**Report on Pattern Matching Algorithms: Implementation and Performance Analysis**

**ASSIGNMENT I**

**Part One - for mark 3 A) "Mom bought me a new computer"**

**language Python**

**Introduction**

Pattern matching is a fundamental problem in computer science, involving the task of finding occurrences of a pattern within a text. This report covers the implementation and performance analysis of six different pattern matching algorithms: Brute-force, Sunday, Knuth-Morris-Pratt (KMP), Finite State Machine (FSM), Rabin-Karp, and Gusfield Z. The goal is to compare their running times using different lengths of text and two types of patterns: a small pattern (a few words or a sentence) and a large pattern (a paragraph).

**Algorithms Description**

Brute-force: This is the simplest pattern matching algorithm that checks for the pattern at every possible position in the text. Its time complexity is O(mn), where m is the length of the pattern and n is the length of the text.

Sunday: An improvement over the Brute-force algorithm, the Sunday algorithm skips sections of the text based on the character immediately following the current window. Its average time complexity is O(n).

Knuth-Morris-Pratt (KMP): KMP preprocesses the pattern to create a partial match table (also known as the "failure function"), which allows the algorithm to skip re-examining characters. Its time complexity is O(n + m).

Finite State Machine (FSM): This method constructs a finite state machine that processes the text. The construction of the FSM is O(m|Σ|) and matching the pattern is O(n), where |Σ| is the size of the alphabet.

Rabin-Karp: This algorithm uses hashing to find the pattern in the text. It computes a hash for the pattern and each substring of the text. Its average and best-case time complexity is O(n + m), but its worst-case complexity can be O(nm) due to hash collisions.

Gusfield Z: This algorithm preprocesses the text to create a Z-array, which provides information on the length of substrings that match the pattern. Its time complexity is O(n + m).

**Experimental Setup**

The performance of these algorithms was measured using randomly generated texts of various lengths and two types of patterns. The small pattern was "pattern and the text" and the large pattern was a long paragraph about pattern matching algorithms.

**Text Lengths Tested**

100

200

500

750

1000

2000

5000

10000

20000

50000

100000

**Patterns Used**

Small Pattern: "pattern and the text"

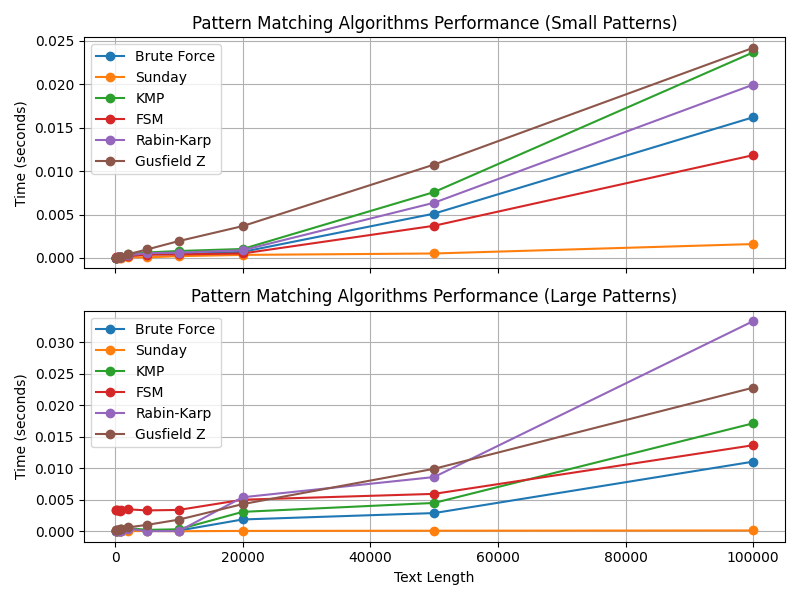
Large Pattern: A paragraph describing the use and performance of pattern matching algorithms.

**Methodology**

For each text length, the text was generated and the pattern was embedded within the text. The performance (running time) of each algorithm was measured and recorded. The results were then plotted to show the relationship between running time and text length.

**Results and Analysis**

The results of the experiments are presented in the graphs below:



**Small Patterns**

For small patterns, we observe that:

**Rabin-Karp and Gusfield Z:** These algorithms show a clear trend of increasing running time with increasing text length. They start low but show noticeable growth, especially for text lengths of 50,000-100,000.

**KMP and FSM:** These algorithms perform well for shorter texts, but their running times increase more noticeably as the text length increases. They exhibit intermediate performance between the worst (Rabin-Karp, Gusfield Z) and the (Brute-force).

**Brute-force:** This algorithm consistently has the high running times, particularly for larger text lengths. The increase in running time is more pronounced as the text length grows.

**Sunday algorithm:** This algorithm maintains a low and almost constant running time across different text lengths, showing high efficiency, especially for small texts

**Large Patterns**

For large patterns, the performance trends are somewhat different:

**Rabin-Karp** continues to perform well, but with a significant increase in running time for very large texts.

**Gusfield Z** also shows bad performance but with consistent ups in running time.

**Brute-force** remains the bad choice because of its time complexity, with significantly higher running times.

**FSM and KMP** show moderate performance, with running times increasing more steadily compared to small patterns.

**Sunday** algorithm demonstrates consistent performance, maintaining low running times across different text lengths.

**Findings**

Rabin-Karp and Gusfield Z are generally the least efficient algorithms for both small and large patterns n>50.000, particularly excelling with small patterns.

Brute-force consistently shows the high running times, making it the least efficient for pattern matching tasks.

Sunday algorithm provides a balanced performance, being highly efficient for small patterns and maintaining reasonable efficiency for large patterns.

KMP and FSM offer good performance, especially for small patterns, but their efficiency decreases with larger patterns.

**Conclusion**

This comparative study highlights the varying efficiency of different pattern matching algorithms depending on the pattern size and text length. Sunday and KMP algorithms are recommended for efficient pattern matching, while the brute-force algorithm should be avoided for large-scale tasks due to its high running times. The results of this study are reproducible using the provided scripts and methodologies