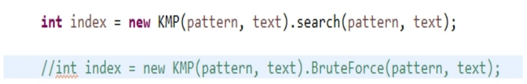
Question 1: Write a short summary of the performance you observed using the two search algorithms.



```
int index = new KMP(pattern, text).search(pattern, text);

//int index = new KMP(pattern, text).BruteForce(pattern, text);
```

I test by common another each time. Pattern that I define that is astronomy, because it enough far away to search word. Otherwise, it will be hard to find difference. Checking word and time table shows below.

algorithm Search string	well	astronomy
KMP algorithm	0	16
Brute Force algorithm	0	32

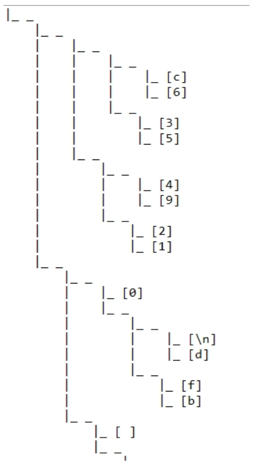
When the searching string is stay at forward. There is not difference in time, but when word in the final part of text. KMP algorithm is faster than brute force algorithm.

KMP:cost O(m+n) time

Brute Force: cost O(mn) time

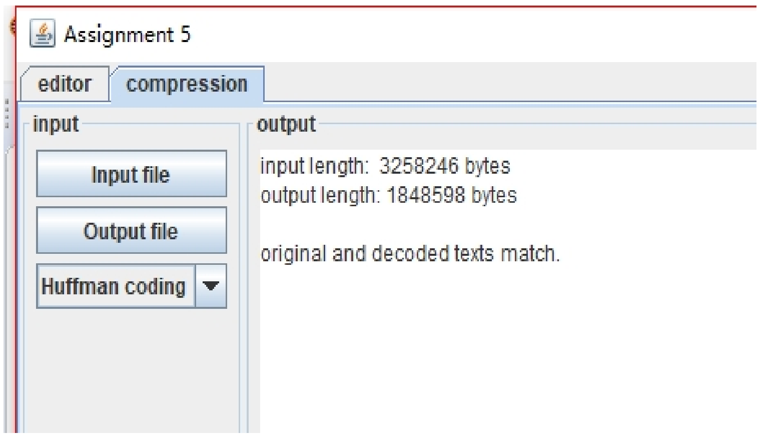
Question 2: Report the binary tree of codes your algorithm generates, and the final size of War and Peace after Huffman coding.

I use comparator to range the order of node's frequency. When node was added to queue, it will range from small to large directly. Each time poll two smallest frequency node from queue and combine them together. So this binary tree looks like below.



```
|_ _
    |_ _
    |    |_ _
    |    |    |_ _
    |    |    |    |_ [c]
    |    |    |    |_ [6]
    |    |    |_ _
    |    |        |_ [3]
    |    |        |_ [5]
    |    |_ _
    |        |_ _
    |        |    |_ [4]
    |        |    |_ [9]
    |        |_ _
    |            |_ [2]
    |            |_ [1]
    |_ _
        |_ _
        |    |_ [0]
        |    |_ _
        |        |_ _
        |        |    |_ [\n]
        |        |    |_ [d]
        |        |_ _
        |            |_ [f]
        |            |_ [b]
        |_ _
            |_ [ ]
            |_ _,
```

Final size of war and peace after Huffman coding is 1848598 bytes.



```
input length:  3258246 bytes
output length: 1848598 bytes

original and decoded texts match.
```

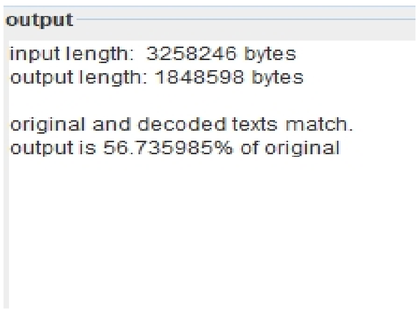
Question 3: Consider the Huffman coding of war_and_peace.txt, taisho.txt, and pi.txt. Which of these achieves the best compression, i.e. the best reduction in size? What makes some of the encodings better than others?

Pi.txt: 43.9% of original
Taisho.txt: 42.2% of original
War_and_peace.txt: 56.7% of original
Taisho.txt is best compression in size
A bit is a single binary digit, 0 or 1 (1/8 byte).
A byte is 8 bits for letter or number, Chinese word is 2 byte(16 bits).
Because we replace the one word by using 1 or 0 which only have one bit. Thus when Chinese word was replaced, it release more space rather than before.

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?

When the size of sliding window increase, quality of compression also increase, but it waste more time than smaller windows size. Because it increase the repeated pattern when window size increased. Thus the length of character which repeated will increase.

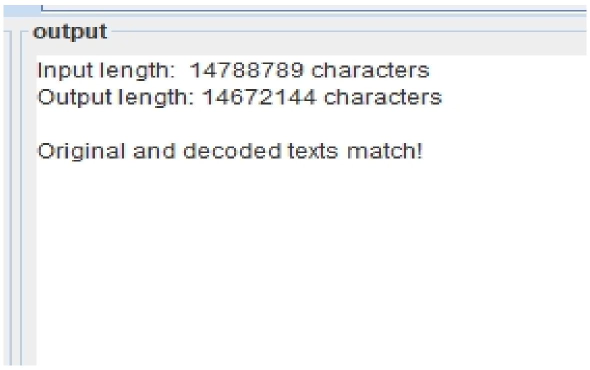
Question 5: What happens if you Huffman encode War and Peace before applying LempelZiv compression to it? Do you get a smaller file size (in characters) overall?
Huffman encode:



```
input length:  3258246 bytes
output length: 1848598 bytes

original and decoded texts match.
output is 56.735985% of original
```

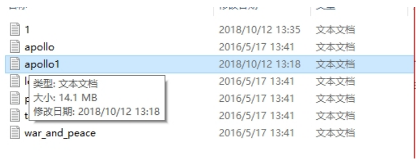
LZ77 encode:



```
output
Input length:  14788789 characters
Output length: 14672144 characters

Original and decoded texts match!
```

It become smaller size in characters , I think the reason maybe that huffman encode make compression as many combination which created by 0 or 1. Thus, more repeated pattern was added. But there is no difference between two size of file both 14.1MB.



```
1                  2018/10/12 13:35   文本文档
apollo             2016/5/17 13:41    文本文档
apollo1            2018/10/12 13:18   文本文档
l                  2016/5/17 13:41    文本文档
p                  2016/5/17 13:41    文本文档
t                  2016/5/17 13:41    文本文档
war_and_peace      2016/5/17 13:41    文本文档

类型: 文本文档
大小: 14.1 MB
修改日期: 2018/10/12 13:18
```