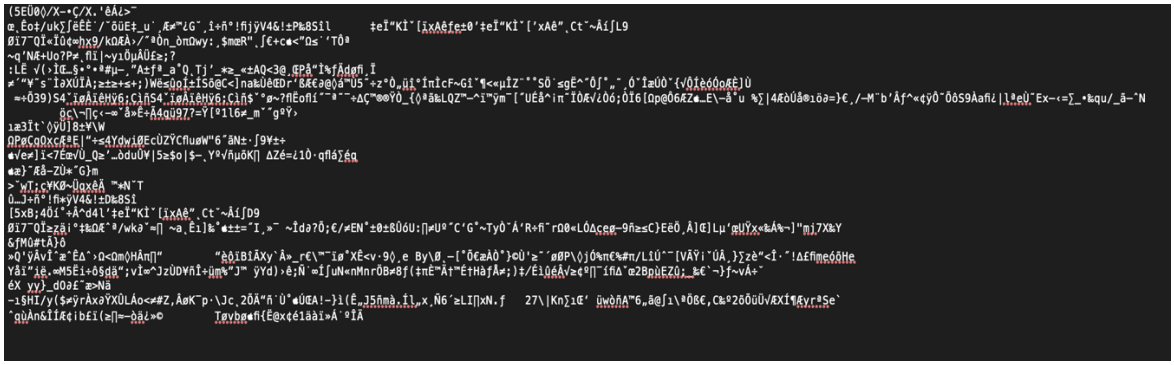
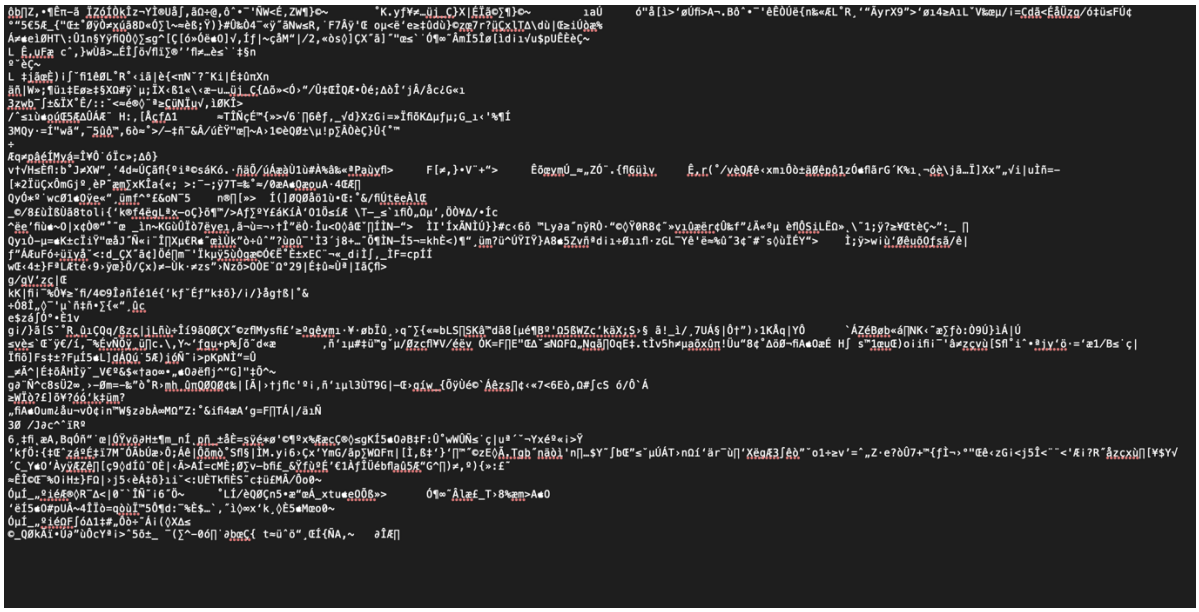
Compression Assignment
Noor Bari 18400034

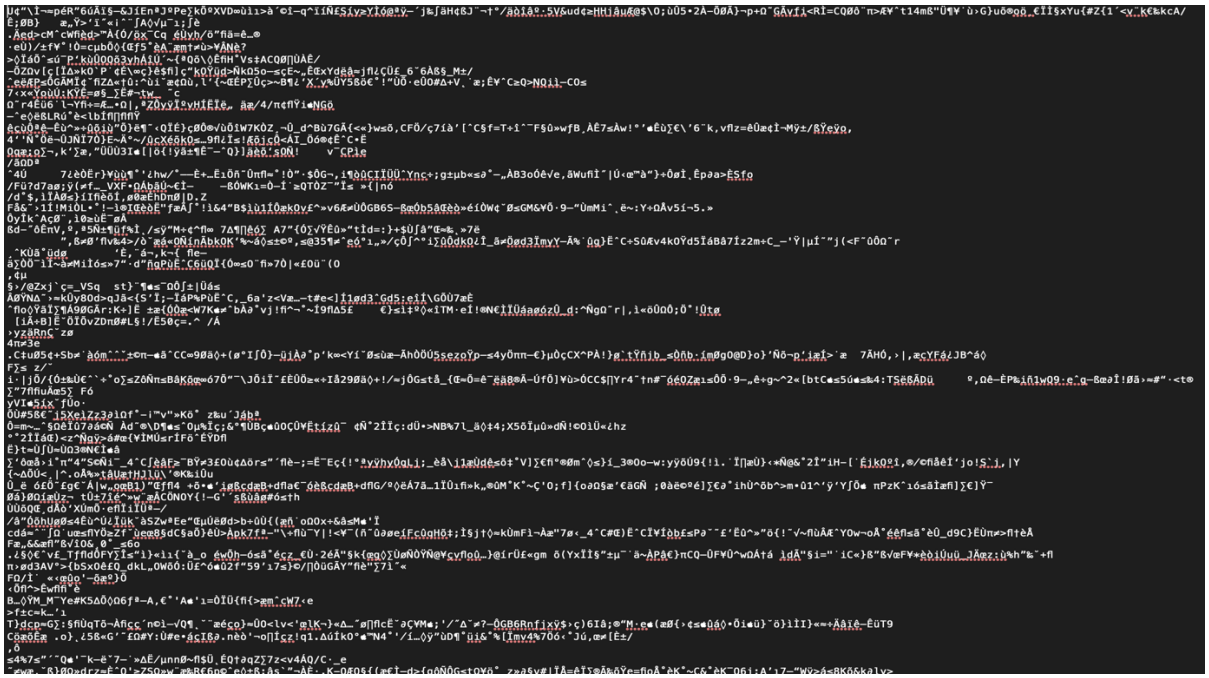
GenomeVirus.txt



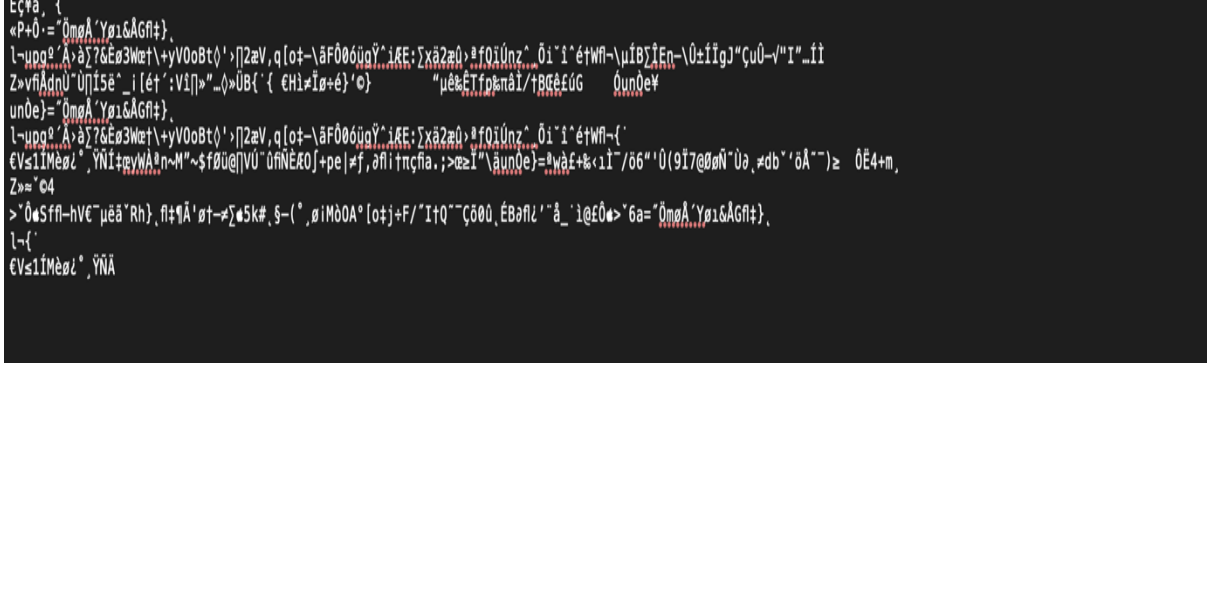
medTale.txt



Mobydick.txt



Flower.txt



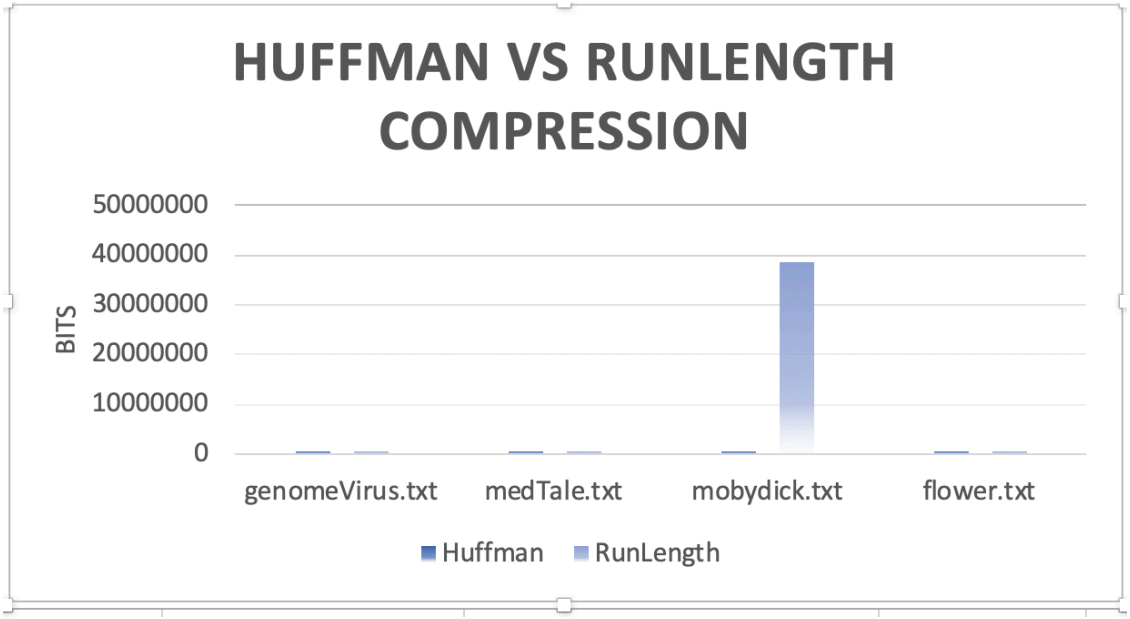
Compression Ratios and Time Taken For Compression :

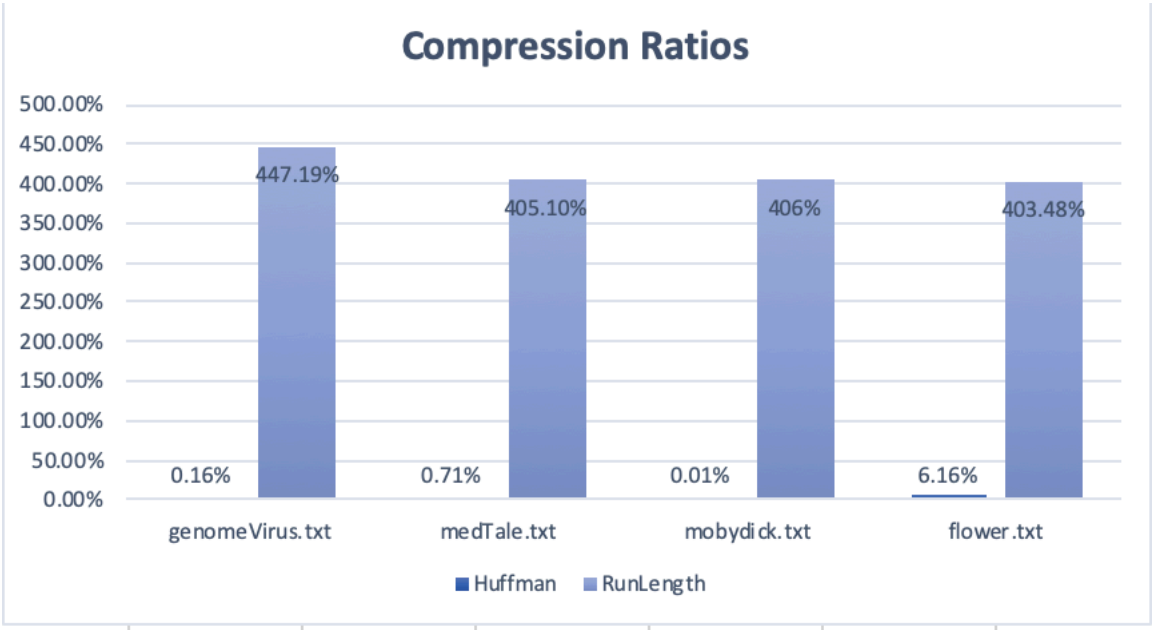
Huffman:

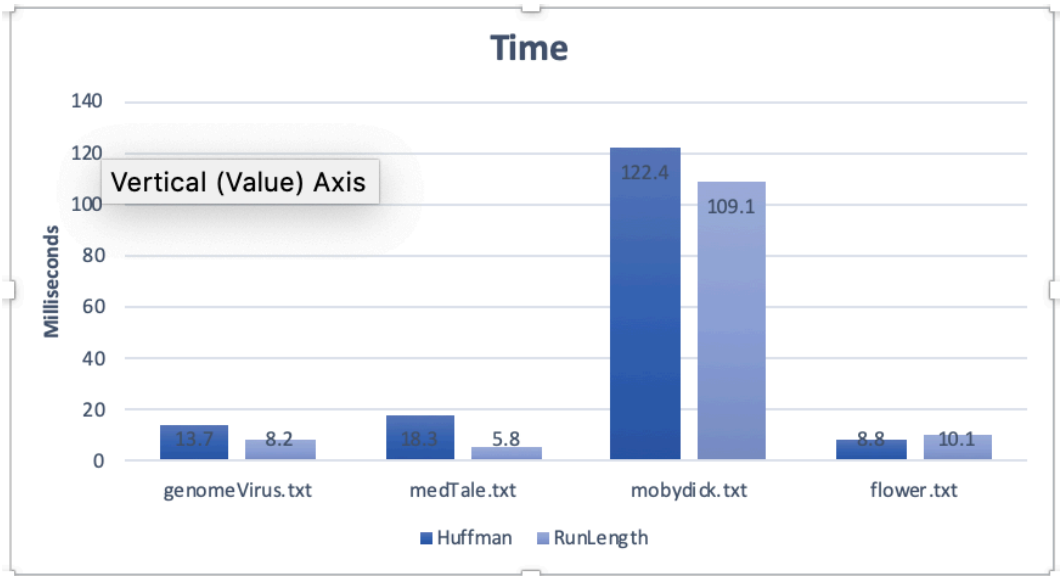
	Original Bits :	Compressed Bits :	Compression Ratio :	Time taken for compression : (milliseconds)
genomeVirus.txt	50008 Bits	80 Bits	0.16%	13.7
medTale.txt	45056 Bits	320 Bits	0.71%	18.3
mobydick.txt	9531704 Bits	848 Bits	0.0089%	122.4
flower.txt	6104 Bits	376 Bits	6.16%	8.8

RunLength Algorithm :

	Original Bits :	Compressed Bits :	Compression Ratio :	Time taken for compression : (milliseconds)
genomeVirus.txt	50008 Bits	223632 Bits	447.19%	8.2
medTale.txt	45056 Bits	182520 Bits	405.10%	5.8
mobydick.txt	9531704 Bits	38698936 Bits	406%	109.1
flower.txt	12632 Bits	50968 Bits	403.48%	10.1


HUFFMAN VS RUNLENGTH COMPRESSION
50000000
40000000
30000000
20000000
10000000
0
BITS
genomeVirus.txt
medTale.txt
mobydick.txt
flower.txt
Huffman
RunLength


Compression Ratios
500.00%
450.00%
400.00%
350.00%
300.00%
250.00%
200.00%
150.00%
100.00%
50.00%
0.00%
447.19%
405.10%
406%
403.48%
0.16%
0.71%
0.01%
6.16%
genomeVirus.txt
medTale.txt
mobydick.txt
flower.txt
Huffman
RunLength



Time Taken For Decompression and Final Bits :

Huffman:

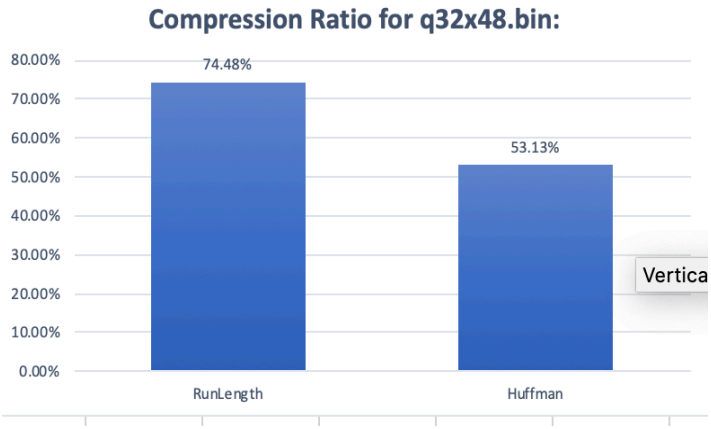
	Final Bits :	Time taken for decompression: (milliseconds)
genomeVirus.txt	50008 Bits	0.5
medTale.txt	45056 Bits	0.6
mobydick.txt	9531704 Bits	60.2
flower.txt	12632 Bits	4.7

RunLength Algorithm:

	Time taken for decompression: (milliseconds)
genomeVirus.txt	12
medTale.txt	11
mobydick.txt	166
flower.txt	5.1

Original bits for q32x48.bin : 1536 bits

	Compressed Bits for q32x48.bin :	Compression Ratio for q32x48.bin:
RunLength	1144 Bits	74.48%
Huffman	816 Bits	53.125%



We can see from the results that the Huffman algorithm worked better to compress the text files. All of the text files compressed by the Huffman algorithm had lower compression ratios compared to the compression ratios of RunLength. This shows that the Huffman algorithm works better to compress files that contain ASCII characters.

We can see that the Huffman algorithm had slower compression times in comparison to the RunLength algorithm, except for the flower.txt file. It had a faster compression time with the Huffman algorithm.

When I compressed an already compressed file it increased the size of the file by a small amount. I think this happens because it needs to convert it from that data set to another set of data. This results in a slightly bigger compressed file which is not ideal as compression should make the size of the file smaller.
When compressing the bitmap file q32x48.bin we can see that the Huffman algorithm has a lower compression ratio compared to the RunLength compression algorithm. Even though the RunLength algorithm gave a bigger compression ration than the Huffman algorithm, we can see that RunLength works much better to compress the .bin file than text files. This is because .bin files contain 0's and 1's, as a result it is evident that RunLength is more effective on binary numbers compared to ASCII characters.

Overall from the results given, it is clear that the Huffman algorithm is a far more effective algorithm when compressing text and bin files. It gives a lower compression ratio which is what we want to achieve.