3473 data points:
Our max CR ≅ 44%
Our Min CR ≅ 0%
Our first quartile ≅ 9%
Our second quartile ≅ 13%
Our third quartile ≅ 17%
Our average ≅ 12%

Encode using insertion:

In the encode experiment, I used the same wizard-of-oz.txt file, and ran it 10 separate times.



This chart shows our time(in seconds) to run wizard-of-oz.txt using insertion.

Our total data is 10 separate times. Out of the 10 data points:

Our max time ≅ 1.37s

Our Min time ≅ 1.24s

Our first quartile ≅ 1.25s

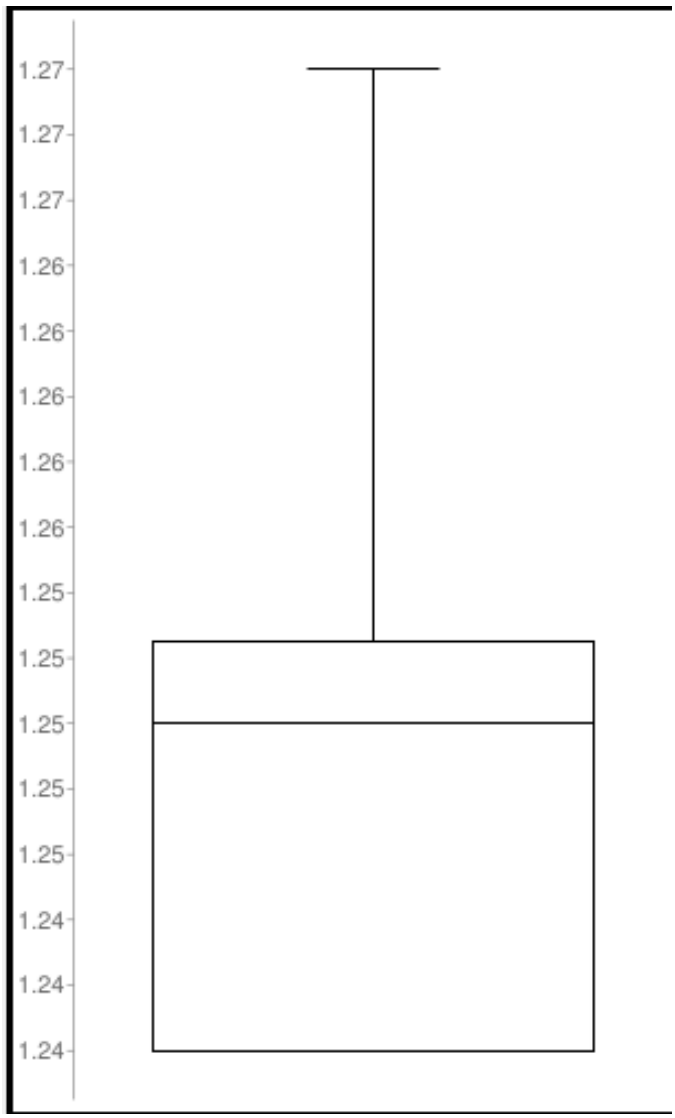
Our second quartile ≅ 1.31s

Our third quartile ≅ 1.34s

Our average ≅ 1.3s

Encode using quick:

In the encode experiment, I used the same wizard-of-oz.txt file, and ran it 10 separate times.



This chart shows our time(in seconds) to run wizard-of-oz.txt using quick.

Our total data is 10 separate times. Out of the 10 data points:

Our max time ≅ 1.27s

Our Min time ≅ 1.24s

Our first quartile ≅ 1.24s

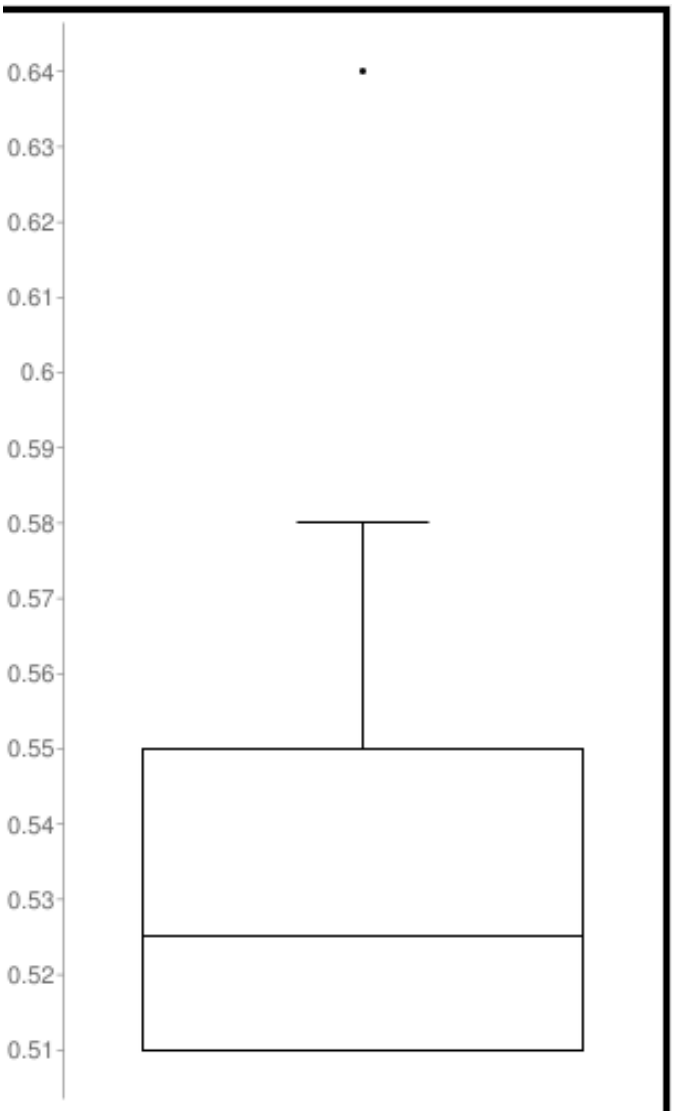
Our second quartile ≅ 1.25s

Our third quartile ≅ 1.25s

Our average ≅ 1.25s

Decode using insertion:

In the decode experiment, I used the same wizard-of-oz.txt file, and ran it 10 separate times.



This chart shows our time(in seconds) to run wizard-of-oz.txt using insertion.

Our total data is 10 separate times. Out of the 10 data points:

Our max time ≅ 0.64s

Our Min time ≅ 0.51s

Our first quartile ≅ 0.51s

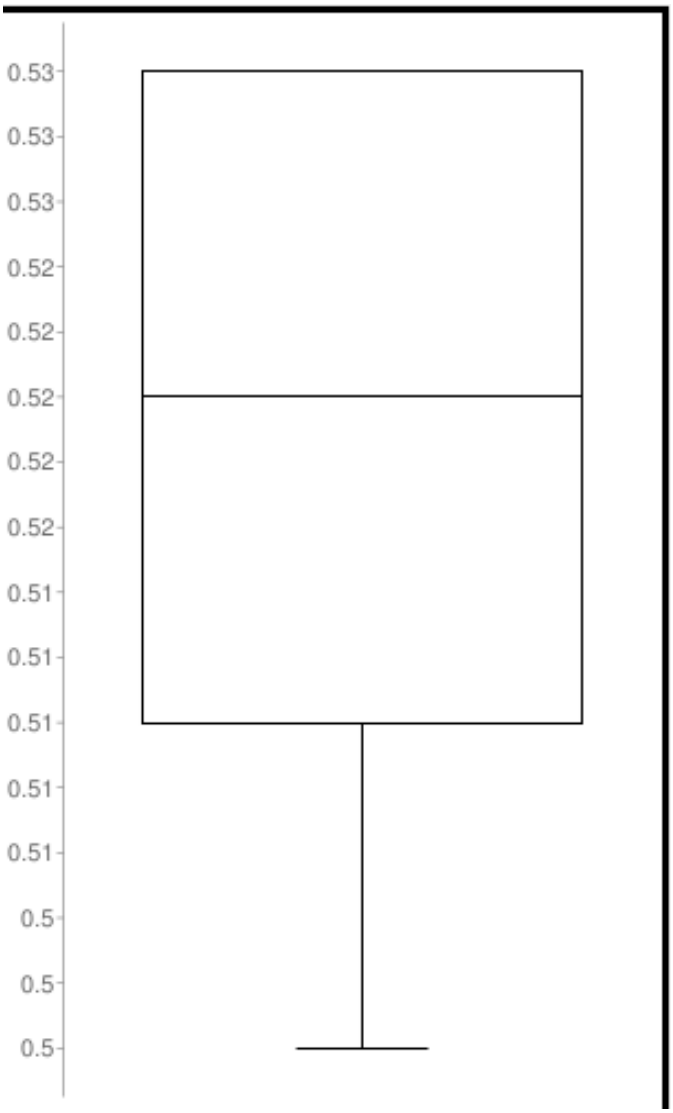
Our second quartile ≅ 0.53s

Our third quartile ≅ 0.55s

Our average ≅ 0.54s

Decode using quick:

In the encode experiment, I used the same wizard-of-oz.txt file, and ran it 10 separate times.



This chart shows our time(in seconds) to run wizard-of-oz.txt using quick.

Our total data is 10 separate times. Out of the 10 data points:

Our max time ≅ 0..53s

Our Min time ≅ 0.5s

Our first quartile ≅ 0.51s

Our second quartile ≅ 0.52s

Our third quartile ≅ 0.53s

Our average ≅ 0.52s

Experimentation of encoding more than one line at a time

Compression ratio related to the amount of lines being encoded at one time. I believe the more lines we encode, the higher the compression ratio. I believe this because as we go through one line at a time, most letters are grouped together. If we continue with compressing more than one line at a time, I feel there's a chance for additional letters to be grouped with the ones before it. If more letters are grouped together, then the compression ratio should increase. The total amount of characters doesn't change, but the cluster amount is suspectable to change , and there's possibilities of clusters of letters to be merged.

From experimentation, I have concluded that my hypothesis is correct. The compression ratio slightly increases when reading three lines at a time. I tested by reading and compressing 3 lines at a time for the test input cases given. While compressing three lines, I have noticed a slight increase in average compression ratio by roughly 2-3%. From these results, I infer that this means the more lines we encode at one time, the higher the compression ratio. The higher the compression ratio, the more space we save when transferring data.

The equation for compression ratio for encoding 1 line at a time is

$$CR(1) = \frac{t-c}{c} \times 100.$$

The compression ratio for encoding n lines at a time would be:

$$CR(n) = \frac{(t_1+t_2+t_3+\cdots+t_n)-c}{c} \times 100.$$

The total number of characters are changed by how many how many lines we encode at a time. The clusters are affected after the sum of all characters being encoded at one time.

With running wizard-of-oz.txt as my testing input and analyzing results, I have also concluded that quicksort is a more efficient sorting algorithm than insertionsort. This backs up the time complexity of quicksort(O(nlogn)) is faster than insertionsort(O(n)).