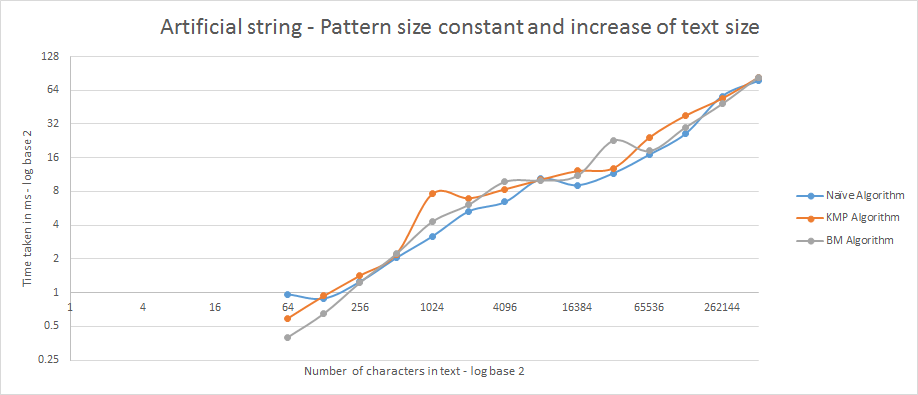
**Technical Explanation of Algorithms:**

1.



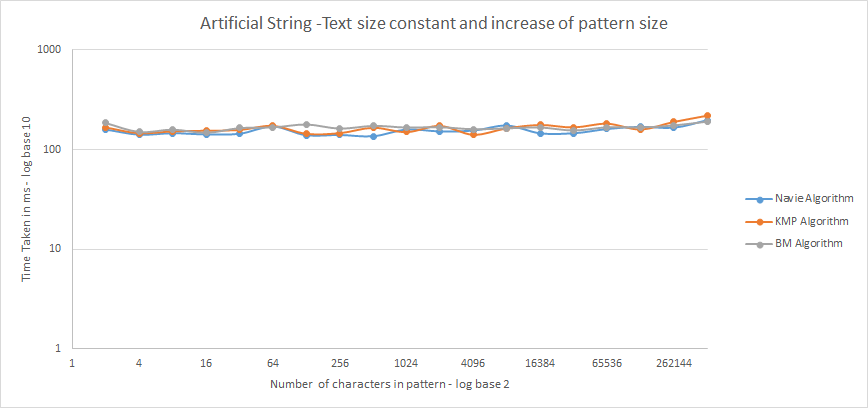
X axis of the graph is the number of characters in text, I started from 64 characters to 524288 characters. But making the pattern size constant at 16. As one can expect if the number of characters in the pattern then the comparisons increase and thereby increasing the run time. Since the increase in power of 2 we take log log graph to see the proportionate increase. If we compare the three algorithms

Question I want to answer is if I change the text size but making the pattern size constant and it resulted in the above graph

– Naïve Two for loops checking every thing

* KMP Algorithm Recomputed the pattern to check for repetition and use it to skip few comparisons
* BM Algorithm Compute the last occurrence of the alphabets and then compare from the back to skip few algorithms

Since this artificial generators only contains 1 and 0, we have repetitions in such a way that most of the characters need to be checked and also we need to pre compute the pattern. (mostly it will contain 0,1,2 not more than that) so skipping will be difficult in kmp. And in BM we go to last occurrence and since this only contains 0, 1 the last occurrence will be mostly at last which will not help much. Since naïve need no preprocessing and only contains 0, 1 it performs fairly okay. So for artificial strings all the algorithms work the same way since its repetition is not consistent and only contains 2 characters.

2.

X axis of the graph is the number of characters in pattern, I started from 2 characters to 524288 characters. But making the size size constant at 1048576.

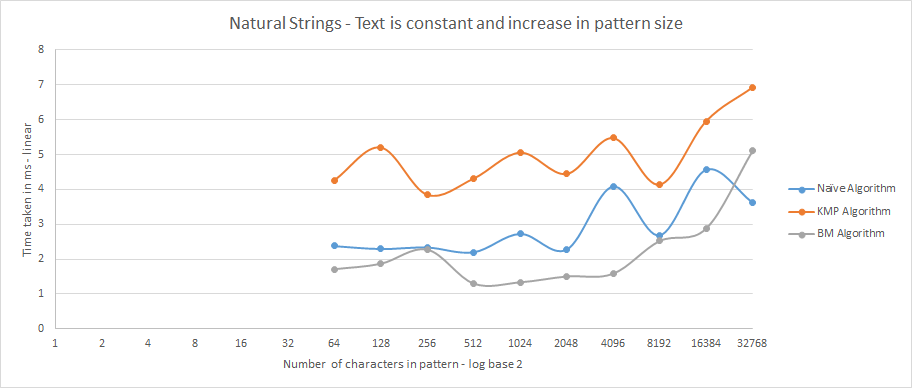
As one can expect if the number of characters in the pattern increase then the comparisons increase and thereby increasing the run time. Since the increase in power of 2 we take log log graph to see the proportionate increase. If we compare the three algorithms

Question I want to answer is if I change the pattern size but making the text size constant and it resulted in the above graph

– Naïve Two for loops checking every thing

* KMP Algorithm Recomputed the pattern to check for repetition and use it to skip few comparisons
* BM Algorithm Compute the last occurrence of the alphabets and then compare from the back to skip few algorithms

Since this artificial generators only contains 1 and 0, we have repetitions in such a way that most of the characters need to be checked and also we need to pre compute the pattern. (mostly it will contain 0,1,2 not more than that) so skipping will be difficult in kmp. And in BM we go to last occurrence and since this only contains 0, 1 the last occurrence will be mostly at last which will not help much. Since naïve need no preprocessing and only contains 0, 1 it performs fairly okay. So for artificial strings all the algorithms work the same way since its repetition is not consistent and only contains 2 characters and as the text remains same the number of comparisons remain the same more or less it need to check all the 1048576 characters (we skip few for BM and KMP but for artificial string the skips will be less). Even when the size of pattern changed it need to make need to compare all the text so for all pattern sizes the run time is almost same.

3.

Keeping text constant and increasing the pattern size.

Now let’s check for natural strings. For this I took DNA sequence from the links provided as resource in the assignment. It contains roughly 65536 alphabets that contain different dna sequences like “TACACCA” so we made Text as constant (number of characters is total number of characters in the dna file). We randomly select few character in the pattern in to match it with the text. So first we selected 64 characters randomly (consequent) then try to match it with the three algorithms – Naïve, KMP, BM algorithms. Then Increase it with power of 2 to match it with text. We use log log graph to accomidate the proportionate increase in both x axis and y axis

As one can observe from the graph, BM takes least time of all and KMP takes the most time, I am guessing this is due to preprocessing takes more time. Naïve takes average time. But since the naïve algorithm works for O(n2) and kmp takes O(n+m), naïve should take more time but results are interestingly different and BM takes O(nm) and best case would be O(n/m). And more surprisingly at the maximum number of character naïve performed better but overall BM performs best followed by naïve followed by KMP.