COMP20290 - Huffman Algorithm Assignment Report

Thomas Reilly - 18483722

Compression Test Results (Q1 & Q2)

File	Original SIze (bytes)	Compressed Size (bytes)	Decompressed Size (bytes)	Compression Ratio	Compression Time (ns)	Decompression Time (ns)
mobydick.txt	1,191,463	667,651	1,191,463	1191463:667651	643687066	233176069
genomeVirus.txt	6,251	1,755	6,251	6251:1755	70122460	33204890
medTale.txt	5,632	2,987	5,632	5632:2987	49760811	18800182
algorithm.txt	8,039	4,574	8,039	8039:4574	68877499	42240219
q32x48.bin	192	102	192	32:17	11040161	2706580

NOTE: Compression outputs can be found in the *Comperssion_Assignment* package under *task-3-compressed-files*.

Q3. What happens if you try to compress one of the already compressed files? Why do you think this occurs?

I compressed the already compressed medTale.txt file. The new compression file was 3245 bytes long. This is an increase of 258 bytes on the already compressed file.
This likely occurs because the compressed file is already reduced down and the second compression is only adding extra bytes with the addition of a second huffman tree

Q.4 Use the provided RunLength function to compress the bitmap file q32x48.bin. Do the same with your Huffman algorithm. Compare your results. What reason can you give for the difference in compression rates?

Original size: 192 bytes
RunLength compression: 143 bytes
HuffmanAlgorithm compression: 102 bytes
The Huffman Algorithm performs better on this particular bitmap file thanks to its small size. It's worth noting however that RunLength might perform better than HuffmanAlgorithm on larger bitmap files due to recurring pixels.