Assignment Five Report:

==================================================================================

Question One:

Binary Tree Codes:

\n = 111011

" = 11110110

= 110

! = 1110100111

' = 111010010

\* = 11110111010010

( = 1111011111111

) = 010010111000

. = 1110101

/ = 0100101110010100100

, = 010001

- = 100011001

3 = 0100101110010111

2 = 111101110100000

1 = 11110111010001

0 = 111101110100001

7 = 01001011100111110

6 = 0100101110010110

5 = 0100101110010101

4 = 010010111001010011

; = 111101110101

: = 111010001001

9 = 01001011100111101

8 = 01001011100100

? = 1000110100

= = 0100101110010100101

D = 11110111000

E = 01001010001

F = 11101000101

G = 111101111101

A = 010010110

B = 1110100001

C = 01001010000

L = 1111011111110

M = 1000110101

N = 1110100000

O = 01001011101

H = 1110100011

I = 100011011

J = 11110111010011

K = 111101111100

U = 01001011100110

T = 100011000

W = 0100101001

V = 111010001000

Q = 01001011100111111

P = 010010101

S = 0100101111

R = 11110111011

Y = 111101111110

X = 01001011100111100

Z = 010010111001110

f = 101100

g = 100010

d = 10111

e = 000

b = 1111010

c = 111001

a = 1001

n = 0110

o = 0111

l = 10000

m = 111000

j = 11110111001

k = 0100100

h = 0011

i = 0101

w = 101101

v = 1000111

u = 111100

\r = 01001011100101000

t = 1010

s = 0010

r = 11111

q = 11110111101

p = 010000

z = 11110111100

y = 010011

x = 1110100110

war\_and\_peace.txt:

output length: 1792281 bytes

=====================================================================================================

Question Two:

war\_and\_peace.txt: taisho.txt: pi.txt

input length: 3196247 bytes input length: 3649944 bytes input length: 1010003 bytes

output length: 1792281 bytes output length: 1542656 bytes output length: 443632 bytes

Reduction: 56% Reduction: 42% Reduction: 44%

war\_and\_peace.txt achieves the best encoding as it consists of a high volume of repeated characters that can be represented with shorter bit strings. In totality, this allows us to compile a shorter bit string that represents the data as a whole. Instead of processing each character individually, using one bit after the other, we can use shortened binary codes to effectively minimise the total number of bits used to represent the data.

Data is harder to compress when there is a high volume of unique characters, which is evident in 'taisho.txt' where there is a greater volume of unique characters than war\_and\_peace.txt hence it produces a lower reduction rate.

=====================================================================================================================

Question Three:

- The size of the sliding window is directly proportional to the time and quality of compression in Lempel-Ziv.

- Using a small window size could eliminate the possibility of detecting previous patterns in the text as text matching is only considered within the bounds of the sliding window which in effect, produces lower quality compressions. However, using a minimal sized window does enhance the time it takes to compress the data as it searches for patterns within smaller bounds.

- Using a large window size increases the time it takes to compress the data as the algorithm attempts to match patterns on a larger scale window. However, this results in a higher quality compression as it is able to find more patterns within a larger sized window which effectively reduces the degree of redundancy in the data.

- The optimal window size would strike an ideal balance between time and compression quality which are determined by the size and content of the data.

======================================================================================================================

Question Four:

HuffmanCoding huffman = new HuffmanCoding(text);

String encoded = huffman.encode(text);

LempelZiv lz = new LempelZiv();

String enc = lz.compress(encoded);

String decoded = lz.decompress(enc);

- HuffmanEncode + LempelZiv compression results in a smaller file size due to the combination of bit string size reduction and the elimination of redundant patterns.

- This combination of compression methods is effective as the data that is passed onto the LempelZiv compressor is already partially compressed by the HuffmanEncode, meaning the input is not a raw bit representation of itself but instead it is in its minimal bit string form.

- Hence the quality of compression done by Lempel-Ziv is only further enhanced as the patterns it detects are minimal bit represenations of itself which in effect results in a high quality compression. Without HuffmanEncode it only detects patterns from the raw bit representation of the text which is evidently not as effective as detecting patterns that are of its minimal bit representation.