Task 3:

*Note: When I ran my files, I ran it from src and I use “java compression_assignment\Huffman.java compress < ‘path to input file’ > ‘path to output file’” like shown below. It might be hard to see so just zoom in.

Q1 .

File	Original Bits	Compressed Bits	Compression Ratio	Compression Time
genomeVirus.txt	50,008	12,576	25.148%	0.007
medTale.txt	45,056	23,912	53.072%	0.010
mobydick.txt	9,531,704	5,341,208	56.036%	0.067
q32x48.bin	1,536	816	53.125%	0.0
bohemian.txt	15,272	9,336	61.131%	0.0

Q2.

File	Compressed Bits	Decompressed Bits	Decompression Time
genomeVirus.txt	12,576	50,008	0.0
medTale.txt	23,912	45,056	0.015
mobydick.txt	5,341,208	9,531,704	0.070
q32x48.bin	816	1,536	0.0
bohemian.txt	9,336	15,272	0.0

Q3. When I tried to compress bohemianCompressed.txt, the output gave a size of 11,552 which is bigger than the initially compressed file.

Q4.

q32x48.bin	Original Bits	Compressed Bits	Compression Ratio
Huffman	1,536	816	53.125%
RLE	1,536	1,144	74.479%

- Huffman gave a smaller size for the compressed file because Huffman looks at a sequence of 8-bit bytes and give that specific sequence a unique code. In the bitmap, there is going to be repeated sequences of bytes which makes using code much better than looking at runs of bits (e.g. 00000001 is a run, 001 is also a run) which is what RLE does. If the RLE algorithm encounters consecutive short runs it will produce a bigger compression size. For example, we 10101010, RLE will see 4 runs of 10 while Huffman will only see one byte sequence and give it a short code. Because of this, there is more redundancy in RLE which gives it a bigger compression size.