Step 3 Benchmark the performance of your two algorithms

Modify your algorithms to take different inputs (either within your program or by reading in text files). Run your algorithm looking for patterns in various sizes of texts. Observe, note and graph the performance of your algorithms.

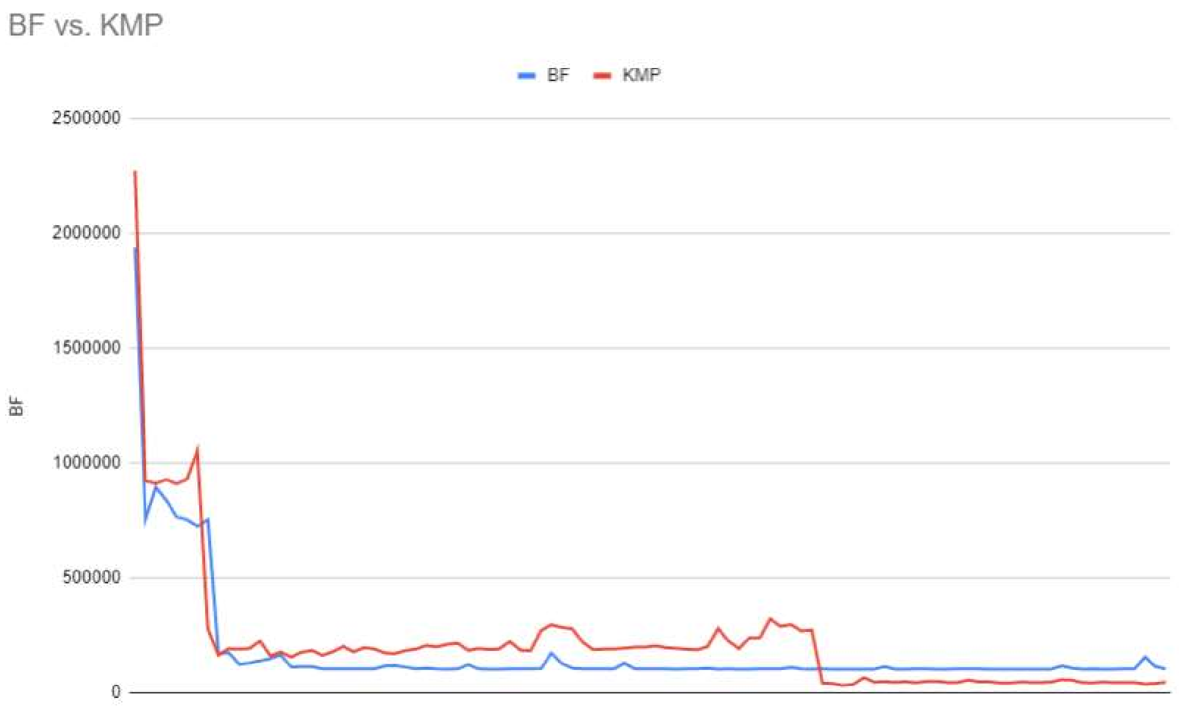
Q. 1 What would you say the complexity of the Brute Force substring search algorithm is?

The Brute Force substring search algorithm has complexity O(m * n), where m is the pattern length and n is the text length. This is because, in the worst case, it will scan each position in the search text for the pattern.

Q. 2 What would you say the complexity of the KMP algorithm is?

KMP is an O(n) algorithm where n is the text length. This is assuming the lps[] array was pre-computed for whatever pattern is being searched for, which is often the case when scanning large collections of text, such as a database.

Please find below a graph comparing the Brute Force search and the KMP search on a variety of inputs.



As can be seen from the above graph, the running time of each is almost exactly the same in almost all cases. However, this is not proof that the algorithms are at all comparable in terms of their Big-Oh complexity. Unfortunately, because modern systems have become so powerful, it is very difficult to test each of these algorithms with sets of inputs which will adequately challenge them. This is because, in order to really put these algorithms against each other, thousands of input strings would be necessary, all designed in a particular way so as to highlight specific shortcomings in each algorithm. Much like during Practical one, when testing the Russian Peasant's algorithm, it is simply the case that modern machines are too computationally powerful in order for the algorithm to be tested on them without using excessive amounts of testing data. Unfortunately, this means that I cannot experimentally verify my above estimates in terms of growth rate.