

## COMP261 Assignment 5 Report

**readme:** There seems to be an error in my code or with the data as when I run the String search program and Huffman encode the texts, the input lengths are much greater than what they actually seem to be. I have checked with my mates and the input lengths they are achieving are all the same, however mine does seem to be wrong and this will have changed the answers that I have been writing.

### Question 1: Write a short summary of the performance you observed using the two search algorithms.

From what I have observed using the two algorithms, KMP and Brute Force, KMP search seemed to be on average performed slightly faster than the Brute force. Testing all the texts, the time taken for both searches was quite similar however as mentioned KMP search was slightly better and used less time to find the given string.

### Question 2: Report the binary tree of codes your algorithm generates, and the final size of *War and Peace* after Huffman coding.

Input length: 14788788 bytes

Output length: 1848598 bytes

Original and decoded texts match.

```
:111010
:111001
:110
!:1110000111
":11111010
':111000010
(:111110111111
):011000111000
*:11111011010010
,:111111
-:100101001
.:1110001
/:01100011100101011110
0:111110110100001
1:11111011010001
2:111110110100000
3:0110001110010111
4:01100011100101010
5:0110001110010100
6:0110001110010110
7:01100011100111110
8:01100011100100
9:01100011100111101
::111000001001
;:111110110101
=:01100011100101011111
?:1001010100
A:011000110
B:1110000001
C:01100010000
D:11111011000
E:01100010001
F:11100000101
G:111110111101
H:1110000011
```

I:100101011  
 J:11111011010011  
 K:111110111100  
 L:1111101111110  
 M:1001010101  
 N:1110000000  
 O:01100011101  
 P:011000101  
 Q:01100011100111111  
 R:11111011011  
 S:0110001111  
 T:100101000  
 U:01100011100110  
 V:111000001000  
 W:0110001001  
 X:01100011100111100  
 Y:111110111110  
 Z:011000111001110  
 à:0110001110010101110  
 a:1000  
 b:1111100  
 c:101111  
 d:10110  
 ä:0110001110010101111010  
 e:000  
 f:100110  
 g:100100  
 h:0011  
 é:0110001110010101111011  
 i:0100  
 j:11111011001  
 ê:011000111001010110  
 k:0110000  
 l:01101  
 m:101110  
 n:0101  
 o:0111  
 p:1111110  
 q:11111011101  
 r:11110  
 s:0010  
 t:1010  
 u:111011  
 v:1001011  
 w:100111  
 x:1110000110  
 y:011001  
 z:11111011100  
 :011000111001010111100

**Question 3: Consider the Huffman coding of war\\_and\\_peace.txt, taisho.txt, and pi.txt. Which of these achieves the best compression, i.e. the best reduction in size? What makes some of the encodings better than others?**

War and Peace reduction size: 12,940,190 bytes

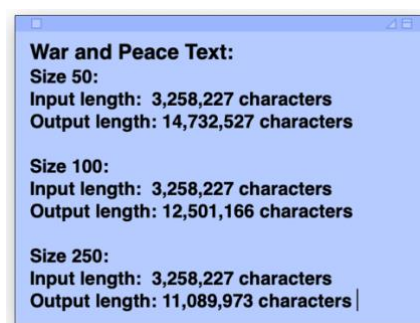
Taisho (input: 12341255, output: 1542656) reduction size: 10,798,599 bytes

Pi (input: 3549064, output: 443632) reduction size: 3,105,432 bytes

From the data, the best reduction in size was the War and Peace.txt. This is because this text has a very wide frequency distribution, and possibly the small alphabet size. This means that the war and peace text will contain more characters which could have smaller encoding codes. What makes some of the encodings better than others is that they have smaller character sets.

**Question 4: The Lempel-Ziv algorithm has a parameter: the size of the sliding window. On a text of your choice, how does changing the window size affect the quality of the compression?**

Using War and Peace text, I tested the test using three different window sizes; 50, 100 and 250. From the results, as the window size got larger the output length size became smaller. As you can see from the picture below, the compression got worse as the size increased.



War and Peace Text:	
Size 50:	
Input length:	3,258,227 characters
Output length:	14,732,527 characters
Size 100:	
Input length:	3,258,227 characters
Output length:	12,501,166 characters
Size 250:	
Input length:	3,258,227 characters
Output length:	11,089,973 characters

**Question 5: What happens if you Huffman encode War and Peace *before* applying Lempel-Ziv compression to it? Do you get a smaller file size (in characters) overall?**

Original input length: 14,788,788 bytes  
After Huffman encoding: 1,848,598 bytes  
After Lempel-Ziv compression: 16,447,594 bytes

The final compressed file size is definitely a lot larger when encoding with both algorithms.