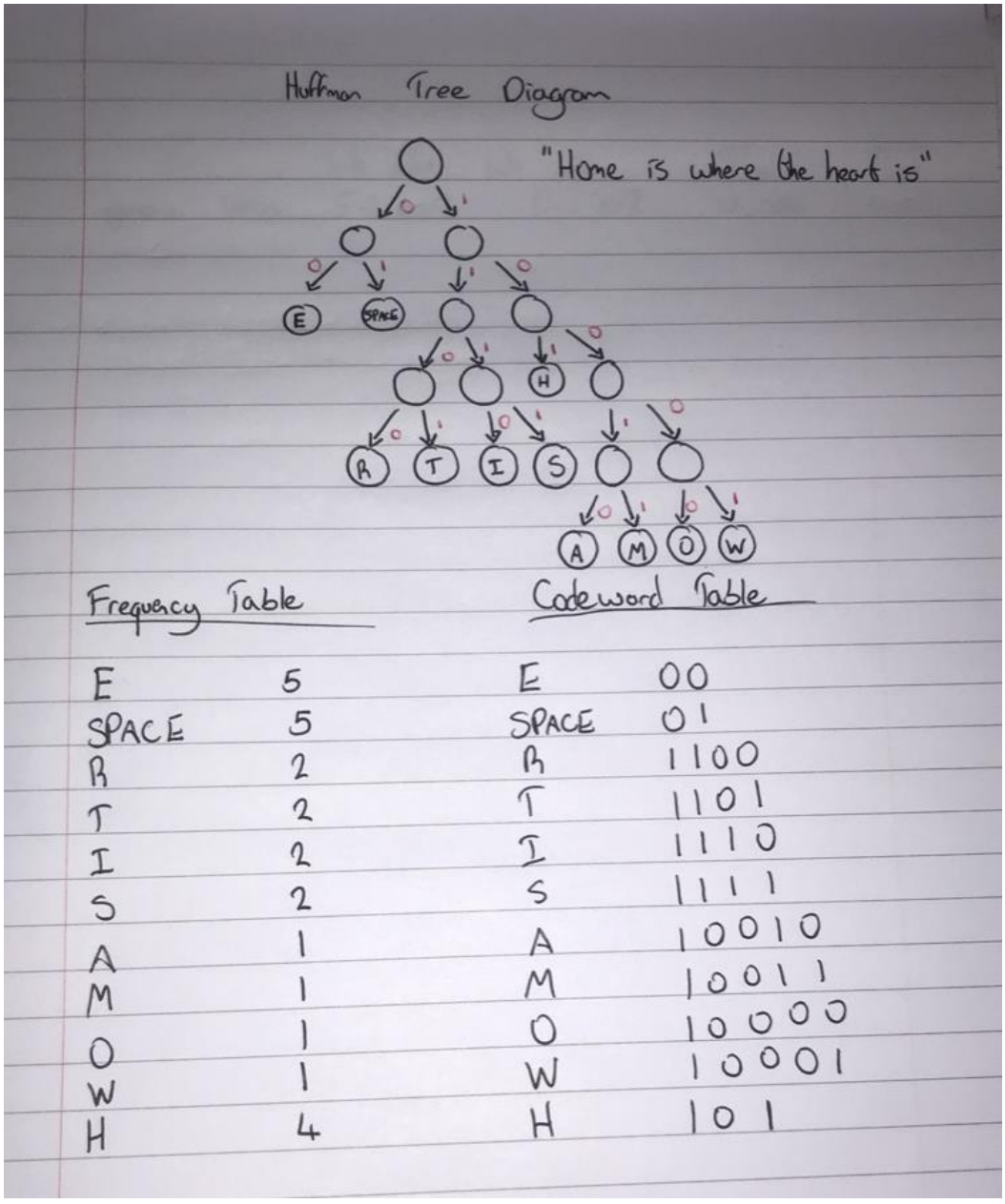
COMP20290
Algorithms

Assignment 2

Huffman Compression

Gearoid Mulligan: 19343146

Task 1: Huffman Tree by Hand



Task 3: Compression Analysis: Step 1 & 2

File Name	Bits	Compression Time	Compressed Bits	Decompression Time	Ratio	Decompressed Bits
GenomeVirus	50,008	0.0	12,576	0.0	12,576/50,008 = 74% compression	50,008
MedTale	45,808	0.0	24,616	0.0	24,616/45,808 = 46% compression	45,808
Moby Dick	9,708,952	0.031	5,505,424	0.014	5505424/9708952 = 43% compression	9,708,952
Step Brothers Script	650,872	0.001	381,120	0.0	381,120/650,872 = 41% compression	650,872

Step 3:

As you can see from the following 4 files I compressed and decompressed, excluding the GenomeVirus which had many re-occurring letters the average compression was around 43%. As you can see for the times it executed them very quickly as the algorithm has a time complexity of O(nlogn). The longest part of this was writing out the binary in the command prompt, especially for moby dick as it took many minutes to write out the bits. Overall, its clear to see that it is a very efficient algorithm.

Q3:

When I compressed one of the already compressed files it seemed to compress it again. I verified this by compressing the gonemeVirus file that was already compressed to another file. I then decompressed this “double” compressed file and it gave me the same random text as in the original decompressed genomeVirus text file. I think this occurs because the compression function takes the compressed file and compresses the random characters that occur when you compress the text file. The number of bits also increased from 12,576 to 15,152 when I compressed the already compressed files also.

Q4:

Algorithm	Bits Before	Bits after	Compression Ratio
Huffman	1536	816	47%
RLE	1536	1114	27%

From the table you can clearly see that the Huffman algorithm is more efficient, nearly twice as efficient as the Run Length function. The reason to why I think this is because the run length algorithm doesn’t create a table of the most frequent letters and assign them with the smallest length of encoding. Instead, it just counts how many chars there are that are the same and each char is a byte of memory and each number is a byte of memory which proves also less efficient than the