**Part 1: Comparison of run-times of Rabin Karp, Knuth Morris Pratt and Bayer Moore Algorithms**

Compared to other two, the Rabin Karp has its uniqueness in the use of Hash operation. This operation has a time complexity of O(M) for the pattern (P) to be searched. The run-time of Rabin-Karp in average and best case is O(N+M), but its worst-case time is O(NM) which occurs when all characters and the hash values of all the sub-strings of pattern and text are exactly the same. On the other hand Knuth Morris Pratt algorithm pattern search has a time complexity of O(N) and is linear even if there are multiple occurrences of the pattern in the string. The Boyer-Moore algorithm has a worst-case complexity of O(NM) for repetitive patterns unlike Knuth Morris Pratt because if all matches are required, the entire pattern would be re-compared after a complete match. The differences are more clearly visible from their run-time plots attached with the assignment.

**Note**:(N is the length of the text and M is the length of the pattern).

**Part 2: Comparison of run-times of Bit Oriented and Aho Corasick Algorithms**

The Bitap or Shift-And algorithm runs in O(NM) operations, no matter what the structure of the text or the pattern is like except it performs much better when applied to inputs over smaller alphabets. The overall run-time is higher compared to other algorithms(as visible in the assignment results) pertaining to the above reason. On the other hand, Aho Corasick is really handy when multiple searches need to be made as it involves preprocessing in order to build the automaton of all the keywords. Compared to a linear time searching algorithm like Knuth Morris Pratt, where the time complexity can be written as O(N\*K + M) for multiple searches, Aho-Corasick Algorithm finds all words in O(N + M + Z) time.

**Note**:(Z is the number of occurrences of words in text and K is the number of keywords or patterns).

**Part 3: Comparison of run-time of Suffix Tree Algorithm with that of algorithms in Part 1**

This algorithm involves Suffix Tree Construction as part of the preprocessing which takes O(N) time to build for a string of length N. The search for sub-string takes O(M) for a pattern of length M and then if there are Z occurrences of the pattern in the text, it will take O(Z) to find all those occurrences. The run-times of preprocessing and searching, in milliseconds, can be seen during the application run. Therefore, the overall pattern complexity is linear as O(M + Z). Suffix tree provides a strong preprocessing of the text and therefore is better to use when lot of queries have to be performed on the text where as for normal string matching and checking if s1 is a sub-string of s2, Knuth Morris Pratt is best out of the above 3 algorithms since it’s time complexity is linear. Also these algorithms prefer to preprocess the pattern rather than text like Suffix tree algorithm. Although, in terms of run-time, this algorithm does much better compared to the above 3 algorithms, its biggest problem is it high memory usage.