Terminology and Required Formulas:

1. CompressionRatio = $\dfrac{[log2n] - ExpectedNumberBits}{[log2n]} * 100\%$

2. ExpectedNumberBits = ∑(freq /n) * LengthInBits(encoding)

Experiment 1:

Data:
A file that big?
It might be very useful.
But now it is gone.

Computations:

ExpectedNumberBits

= (3/25)*3 + (11/25)*2 + (2/25)*4 + (1/25)*4 + (1/25)*6 + (1/25)*6 + (1/25)*6 + (1/25)*6 + (2/25)*6 + (5/25)*4 + (2/25)*6 + (3/25)*5 + (2/25)*6 + (5/25)*4 + (2/25)*6 + (1/25)*6 + (2/25)*6 + (2/25)*6 + (1/25)*6 + (2/25)*6 + (6/25)*4 + (3/25)*5 + (1/25)*6 + (1/25)*6 + (1/25)*6

= 0.36 + 0.88 + 0.32 + 0.16 + 0.24 + 0.24 + 0.24 + 0.24 + 0.48 + 0.8 + 0.48 + 0.6 + 0.48 + 0.8 + 0.48 + 0.24 + 0.48 + 0.48 + 0.24 + 0.48 + 0.96 + 0.6 + 0.24 + 0.24 +0.24

= 11

CompressionRatio

= $\dfrac{log2[25] - 11}{log2[25]} * 100\%$

= $\dfrac{5 - 11}{5} * 100\%$ = -1.2

Experiment 2:

Data:
She sells sea shells by the sea shore.

Computations:

ExpectedNumberBits = ∑(freq /n) * LengthInBits(encoding)

= (1/9)*3+(3/9)*2 +(1/9)*3+(1/9)*5+(1/9)*5 +(4/9)*3 +(2/9)*4 +(4/9)*3 +(5/9)*3

= 7.666666666667
= 8

CompressionRatio = $\frac{4-8}{4} * 100\%$ = -1

Experiment 3:

Data:
ABORTED effort:
Close all that you have.
You ask way too much.

Computations:

ExpectedNumberBits

= (3/29)*3 + (9/29)* 2 + (2/29)*4 + (1/29) * 4 + (1/29)*7 + (1/29)* 7 +(1/29)*7 + (1/29)*7 + (1/29)*7 + (1/29)*7 + (1/29)*7 + (1/29)*7 + (1/29)*6 + (5/29)*4 + (1/29)*6 + (3/29)*5 + (2/29)*6 + (3/29)*5 + (1/29)*6 + (3/29)*5 + (1/29)*6 + (6/29)*4 + (1/29)*6 + (2/29)*6 + (4/29)*5 +(3/29)*5 +(1/29)*6 +(1/29)*6 + (2/29)*6
= 10.241…
= 10

CompressionRatio
= $\frac{5-10}{5} * 100\%$ = -1

Average of compression ratios:
= $\frac{-1.2-1-1}{3}$ = -3.2 / 3 = -1.0666….

Minimum of compression ratios:
-1.2

Maximum of compression ratios:
-1

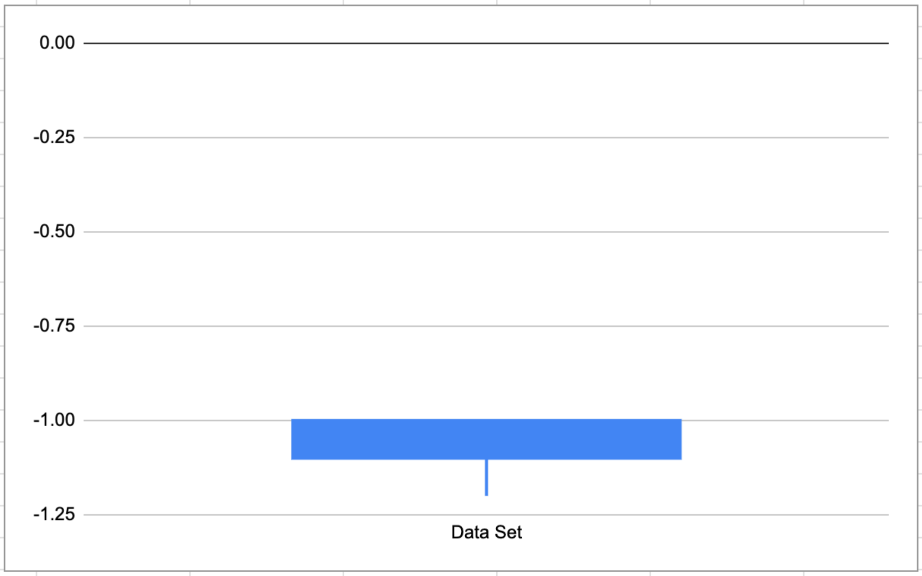
Standard deviation:
Mean, μ = -1.066

$$\sigma^2 = \frac{\Sigma(xi-\mu)^2}{N} = \frac{(-1+1.066)+(-1+1.066)+(-1.2+1.066)}{3}$$

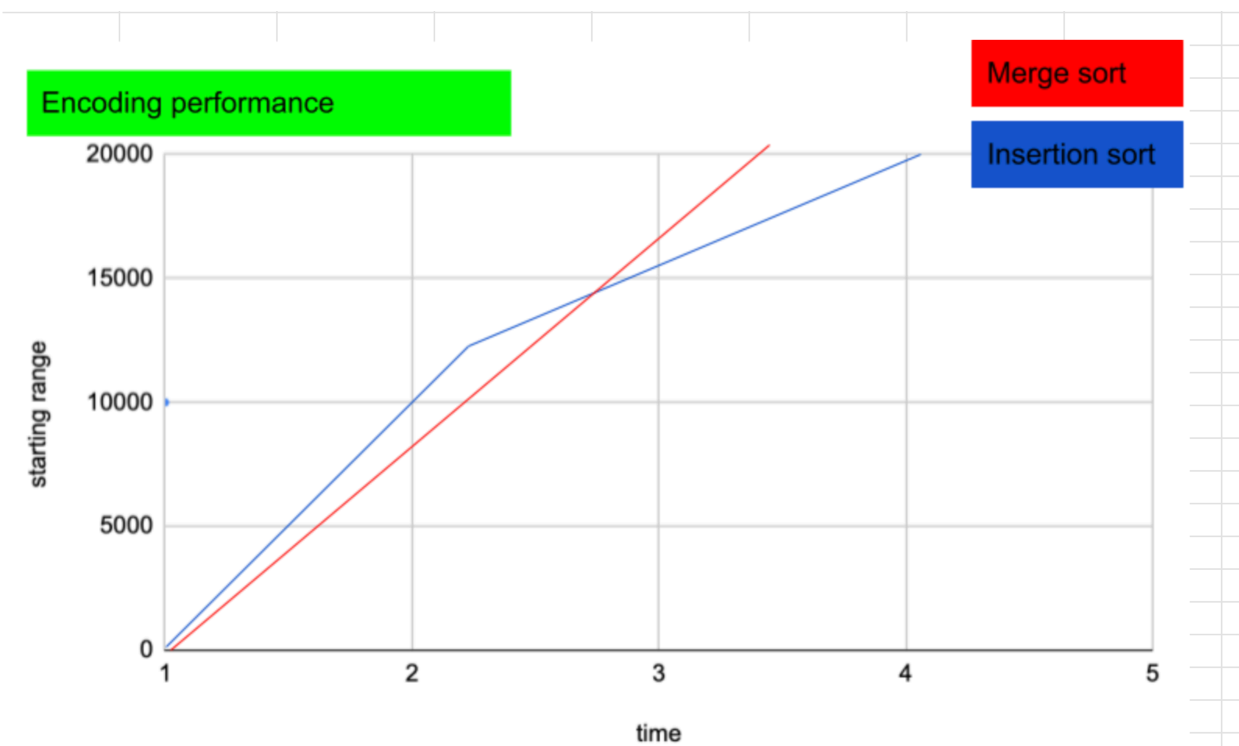
σ = 0.09428…

Box and Whiskers plot of Compression Ratios

MIN	FIRST QUARTIL	THIRD QUARTII	MAX	LABELS
-1.2	-1.1	-1	-1	Data Set

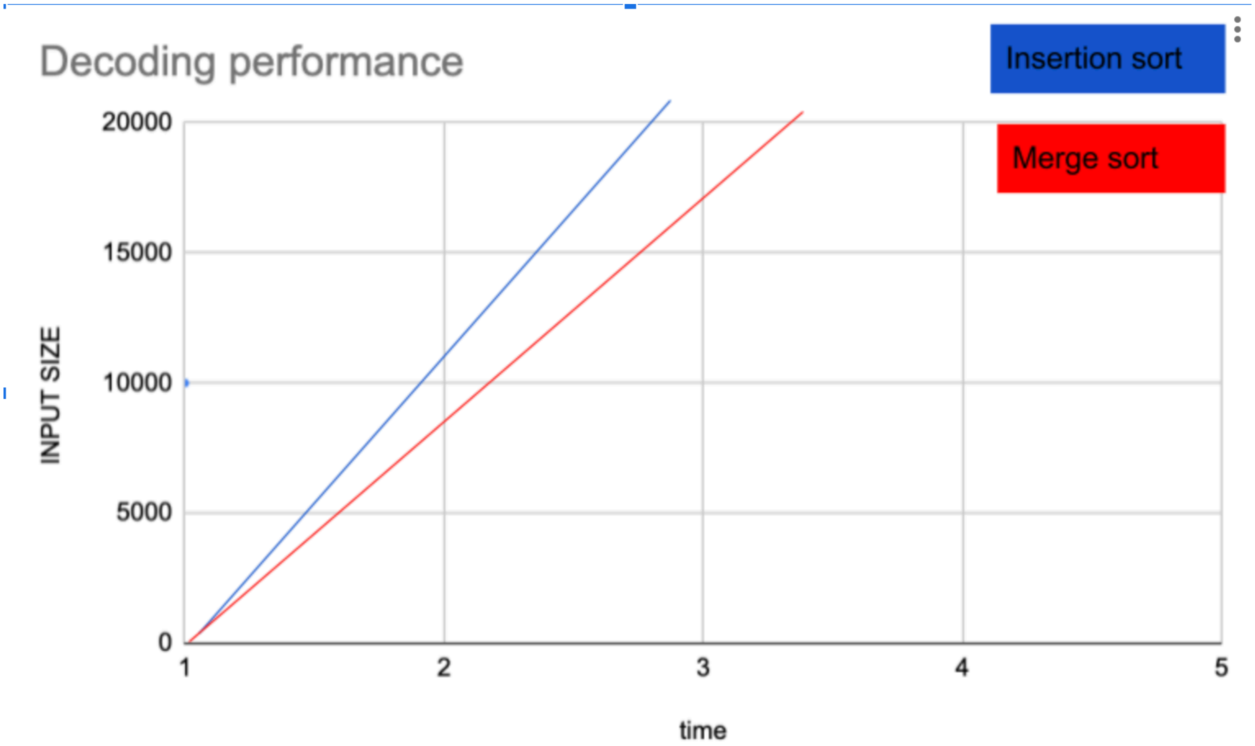


Performance of Merge sort and Insertion sort in Encoding



Performance of Merge sort and insertion sort in Decoding

Analyzing both the graphs above, the performances of Insertion sort and Merge sort respectively during the encoding and decoding testings, a conclusion can be derived that Insertion sort is faster for fewer input values. For less number of input values, Insertion sort takes less amount of time compared to merge sort. While insertion sort is efficient in terms of space, it can be observed that merge is efficient in terms of time in the long run. In the Encoding performance, the running times of Insertion and merge sort cross each other as the



input size increases.