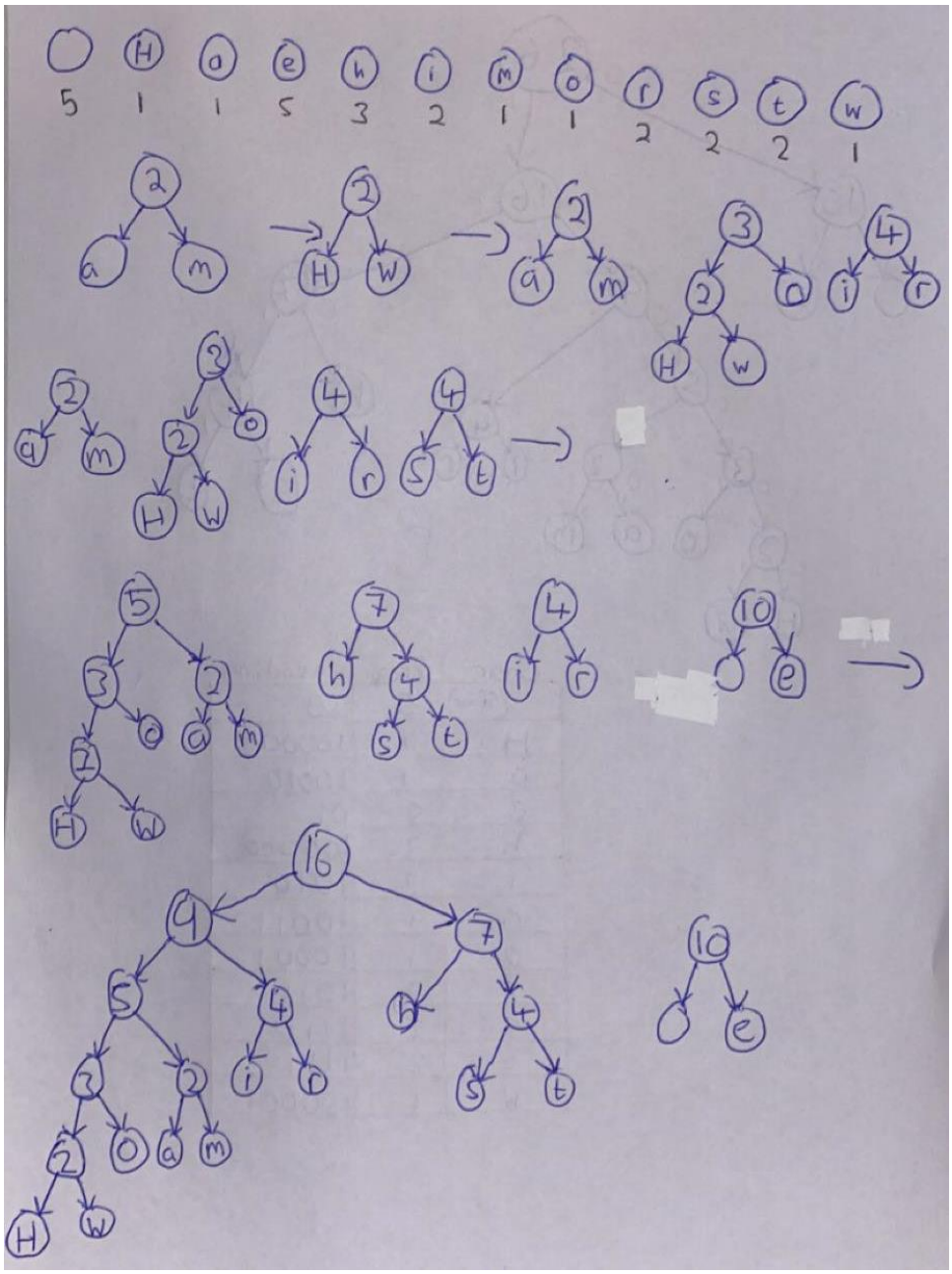
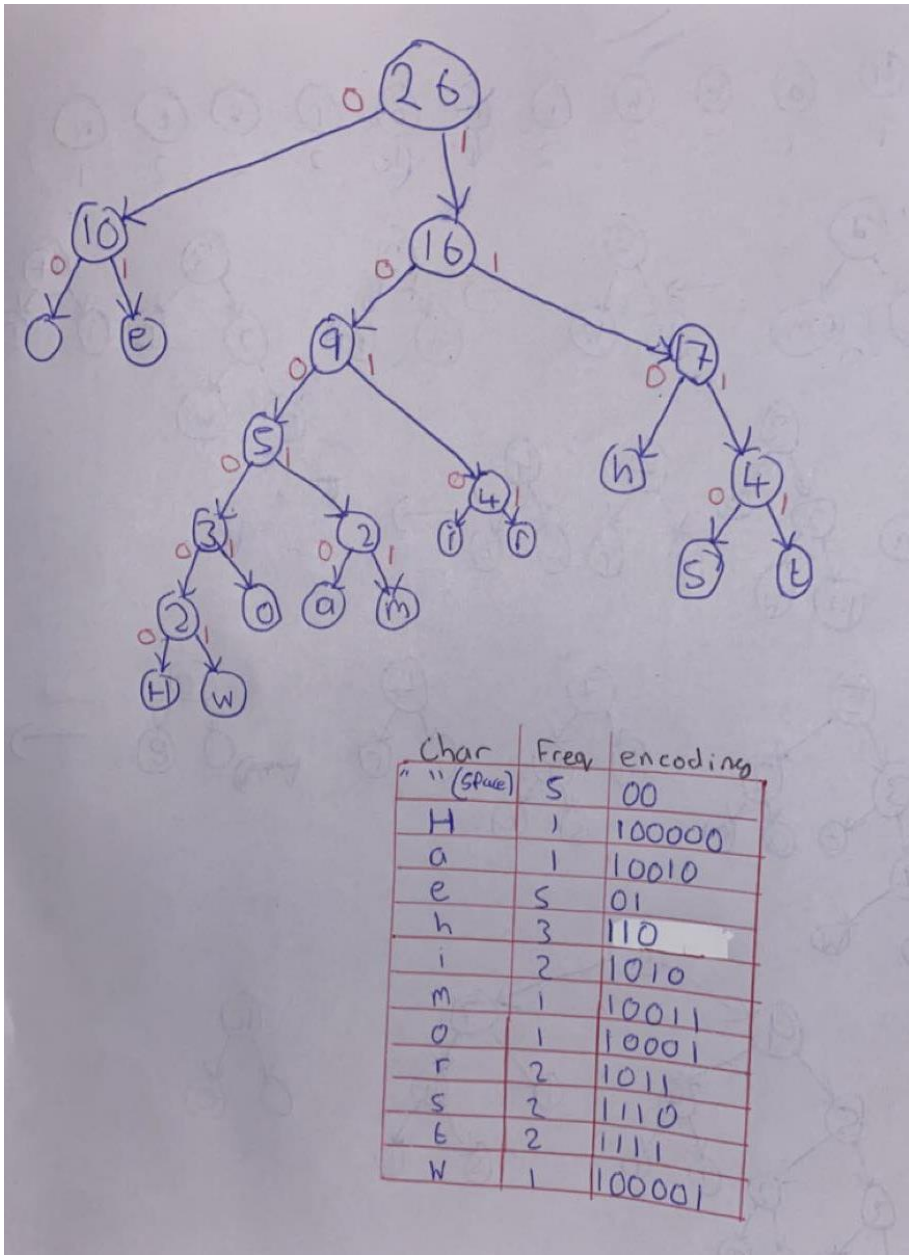
Assignment

Huffman Compression

Ross Murphy – 20207271

Task 1 – Code Huffman Tree of phrase by hand:





Char	Freq	encoding
" " (Space)	5	00
H	1	100000
a	1	10010
e	5	01
h	3	110
i	2	1010
m	1	10011
o	1	10001
r	2	1011
s	2	1110
t	2	1111
w	1	100001

Task 3 - Compression Analysis:

Step 1:

Q1:

File to compress	Starting Bits	Compressed Bits	Compression Ratio	Time in Milliseconds
genomeVirus.txt	50008	12576	25.15%	6
medTale.txt	45056	23912	53.07%	6
mobydick.txt	9537200	5348624	56.08%	80
q32x48.bin	1536	816	53.31%	3
myText.txt	7488	4824	64.42%	4

Q2:

Files to decompress	Final Bits	Time in Millisecond
genomeVirusrle.txt	50008	3
medTalerle.txt	45056	3
Mobydickrle.txt	9537200	40
q32x48rle2.bin	1536	2
myTextrle.txt	7488	2

Q3 - Analysis:

From looking at the compression table we can see that some of the files compress better than others. The genomeVirus.txt file has a compression ratio of 25.15%. This is because it only consists of 4 characters, A, T C and G and they all occur regularly. It converts the 8-bit characters to 2-bit codes resulting in approx. 25% compression. The mobyDick.txt, medTale.txt and my sample text, myText.txt are all files that contain the English language, and the compression ratio sits between 53-65 percent which is very good. The final file, q32x48.txt has a compression ratio of 53.31%. This is a bitmap is reduced to approximately half the original size which is a good compression ratio.

As we can see from the tables the time to decompress the files is much lower than the time to compress them, approximately half the time. This is due to the fact that when compressing the files the Huffman tree needs to be built, whereas it only needs to expand an already made tree in order to decompress the file back to the same original bits.

Compressing a compressed file:

Compressing an already compressed file results in the new file having more bits than the original compressed file. My guess as to why this happens is probably because there are many unfrequently occurring symbols you can see if you open the compressed file.

Comparing RunLength and Huffman:

The Huffman algorithm compresses the q32x48.bin file from **1536 bits** to **816 bits** giving a compression ratio of **53.31%** compared to RunLength which compress it from **1536 bits** to **1144 bits** giving a compression ratio of **74.5%.** This is due to the fact that RunLength compression needs the 8-bit characters to occur in order frequently to have a more effective compression ratio, whereas with Huffman compression the doesn't need the characters to occur frequently and results in less symbols occurring.