Question One:

KMP:

- Search at start of text: ""Well, Prince, so Genoa and Lucca" (34 char) 2ms
- Middle of text: "Napoleon looked up and down the r" (34 char) 4ms
- End of text: "In the first case it was necessary" (34 char) 7ms

After testing KMP 45,844 times with lines of exactly 33 characters in length the average was calculated to be 4.430033ms

Brute Force:

- Search at start of text: ""Well, Prince, so Genoa and Lucca" (34 char) 3ms
- Middle of text: "Napoleon looked up and down the r" (34 char) 5ms
- End of text: "In the first case it was necessary" (34 char) 6ms

After testing BruteForce 45,844 times with lines of exactly 33 characters in length the average was calculated to be 4.159588ms

Question Two:

```
Char = 'e' encoded with code = 111

Char = 's' encoded with code = 1101

Char = 'h' encoded with code = 1100

Char = 'i' encoded with code = 1011

Char = 'n' encoded with code = 1010

Char = 'k' encoded with code = 1001111

Char = 'C' encoded with code = 10011101111

Char = 'E' encoded with code = 10011101110

Char = 'W' encoded with code = 1001110110

Char = 'P' encoded with code = 100111010

Char = 'A' encoded with code = 100111001

Char = ')' encoded with code = 100111000111

Char = '8' encoded with code = 10011100011011

Char = '4' encoded with code = 10011100011010111

Char = 'à' encoded with code = 1001110001101011011

Char = '' encoded with code = 10011100011010110101l
Char = 'ä' encoded with code = 1001110001101011010100

Char = '=' encoded with code = 100111000110101101001

Char = '/' encoded with code = 100111000110101101000

Char = 'ê' encoded with code = 100111000110101100

Char = '5' encoded with code = 1001110001101010

Char = '6' encoded with code = 1001110001101001

Char = '3' encoded with code = 1001110001101000

Char = 'U' encoded with code = 10011100011001

Char = 'Z' encoded with code = 100111000110001

Char = 'X' encoded with code = 10011100011000011

Char = '9' encoded with code = 10011100011000010

Char = '7' encoded with code = 10011100011000001

Char = 'Q' encoded with code = 10011100011000000

Char = 'O' encoded with code = 10011100010

Char = 'S' encoded with code = 1001110000

Char = 'y' encoded with code = 100110

Char = 'l' encoded with code = 10010

Char = 'o' encoded with code = 1000

Char = 'a' encoded with code = 0111

Char = 'g' encoded with code = 011011

Char = 'T' encoded with code = 011010111

Char = '-' encoded with code = 011010110

Char = '?' encoded with code = 0110101011

Char = 'M' encoded with code = 0110101010

Char = 'I' encoded with code = 011010100

Char = 'v' encoded with code = 0110100

Char = 'f' encoded with code = 011001

Char = 'w' encoded with code = 011000

Char = 't' encoded with code = 0101

Char = 'd' encoded with code = 01001

Char = 'm' encoded with code = 010001

Char = 'c' encoded with code = 010000
Char = ' ' encoded with code = 001

Char = 'N' encoded with code = 0001111111

Char = 'B' encoded with code = 0001111110

Char = 'V' encoded with code = 000111110111

Char = ':' encoded with code = 000111110110

Char = 'F' encoded with code = 00011111010

Char = 'H' encoded with code = 0001111100

Char = ''' encoded with code = 000111101

Char = 'x' encoded with code = 0001111001

Char = '!' encoded with code = 0001111000

Char = '.' encoded with code = 0001110

Char = '' encoded with code = 000110

Char = '\n' encoded with code = 000101

Char = 'u' encoded with code = 000100

Char = 'r' encoded with code = 00001

Char = 'b' encoded with code = 0000011

Char = '"' encoded with code = 00000101

Char = 'D' encoded with code = 00000100111

Char = 'j' encoded with code = 00000100110

Char = '2' encoded with code = 000001001011111

Char = '0' encoded with code = 000001001011110

Char = '1' encoded with code = 00000100101110

Char = '*' encoded with code = 00000100101101

Char = 'J' encoded with code = 00000100101100

Char = ';' encoded with code = 000001001010

Char = 'R' encoded with code = 00000100100

Char = 'z' encoded with code = 00000100011

Char = 'q' encoded with code = 00000100010

Char = 'K' encoded with code = 000001000011

Char = 'G' encoded with code = 000001000010

Char = 'Y' encoded with code = 000001000001

Char = 'L' encoded with code = 0000010000001
Char = '(' encoded with code = 0000010000000

Char = 'p' encoded with code = 0000001

Char = ',' encoded with code = 0000000
```

input length: 3258246 bytes

output length: 1848598 bytes

Resulting in a reduction of 57% in file size (War and Peace).

Question Three

War and Peace – 57%
Taisho – 42%
Pi – 43%

War and Peace had the greatest reduction of 57%. This is mostly likely due to the relatively small alphabet size (compared to taisho), which means there will be more of certain characters which could have a rather small encoding code.

Question Four

I found that increasing the window size for the text “War and Peace” generally resulted in a better result in compression, which a window size of 10240 having a compressed size of 5573913 characters. While a window size of 100 gave out a compressed size of 10020112 characters.

Question Five

After applying Huffman encoding algorithm to War and Peace and then applying Lempel Ziv I ended up with a file size of:

Input length: 14788789 characters

Output length: 8194589 characters,

Which if we take literally is an overall net gain of bytes over the original war and peace file size which is 3258227 characters in length.