String-Search Algorithms Performance Comparison

The algorithms included are:

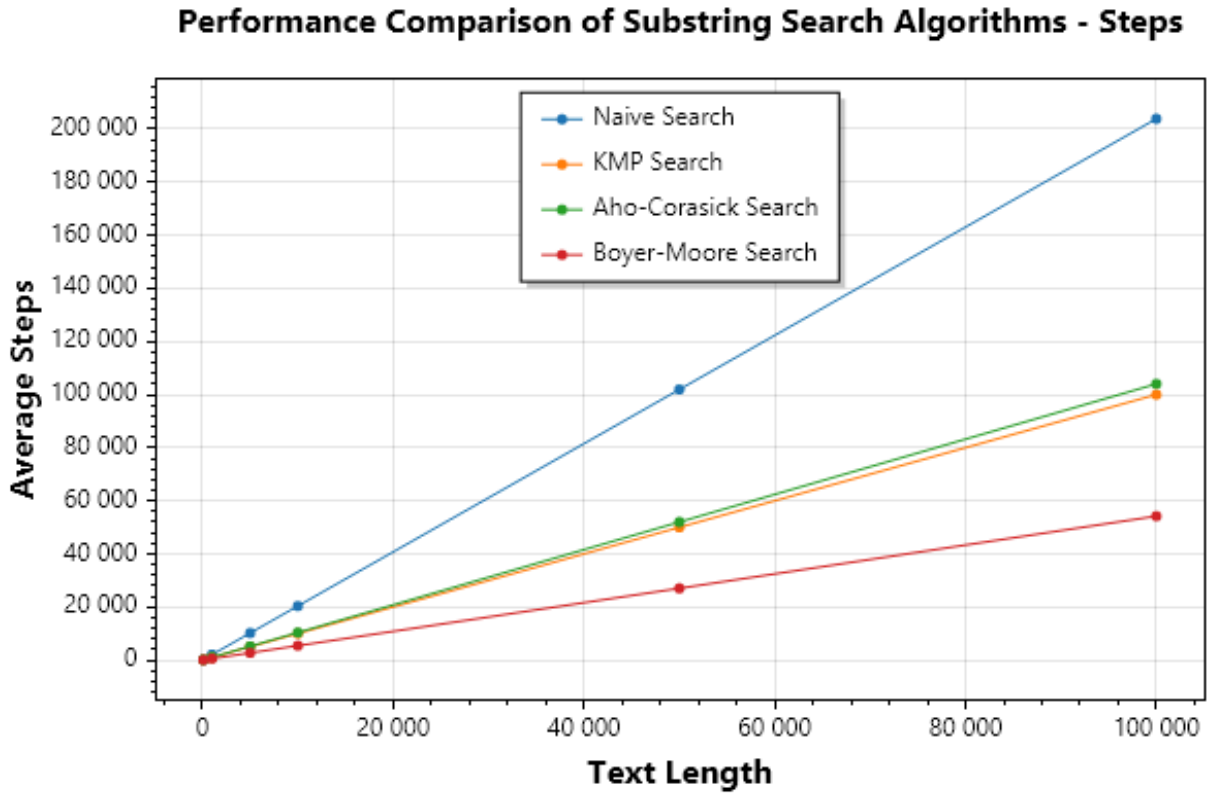
- Naive Substring Search
- Knuth-Morris-Pratt (KMP)
- Aho-Corasick
- Boyer-Moore

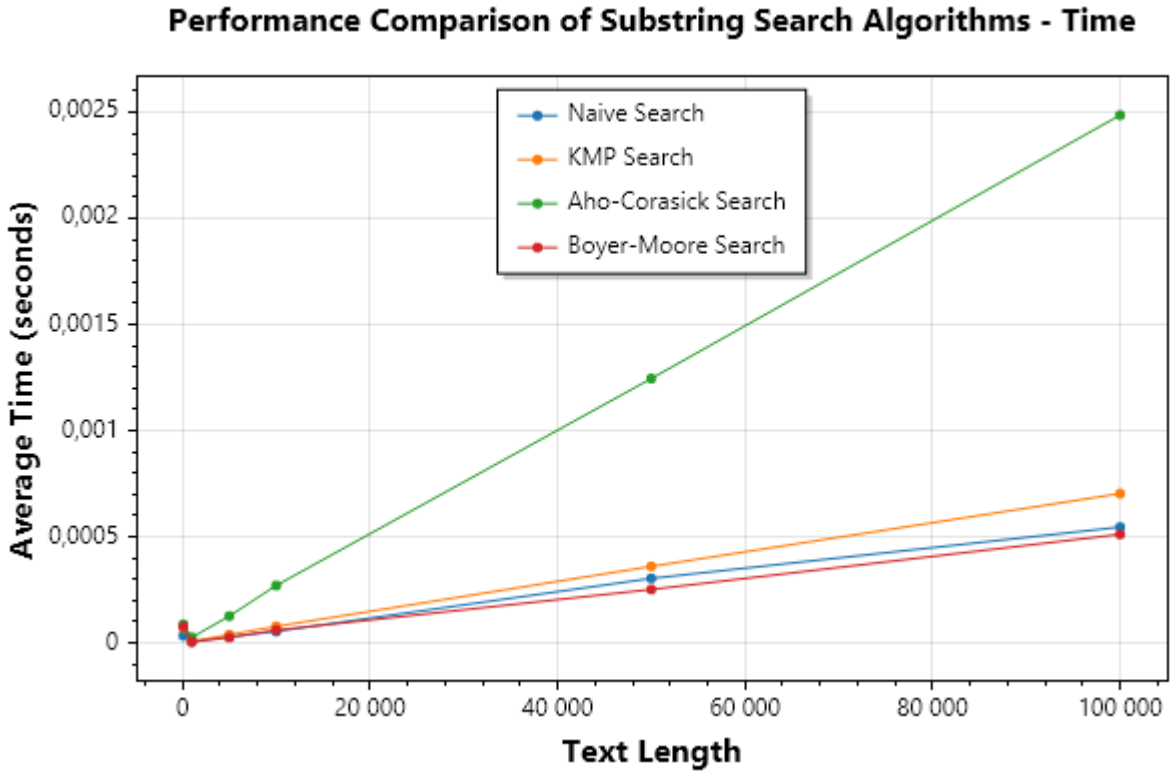
Testing Suites

Text Lengths: 100, 1000, 5000, 10000, 50000, 100000 characters

Pattern Length: 2 characters

Number of Trials: 5 per configuration

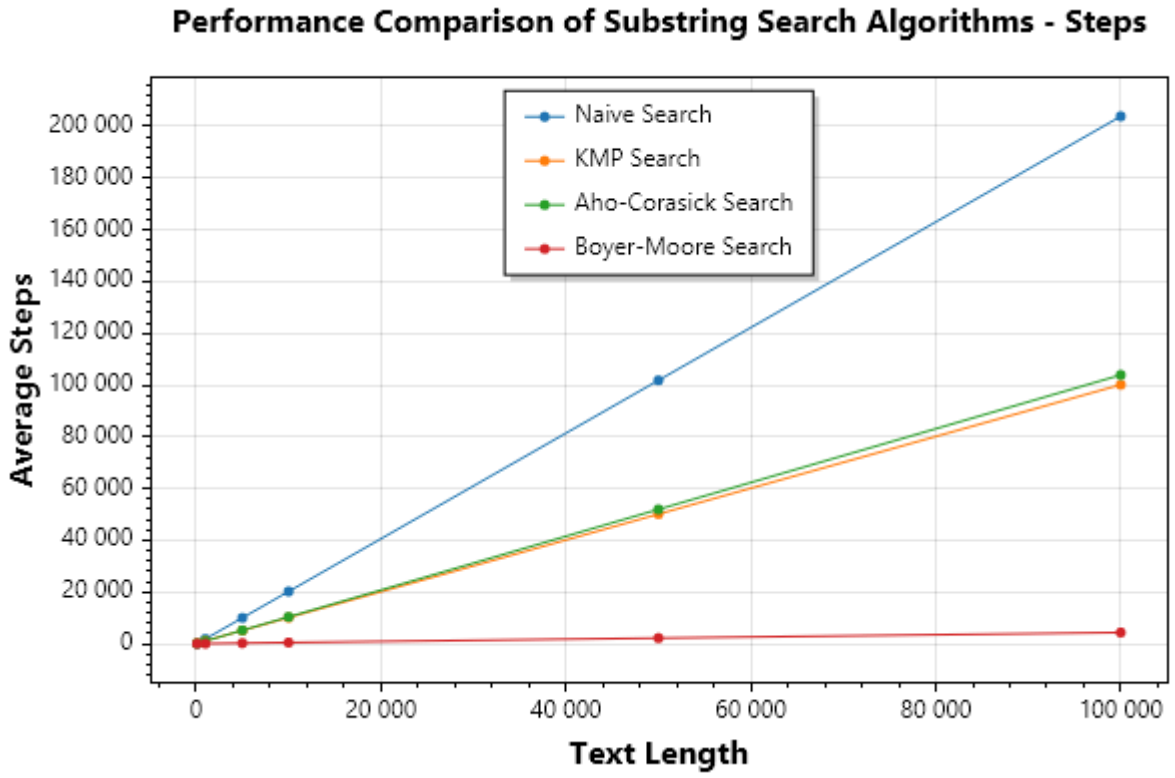


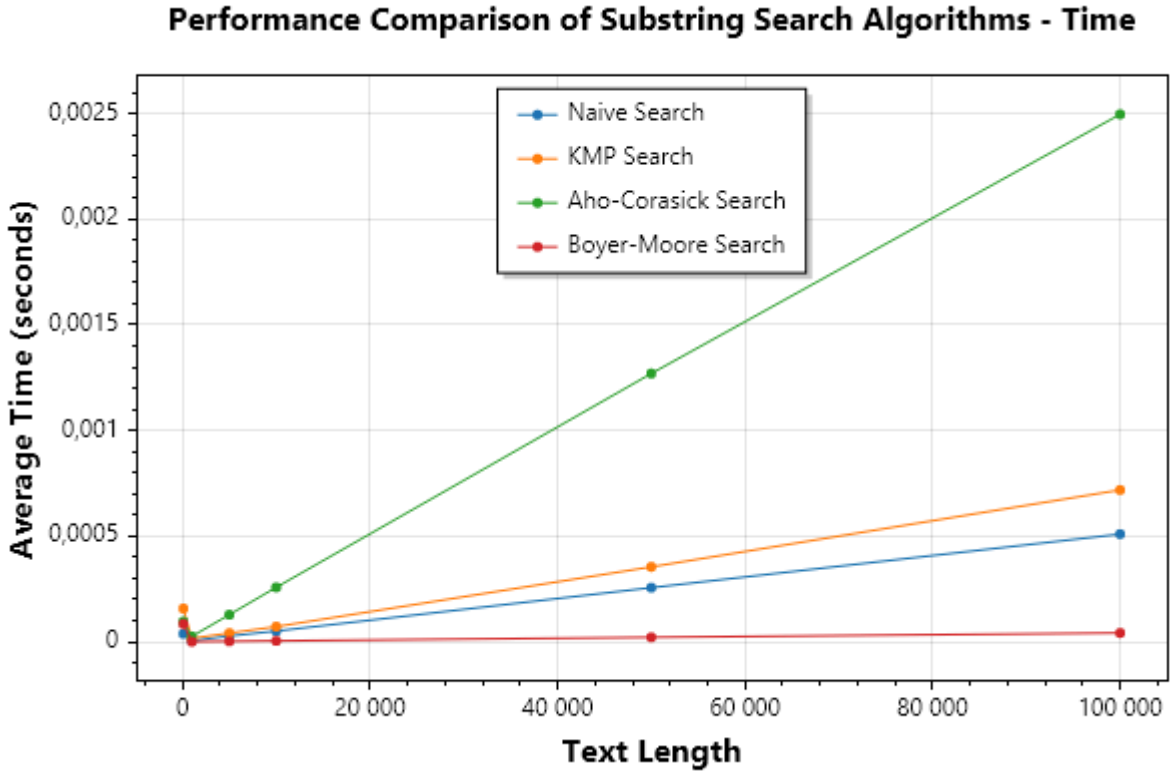


Text Lengths: 100, 1000, 5000, 10000, 50000, 100000 characters

Pattern Length: 100 characters

Number of Trials: 5 per configuration



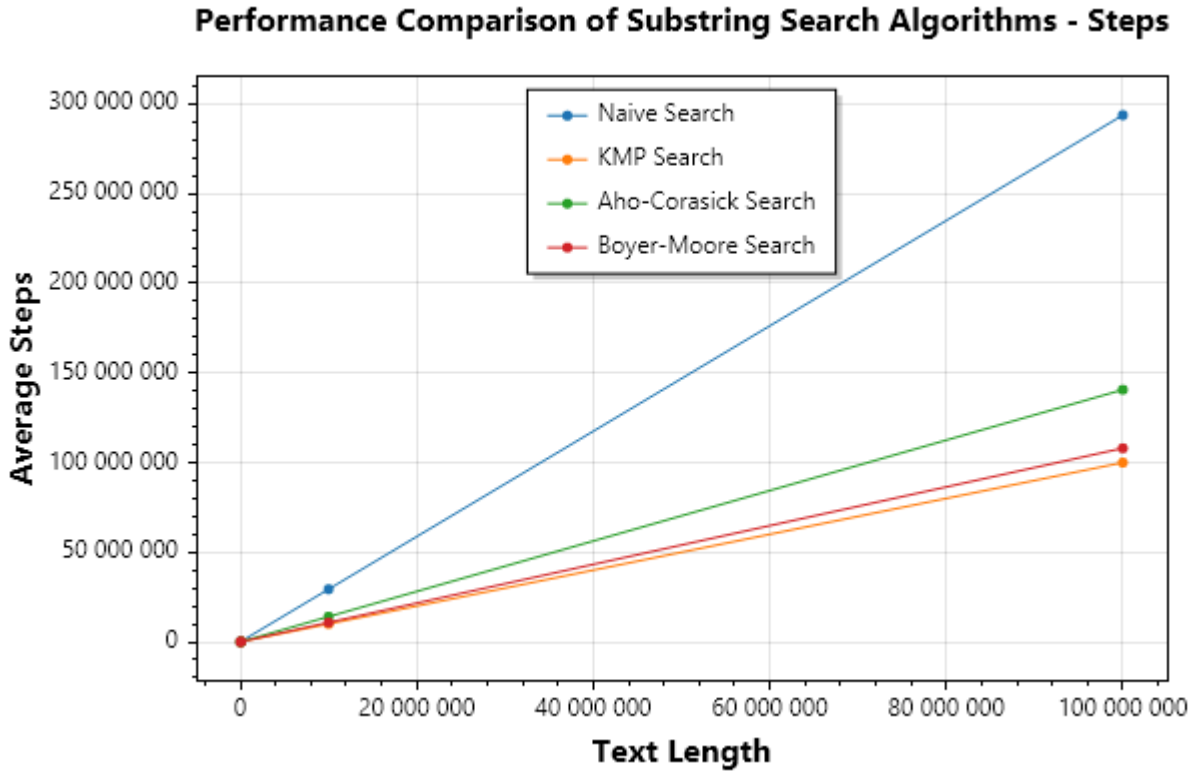


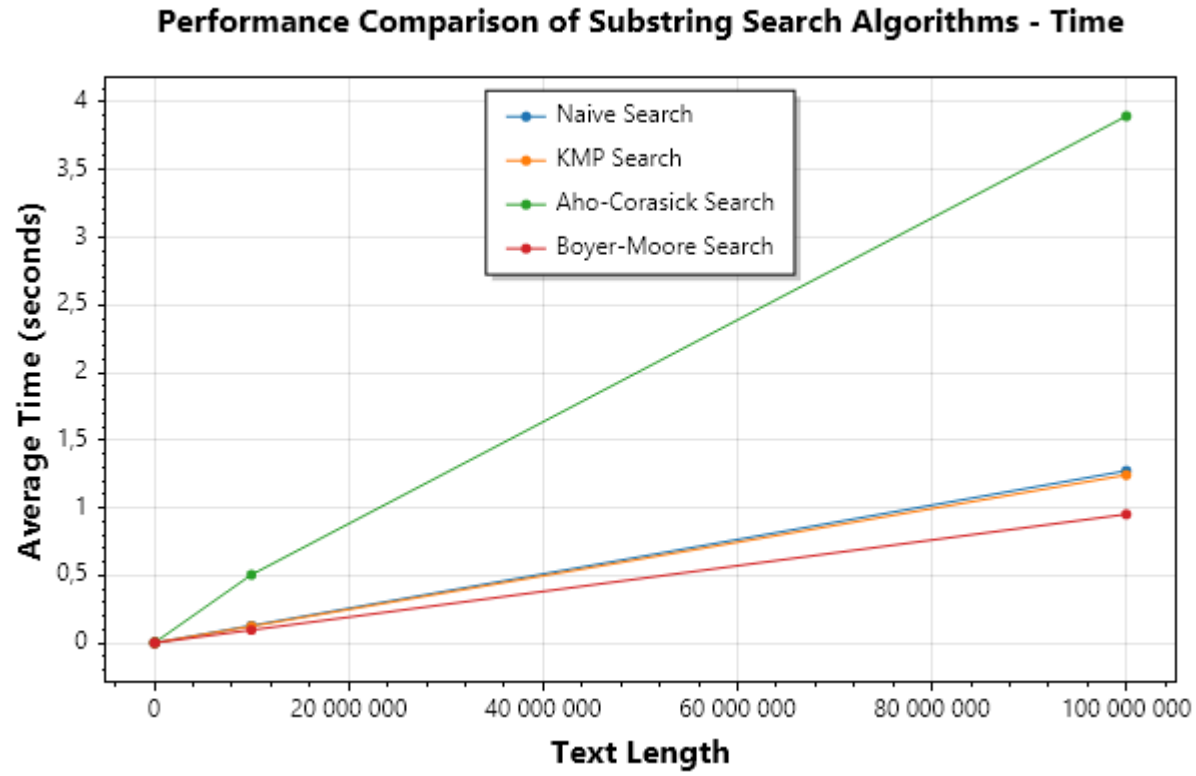
Text Lengths: 10.000, 10.000.000, 100.000.000, 1.000.000.000 characters

Pattern Length: 5 characters

Number of Trials: 1 per configuration

Alphabet: ‘ab’



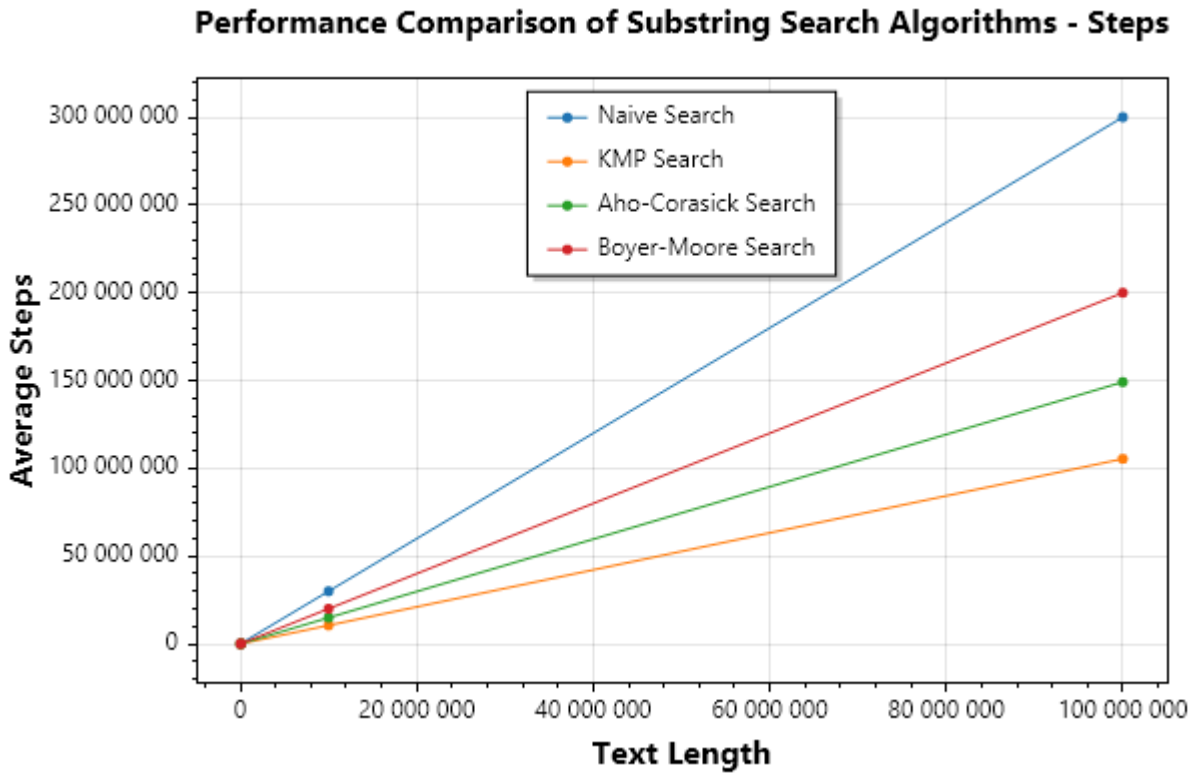


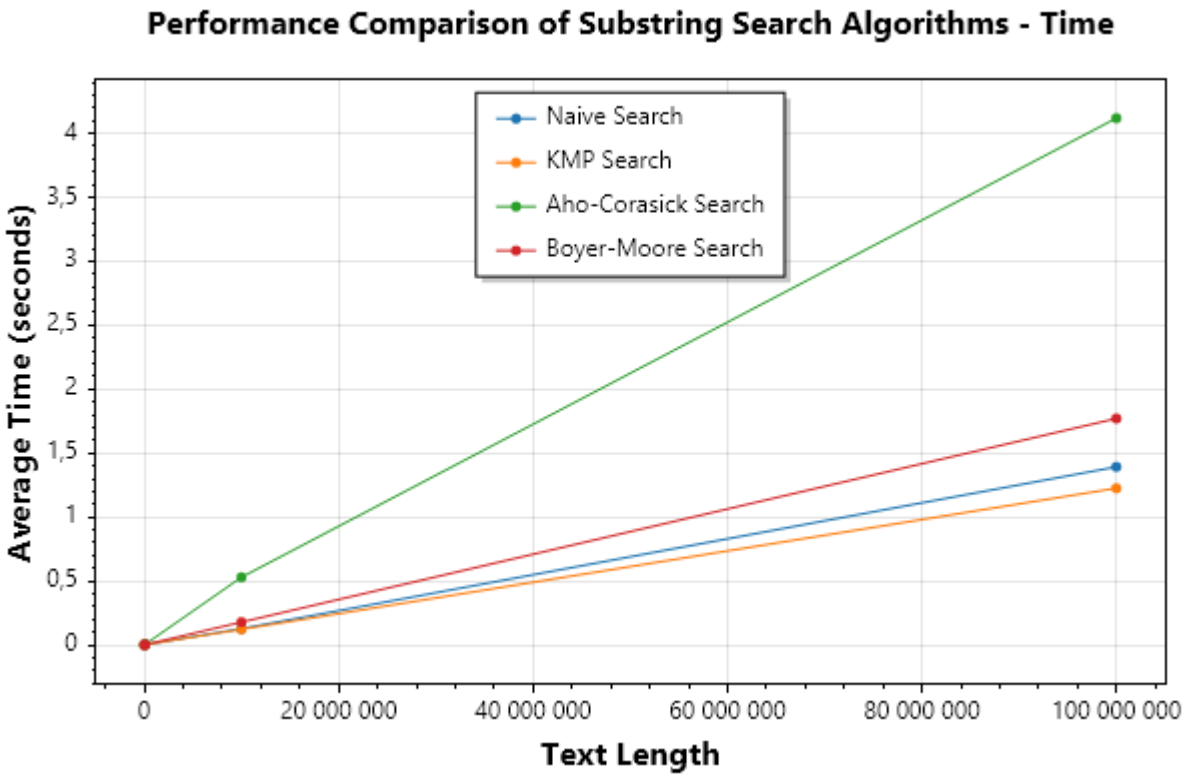
Text Lengths: 10.000, 10.000.000, 100.000.000, 1.000.000.000 characters

Pattern Length: 5000 characters

Number of Trials: 1 per configuration

Alphabet: ‘ab’





Complexity Analysis

1. **Naive Substring Search**:
    - **Steps**: The Naive Search has a worst-case complexity of O(n×m), where n is the text length and m is the pattern length, as it checks each position in the text for a potential match.
    - **Time Complexity**: The naive approach tends to perform worse with longer texts and larger patterns, which matches with increased execution time in tests with longer text lengths.

2. **Knuth-Morris-Pratt (KMP)**:
    - **Steps**: KMP optimizes by avoiding redundant checks, achieving a complexity of O(n+m). The preprocessing of the pattern to build the partial match table takes O(m), and the search through the text takes O(n).
    - **Time Complexity**: KMP's performance should remain consistent regardless of pattern length, as indicated by the tests showing improved performance in larger configurations compared to the naive approach.

3. **Aho-Corasick**:
    - **Steps**: Designed for searching multiple patterns, Aho-Corasick has a complexity of O(n+k), where k is the total length of all patterns. Building the automaton is typically more costly, but the search itself is linear with respect to the text length.
    - **Time Complexity**: Aho-Corasick's performance benefits from large patterns and texts, with results showing efficient performance in multi-pattern matching scenarios in large text sizes.

    (at least it is supposed to be so? because i took realization from ur python file and remade it in C#, maybe this strange time-consuming operations are the consequences of using C# structures?)

4. **Boyer-Moore**:
    - **Steps**: With a complexity averaging O(n/m)) but potentially O(n×m) in the worst case, Boyer-Moore is highly efficient when the pattern is long and contains fewer common letters with the text. The bad-character heuristics improve its performance significantly.
    - **Time Complexity**: The Boyer-Moore algorithm shows the fastest times for certain large text configurations, especially when the pattern is short.