

Marmara University   
Faculty of Engineering

CSE 2246

ANALYSIS OF ALGORITHMS



**Homework 1 Report**



Instructure: Ömer KOÇAK Date:17.05.2023

|  | Department | Student Id Number | Name & Surname |
| --- | --- | --- | --- |
| 1 | CSE | 150120012 | Kadir BAT |
| 2 | CSE | 150120055 | Muhammet Talha KARAGÜL |
| 3 | CSE | 150121021 | Feyzullah ASILLIOĞLU |

## Main Code

We have written this homework in JavaScript language. The program implements string-matching algorithms, namely Brute Force, Boyer-Moore, and Horspool. It performs a search for a specified pattern within an HTML file. The program first reads the HTML file and extracts the text from the body tag. It then applies each algorithm to find occurrences of the pattern in the text.

The code measures the execution time of each algorithm using timers and outputs the comparison count and occurrence count for each algorithm. For the Boyer-Moore displays a bad-symbol table and a good-suffix table. For Horspool algorithm displays a shift table.

Additionally, the code includes a marking process that highlights the occurrences of the pattern in the input text. It handles overlapping and non-overlapping occurrences differently and modifies the text by inserting HTML markup to visually represent the matched patterns.

Finally, the program saves the modified HTML file with the marked occurrences to an output file.

Overall, this code provides a practical implementation of string-matching algorithms, evaluates their performance, and offers a visual representation of the matched patterns within an HTML context.

## Brute Force Algorithm

We separated the brute force algorithm into its own function. The program compares each character of the pattern to the text's corresponding characters. It begins by initializing the comparisons and occurrence variables, including the text and pattern lengths, as well as empty arrays to record the indices where the pattern appears.

The program then uses a for loop to run over the text, testing each possible starting place for the pattern. A while loop within the loop checks the pattern and text characters at each place. If the characters match, the loop moves on to the next set of characters. With each comparison, the number of comparisons is increased. If the loop reaches the end of the pattern, it indicates that a match was discovered, and the occurrences counter is increased. The match's index is added to the indexes array. The algorithm provides an object holding the indices where the pattern appears, the total number of comparisons done, and the number of instances detected after iterating through all possible spots in the text. The brute force technique provides a basic but potentially inefficient way for string matching, especially for big texts or patterns, with a time complexity of O(mn), where 'm' is the length of the pattern and 'n' is the length of the text and a space complexity of O(1).

## Boyer Moore Algorithm

We also created a new function for Boyer-moore.To optimize the search process, the method employs two tables: the bad symbol table and the good suffix table. The maximum shift value for each character in the pattern is stored in the bad symbol database, indicating how far the pattern can be moved in the event of a mismatch with a given character. The excellent suffix table determines the shift based on the pattern's matching suffixes and prefixes, allowing for bigger shifts where possible.

The bad symbol table and the starting values for the good suffix table are created first by the algorithm. The pattern is then searched for inside the text by iterating over it from right to left. It checks the characters in the pattern with the corresponding characters in the text at each position, beginning with the rightmost character. If a match is detected, the method increases the occurrences counter, adds the current index to the indexes array, and calculates the necessary shift using the bad symbol table and the good suffix table values.

The algorithm runs until the end of the text or all occurrences of the pattern are discovered. Finally, it provides an object containing the bad symbol table, good suffix table, pattern occurrence indexes in the text, the total number of comparisons done, and a number of occurrences discovered.

The Boyer-Moore algorithm provides an efficient technique for string matching, particularly when the pattern and text are big, with a time complexity of O(mn), where 'm' is the length of the pattern and 'n' is the length of the text and a space complexity of O(m + n).

## Horspool Algorithm

The Horspool algorithm for efficient string matching is implemented in the given code. When there is a mismatch, the method employs a shift table to calculate the maximum shift value for each character in the pattern, indicating how far the pattern may be shifted. This shift table helps the algorithm bypass superfluous comparisons throughout the search phase, increasing its efficiency.

The code starts by building a shift table based on the pattern. The pattern is then found inside the text by iterating from right to left. Beginning with the rightmost character, it matches the characters in the pattern to the corresponding characters in the text.If a match is found, the procedure adds the current index to the indexes array and updates the search position depending on the shift value from the shift table for the character at the current location in the text.

The algorithm runs until the end of the text or all occurrences of the pattern are discovered. Finally, it produces an object containing the shift table, pattern occurrence indexes in the text, the total number of comparisons done, and number of occurrences discovered.

The Horspool algorithm provides an efficient technique for string matching, especially when working with bigger patterns and texts, with a time complexity of O(mn), where 'm' is the length of the pattern and 'n' is the length of the text, and a space complexity of O(m + n).

## Output.html File

The given code is an HTML page that shows the use of string-matching techniques to detect and highlight instances of a certain pattern throughout the document's body section. The code begins by scanning an HTML file and extracting the text from the body> elements. It then defines a pattern to look for within the text.

Three different string-matching algorithms are applied to the text: Brute-Force, Boyer-Moore, and Horspool. The code measures the execution time of each algorithm and outputs the number of comparisons made and the number of occurrences found.

After running the algorithms, the code alters the original HTML content by using the mark> HTML element to indicate the instances of the matched pattern. It also includes helpful sections that illustrate the results of each algorithm, such as the number of comparisons, occurrences, and time and space complexity.

After that, the updated HTML content is written to an output file that can be viewed in a web browser. This creates a visual representation of the matched patterns, making them easier to detect and evaluate within the original information.

In summary, the code shows how to utilize several string-matching techniques to discover and highlight occurrences of a pattern in an HTML page, making it a handy tool for pattern search and analysis inside textual material.



## Work Sharing

**Feyzullah:** HTML, brute force, and main code sections.

**Kadir:** Horspool algorithm implementation and report

**Muhammet:** Boyer-Moore algorithm implementation and report

### Feyzullah:

#### HTML:

Feyzullah worked on the HTML file by scanning it, extracting the content from the body element, and saving the amended HTML file with the noted occurrences.

#### Brute Force:

Feyzullah implemented the brute force algorithm, which compares each character of the pattern to the text's corresponding characters, records the indices where the pattern appears, and measures the execution time using timers.

### Kadir:

#### Horspool:

Kadir used the Horspool algorithm, which uses a shift table to determine the maximum shift value for each character in the pattern, allowing for quick string matching. Kadir also assessed the Horspool algorithm's execution time and gave the relevant output, such as the shift table, occurrence indexes, and comparison count.

### Muhammet:

#### Boyer-Moore:

Muhammet used the Boyer-Moore algorithm, which optimizes the search process for efficient string matching by using bad symbol and good suffix tables. Muhammet calculated the Boyer-Moore method execution time, gave the bad symbol and good suffix tables, and produced the occurrence indexes and comparison count.

## 

## 

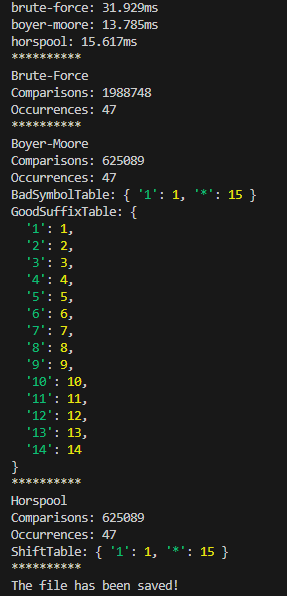
## 

## Illustrating and Analyzing the Results

### First example

Our pattern is: 111111111111111

Our sample file is: *sample1.html* (in zip file)



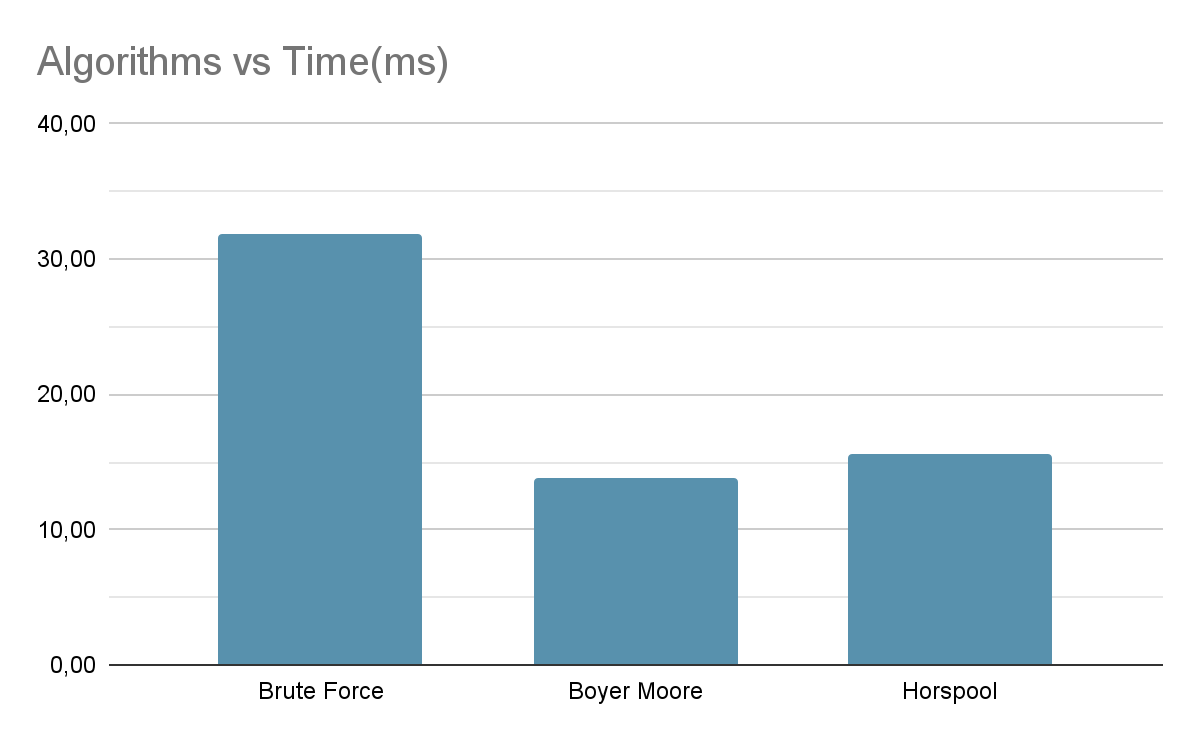
For bad-symbol table \* represents letters that are not in the pattern.

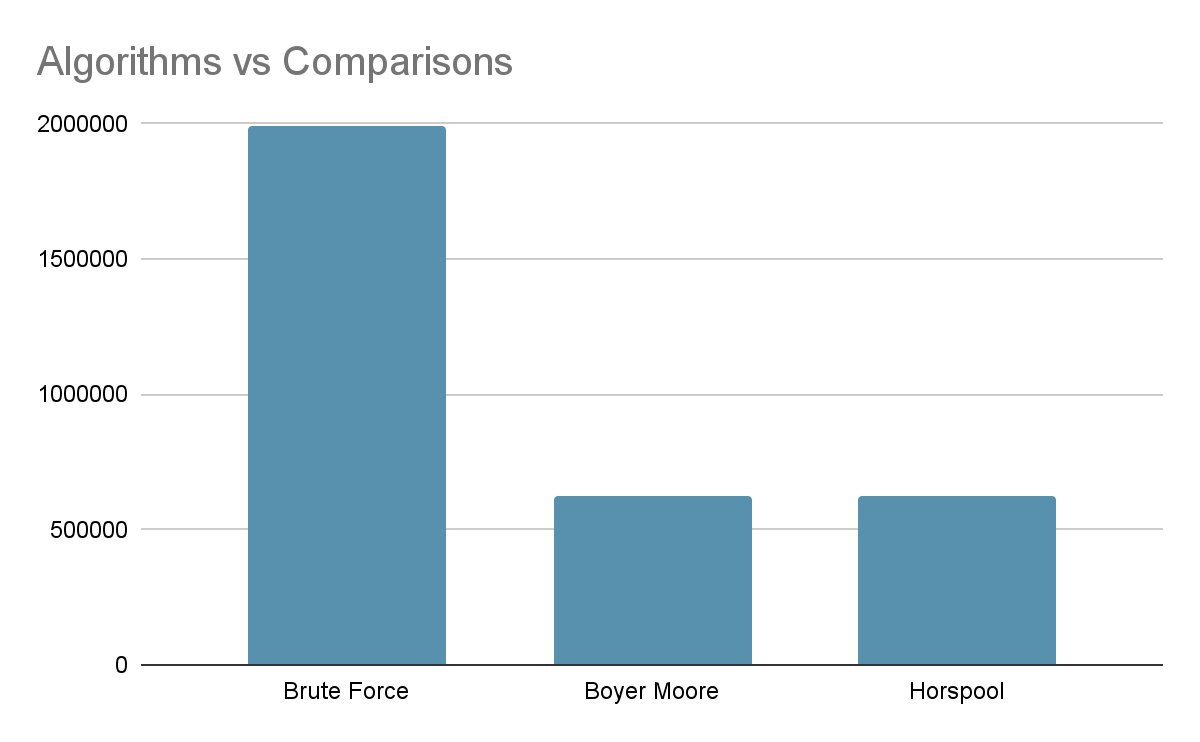
For shift table \* represents letters that are not in the pattern.

For a good-suffix table, green colored letters represent k. yellow colored letters represent d2.

Comparison:

Among the three algorithms, the Boyer-Moore algorithm proved to be the most efficient, taking only 13.785 ms to complete the search. It utilizes preprocessing and heuristics to optimize the search process, resulting in a significant improvement over the brute force algorithm, which took 31.929 ms. The Horspool algorithm, a simplified variant of Boyer-Moore, performed better than brute force but not as well as Boyer-Moore, with a time of 15.617 ms. Overall, Boyer-Moore demonstrated superior performance, showcasing its effectiveness in reducing search time by efficiently skipping unnecessary comparisons.

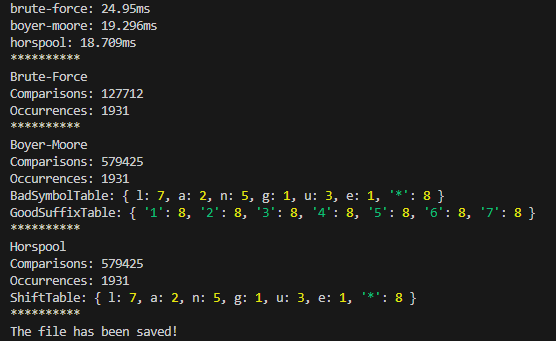




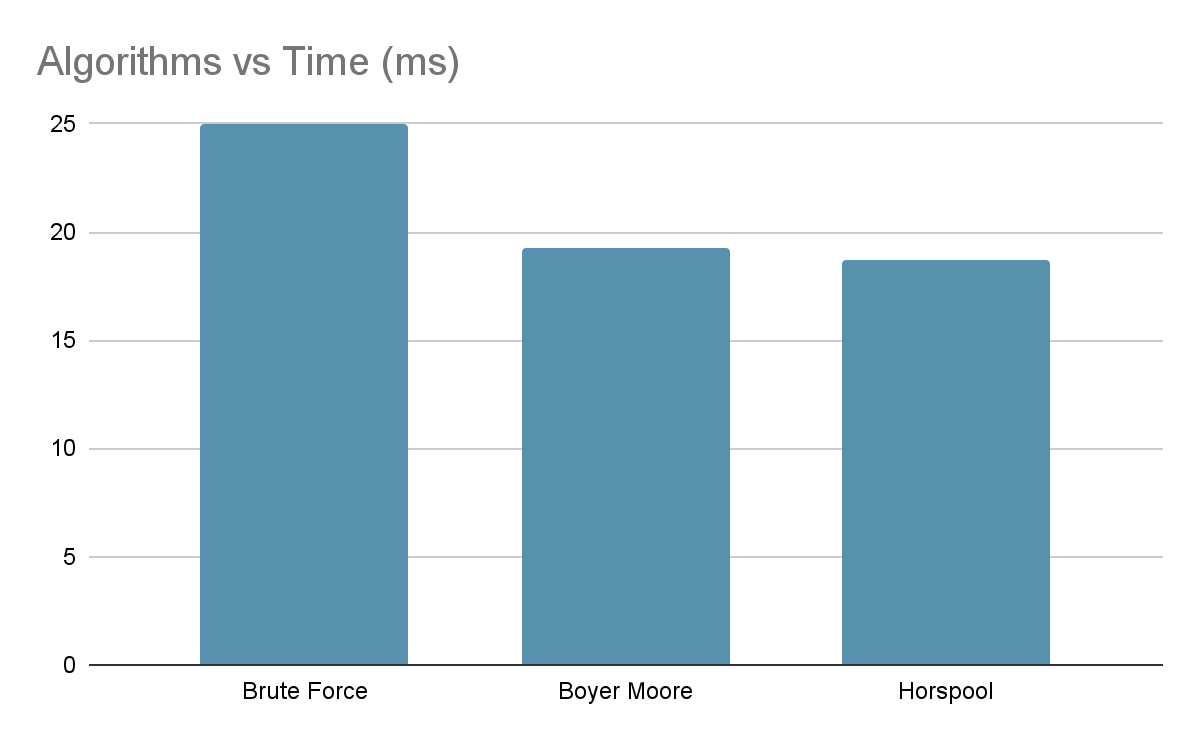
### Second Example

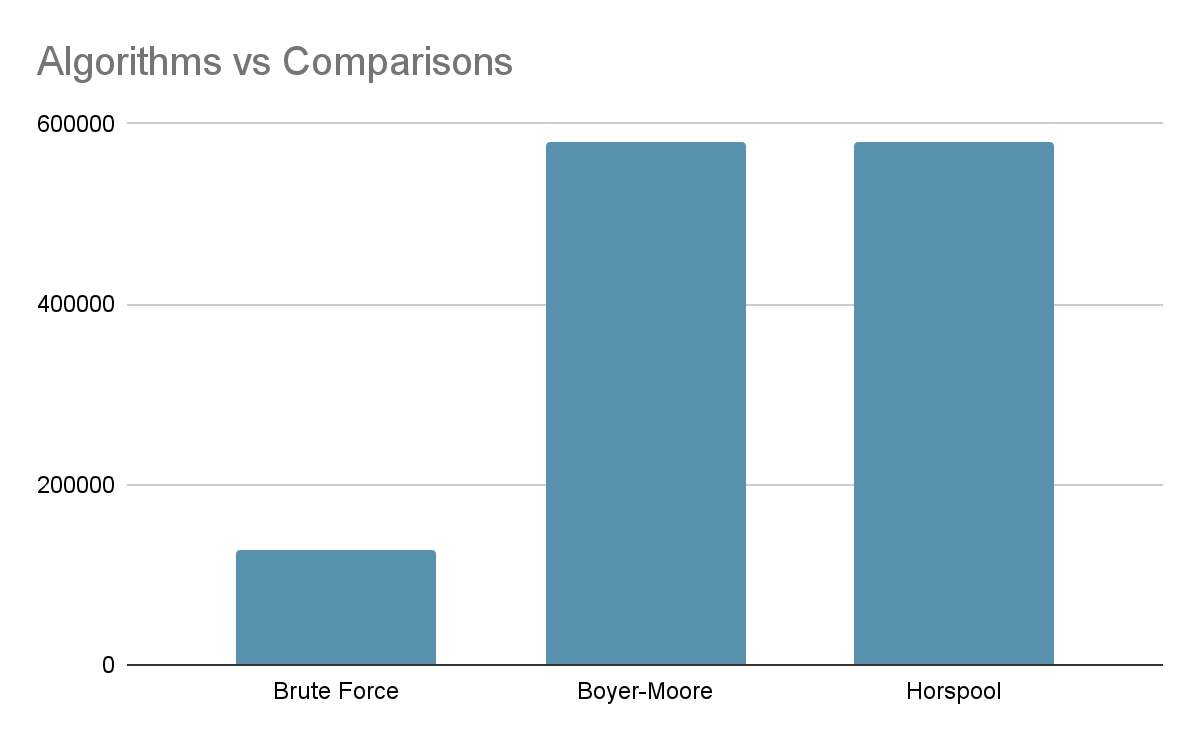
Our pattern is: language

Our sample file is: *sample2.html* (in zip file)



Among the three algorithms, the Horspool algorithm proved to be the most efficient, taking only 18.709 ms to complete the search. It utilizes preprocessing and heuristics to optimize the search process, resulting in a significant improvement over the brute force algorithm, which took 24.95 ms. The Boyer-Moore algorithm, a simplified variant of Horspool, performed better than brute force but not as well as Horspool, with a time of 19.296 ms. Overall, Horspool demonstrated superior performance, showcasing its effectiveness in reducing search time by efficiently skipping unnecessary comparisons.





For bad-symbol table \* represents letters that are not in the pattern.

For shift table \* represents letters that are not in the pattern.

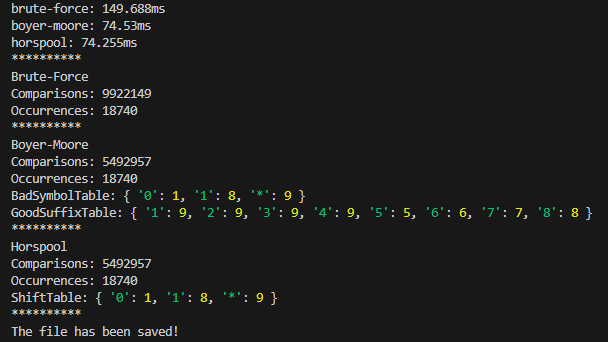
For a good-suffix table, green colored letters represent k. yellow colored letters represent d2.

### 

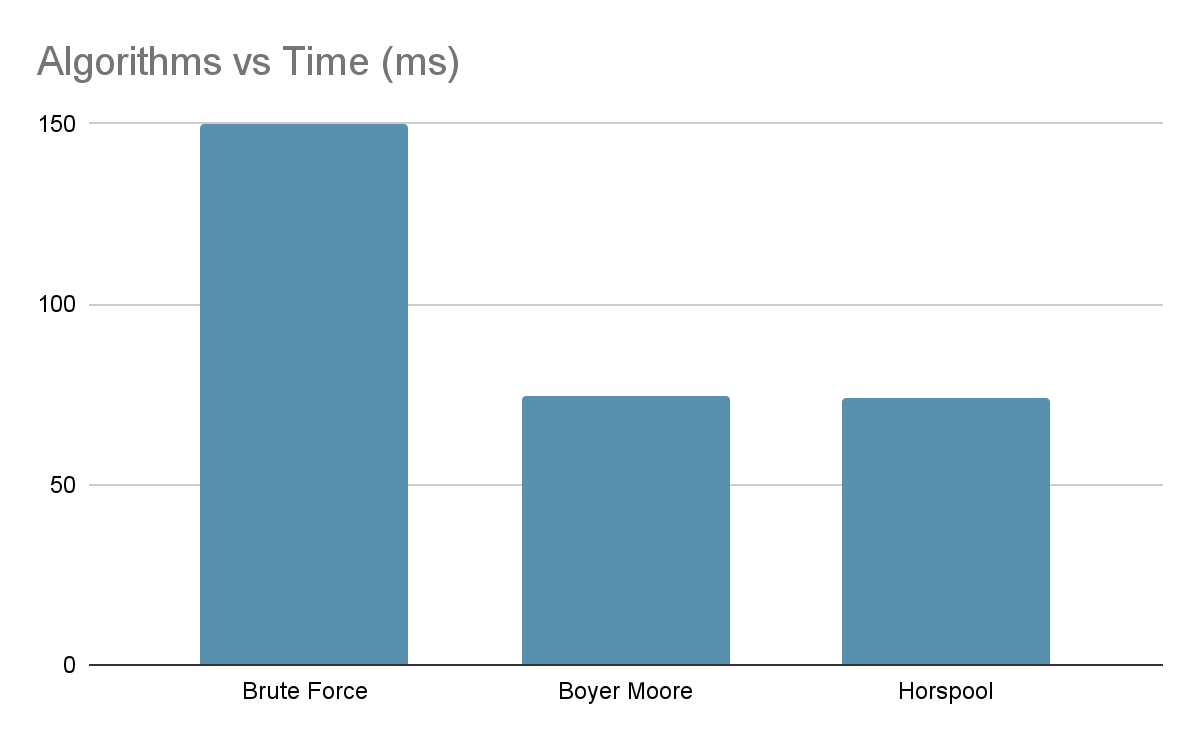
### Third Example

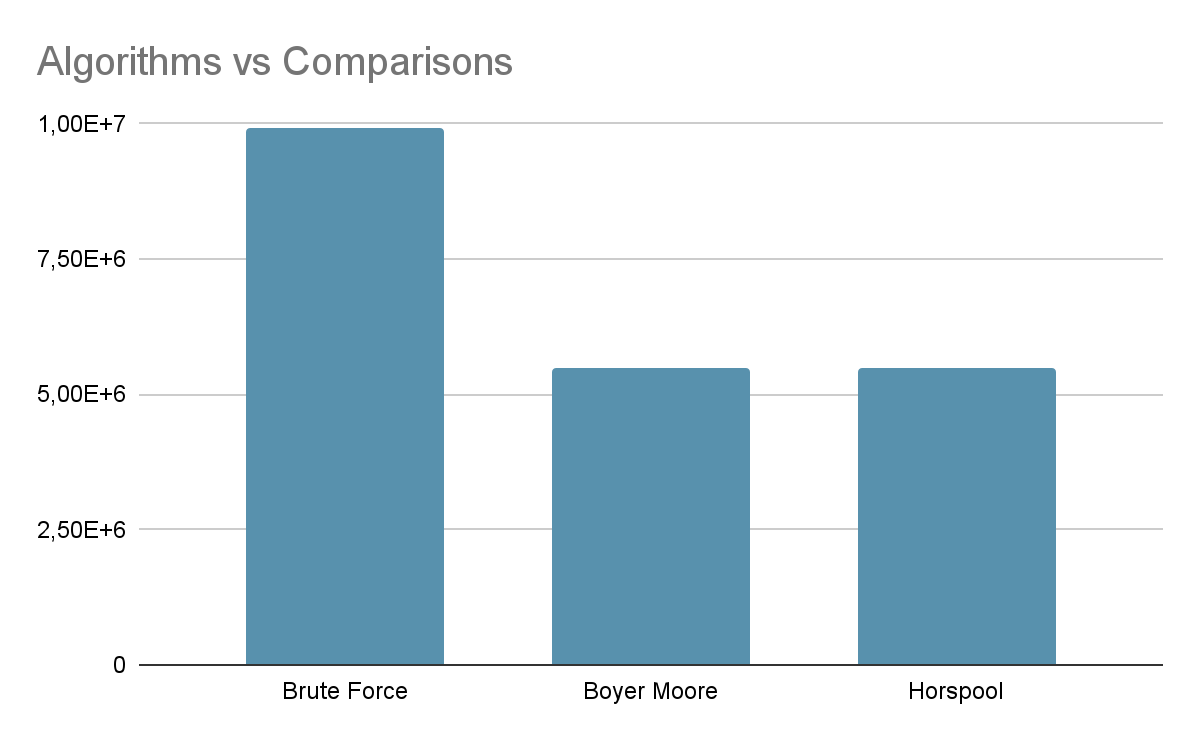
Our pattern is: 100000000

Our sample file is: *sample.html* (in zip file)



Among the three algorithms, the Horspool algorithm proved to be the most efficient, taking only 74.255 ms to complete the search. It utilizes preprocessing and heuristics to optimize the search process, resulting in a significant improvement over the brute force algorithm, which took 149.688 ms. The Boyer-Moore algorithm, a simplified variant of Horspool, performed better than brute force but not as well as Horspool, with a time of 74.53 ms. Overall, Horspool demonstrated superior performance, showcasing its effectiveness in reducing search time by efficiently skipping unnecessary comparisons.





For bad-symbol table \* represents letters that are not in the pattern.

For shift table \* represents letters that are not in the pattern.

For a good-suffix table, green colored letters represent k. yellow colored letters represent d2.

Overall, the best algorithm is Boyer Moore then Horspool algorithm. The worst algorithm is Brute Force algorithm.