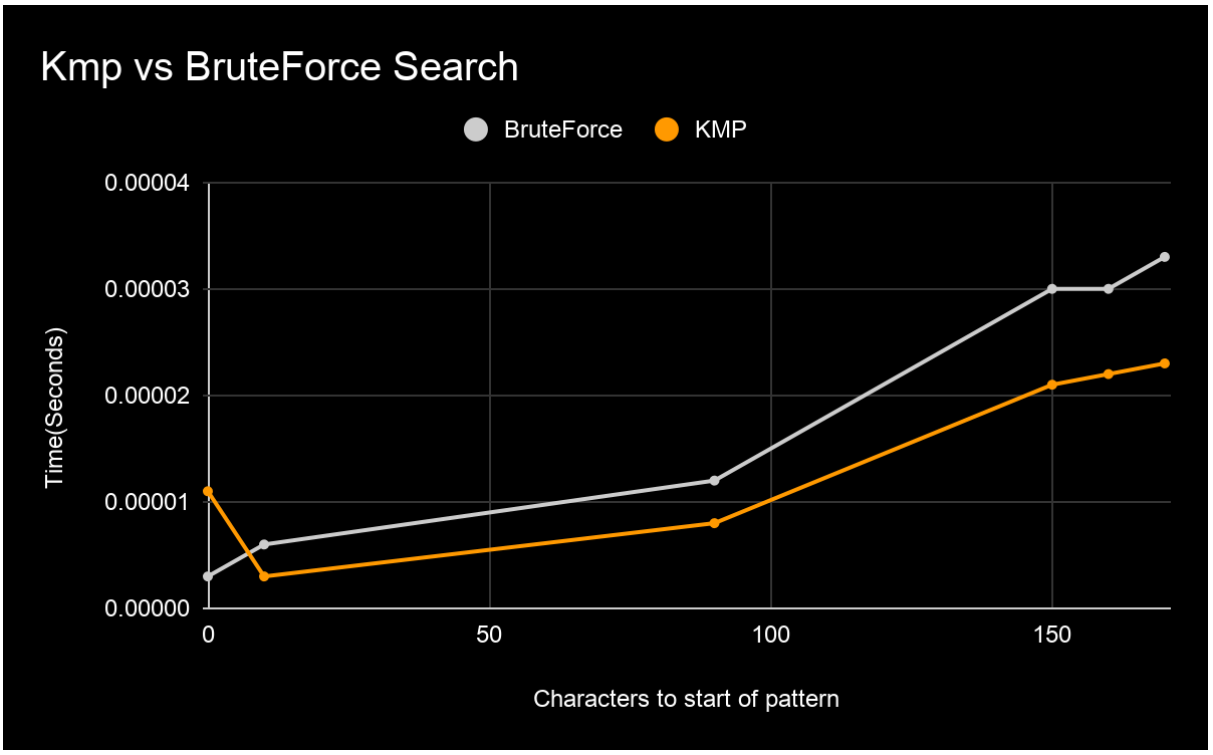
SubString Search Analysis



(Characters to start of pattern means how far into the text before we find the pattern)

The above graph illustrates the relationship between the kmp algorithm and the bruteForce method of searching Strings. We can assume once again from our graph that the kmp algorithm runs with a time complexity of O(n+m) and that brute force runs with a time complexity of O(n*m) (where n is text length and m is pattern length).

We can also see in the above example Kmp was slower in one test case, this was mainly due to the overHead when generating the LPSArray allowing bruteforce to finish before array generation.

To demonstrate why kmp and bruteForce search are different we consider the pattern acacagt in acatacgacacagt.

For brute force everytime we find a run of characters which are in our pattern that is not the full pattern, we increment j until the first discrepancy then we have to start again from 0 at the next character. In case of acat and acac brute force will reach t and be forced all the way back to acac meaning that this search results in O(n*m).

Kmp however generates a lps table which finds the longest common suffix prefix of the pattern, while this causes some overhead it allows traversal of the string without reversing.

LpsArray:

a	c	a	c	a	g	t
0	0	1	2	3	0	0

If we look again at the case of reaching t after aca we look at our prefix table at c and jump back 2 indices. This ensures that kmp will run in time complexity of O(n+m) as the index never moves back in the string.