Homework 1

CSE 2046/2246 Analysis of Algorithms, Spring 2023

Group members

| Name and Surname | Number |
| --- | --- |
| Muhammed İkbal Aktaş | 150119710 |
| Emre Demir | 150119808 |
| Murat Tüzün | 150119633 |

**Division for group members**

Muhammed ikbal aktaş: Horspool algorithm, report, arrange bit string files

Emre Demir: Boyer-Moore algorithm, report, finding html files

Murat Tüzün: Brute force algorithm, report, finding html files

written language:java-18.0.2

written ide: vscode

**Introduction**

In this homework we will try to compare three different algorithms on various html files with different patterns and we will measure the relations betweens patterns text length and we will compare algorithms.

**1-)Designing the Experiment:**

**(a) Deciding on / generating long HTML files**

We found HTML file from wikipedia;

* Hindi\_songs.html

<https://en.wikipedia.org/wiki/List_of_Hindi_songs_recorded_by_Asha_Bhosle>

* 2022\_in\_science.html

<https://en.wikipedia.org/wiki/2022_in_science>

* List\_of\_battles.html

<https://en.wikipedia.org/wiki/List_of_battles_by_geographic_location>

* List\_of\_gunsmoke.html

<https://en.wikipedia.org/wiki/List_of_Gunsmoke_(TV_series)_episodes>

* Municipal\_history.html

<https://en.wikipedia.org/wiki/Municipal_history_of_Quebec>

* regnal\_list.html

<https://en.wikipedia.org/wiki/1922_regnal_list_of_Ethiopia>

* Statutory\_rules.html

<https://en.wikipedia.org/wiki/List_of_Statutory_Rules_and_Orders_of_Northern_Ireland>

bit\_input\_1.htlm

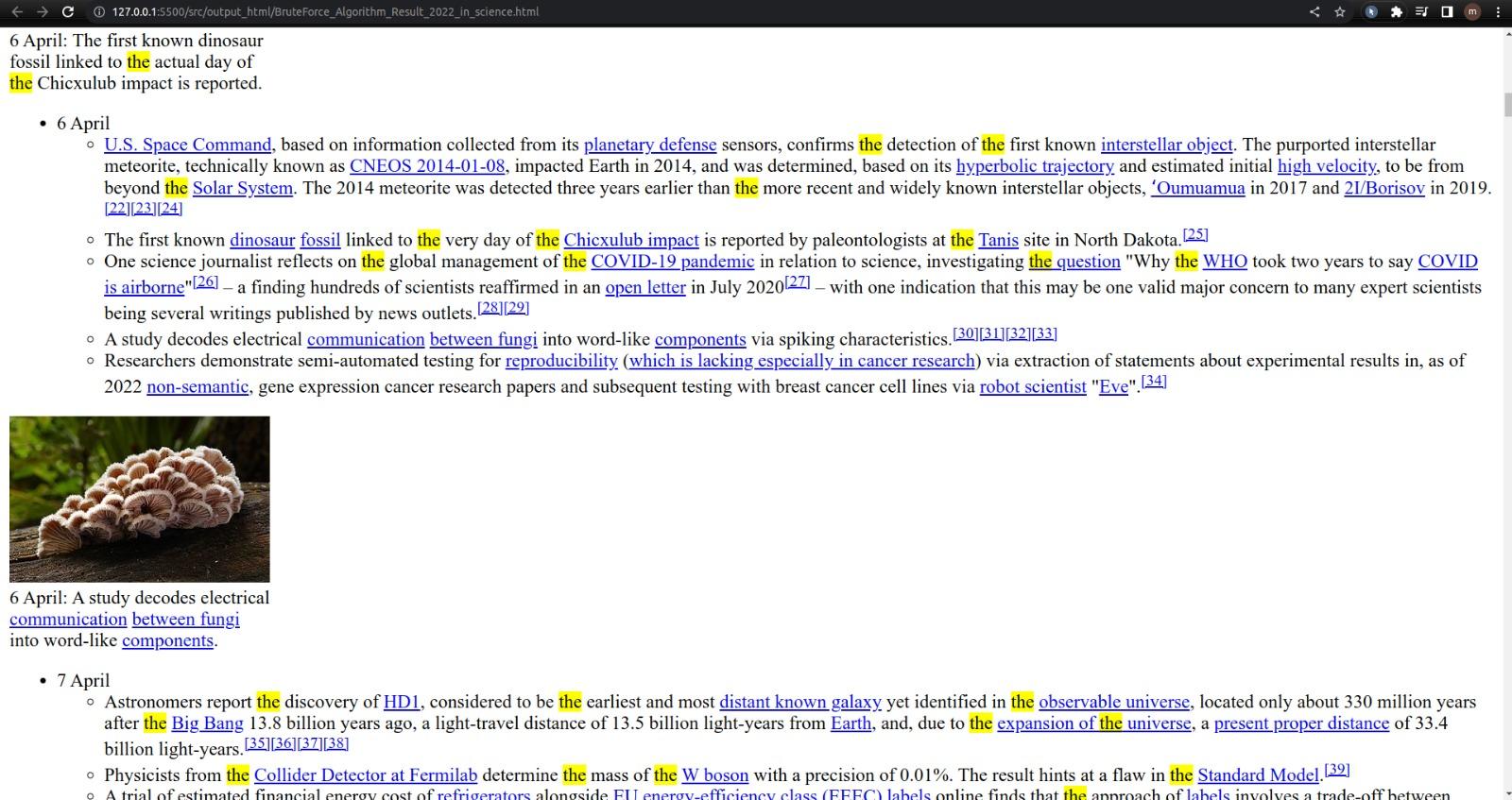
bit\_input\_2.htlm

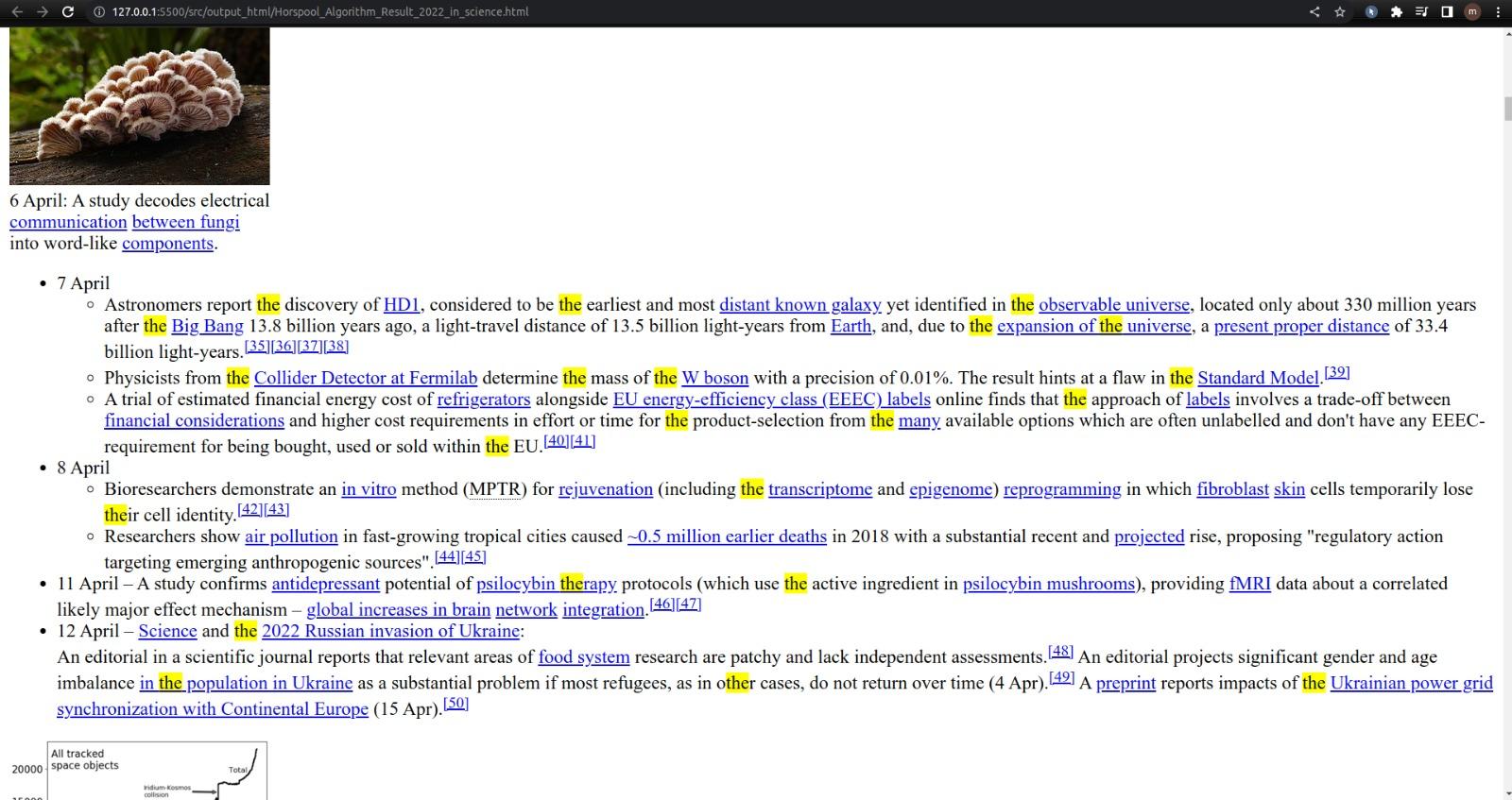
bit\_input\_3.htlm

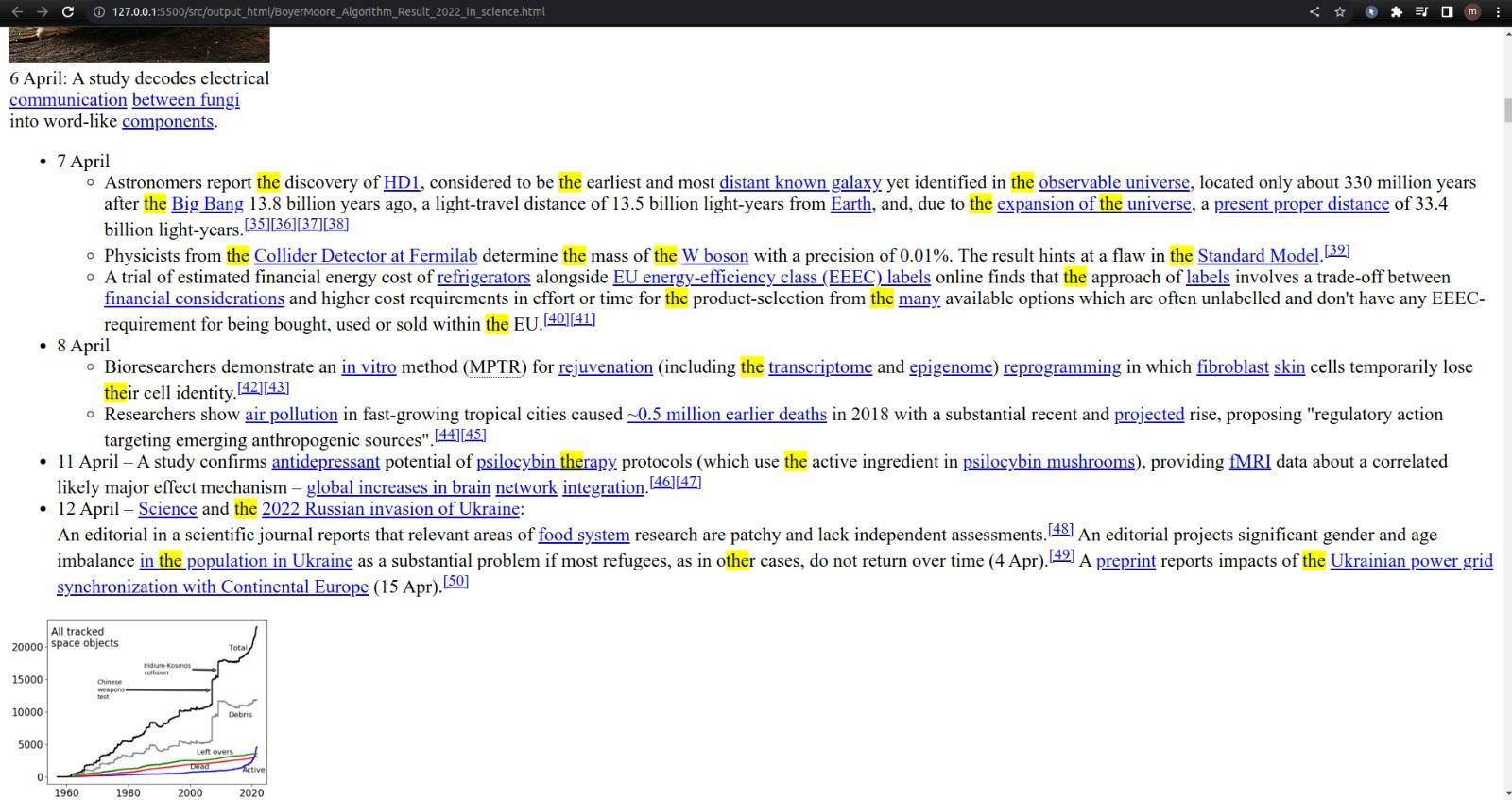
* Firstly, we manually created 3 bit string html files.
* Then, we converted some of these files to bit format from texts thanks to bit to string site (<https://codebeautify.org/string-binary-converter>)
* Finally, we copy and pasted these texts and obtained a file over one megabyte.

**(b) Deciding on patterns.**

We generally searched for Patterns in 3 types that are less, often and more often. We also examined 3 different patterns to observe the effect on the execution speed. We chose the pattern that could be found the most in the first pattern file, and as a reference, we chose the often and less patterns. For some files, we chose a pattern with a long but low execution speed. For example, We chose a pattern that is 472 characters long but has only one in the 2022\_in\_science.html file.

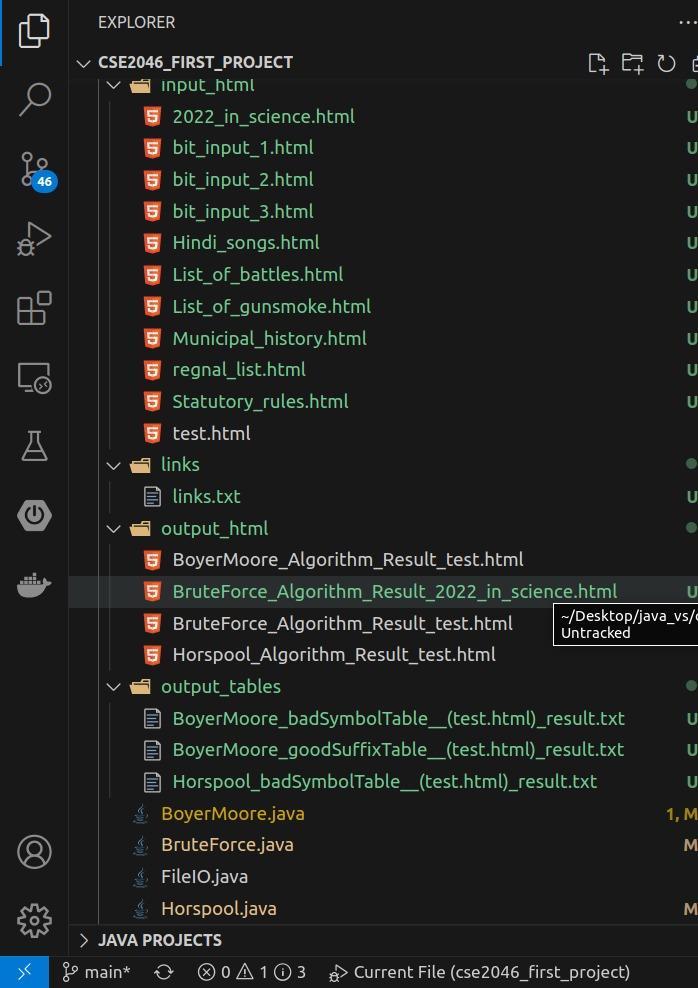






**2-)Explanation of program structure and execution:**

**(a) Bad symbol table and good suffix table for each pattern**



All outputs are in the output file and file names are stored as algorithm name + operation name + result.txt for bad symbol table and good suffix table for html result files are stored as algorithm name + “Algorithm” + “Result” + input file name.html as seen photo.

**(b) Output HTML file with highlighted pattern occurrences**

We both tested the test.html file in the submitted pdf as described in the pdf, and also tested it with the html file we found on the internet.

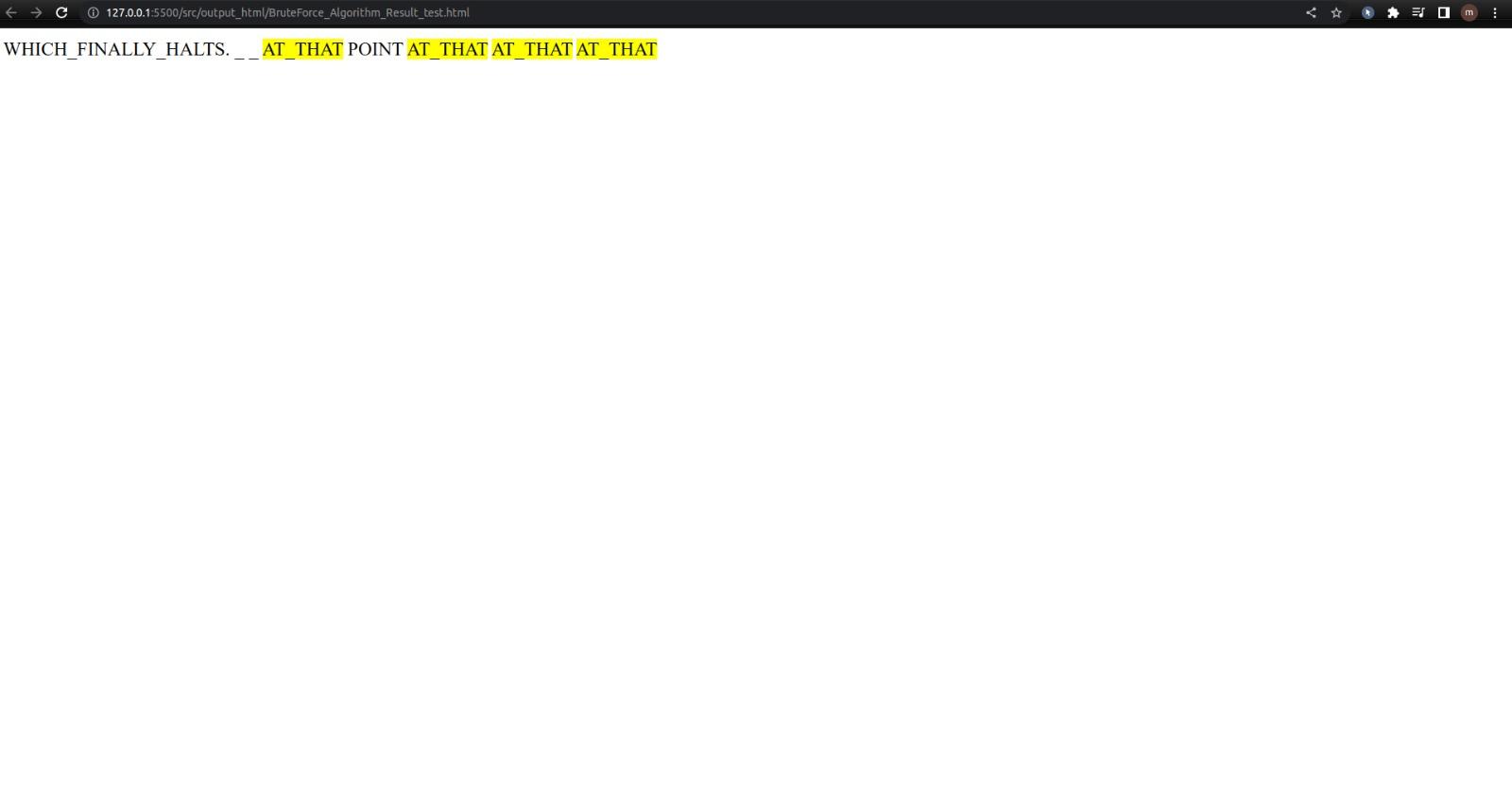
The output of html files are stored in “src/output\_html” path.

We took screenshots for both and added them to the report;

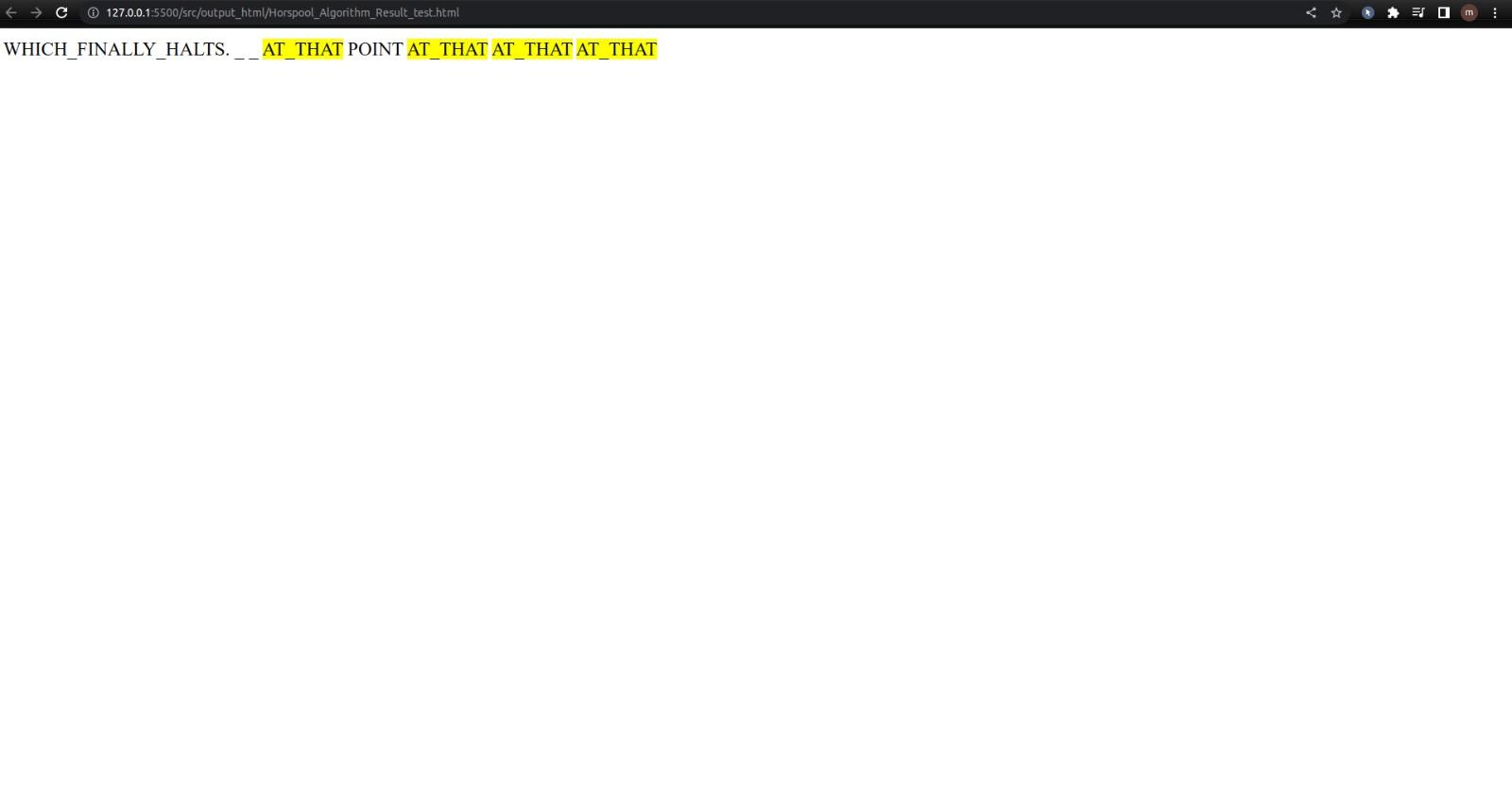
input\_html;



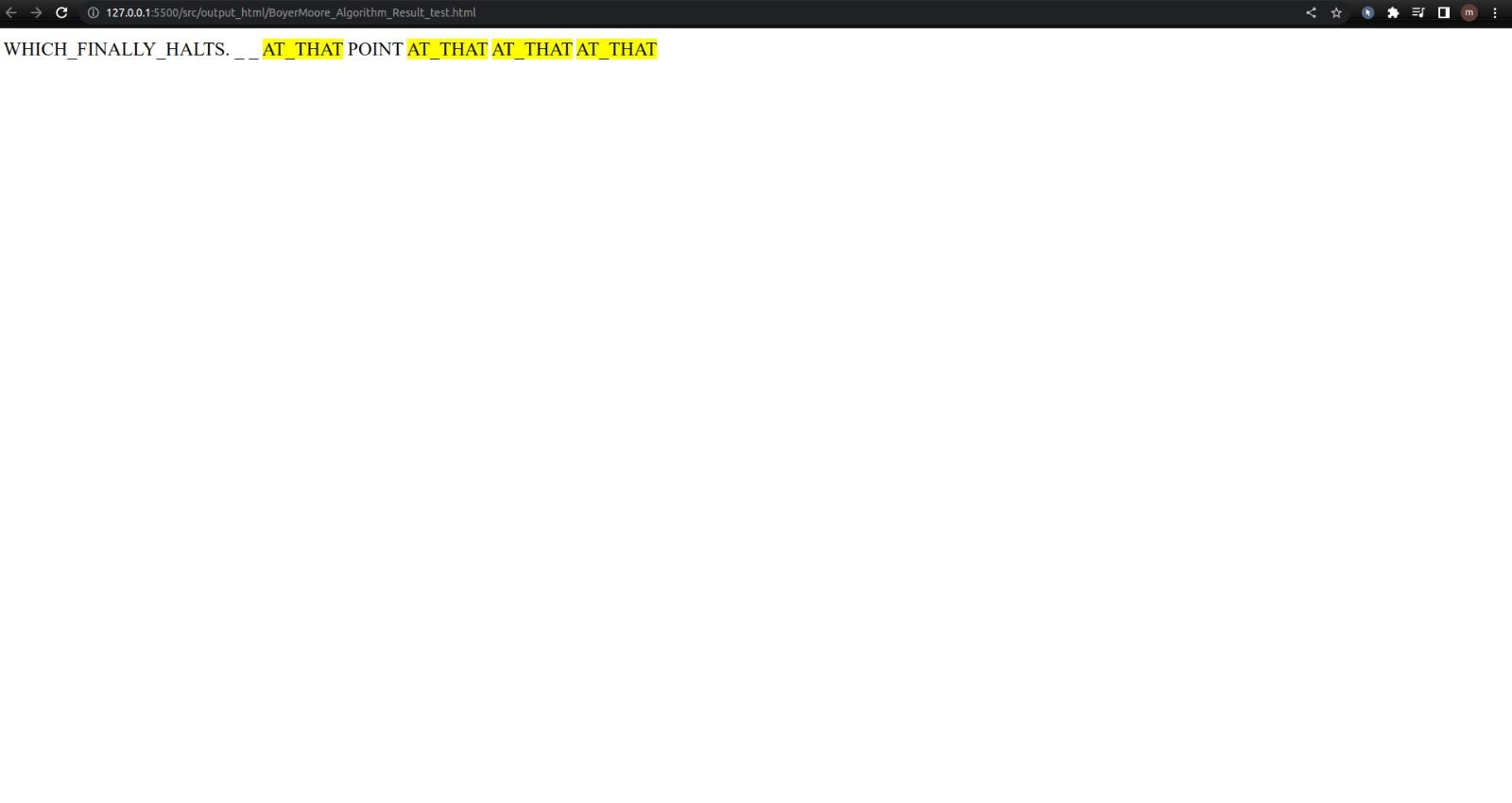
output\_html/BruteForce\_Algorithm;



output\_html/Horspool\_Algorithm



output\_html/BoyerMoore\_Algorithm

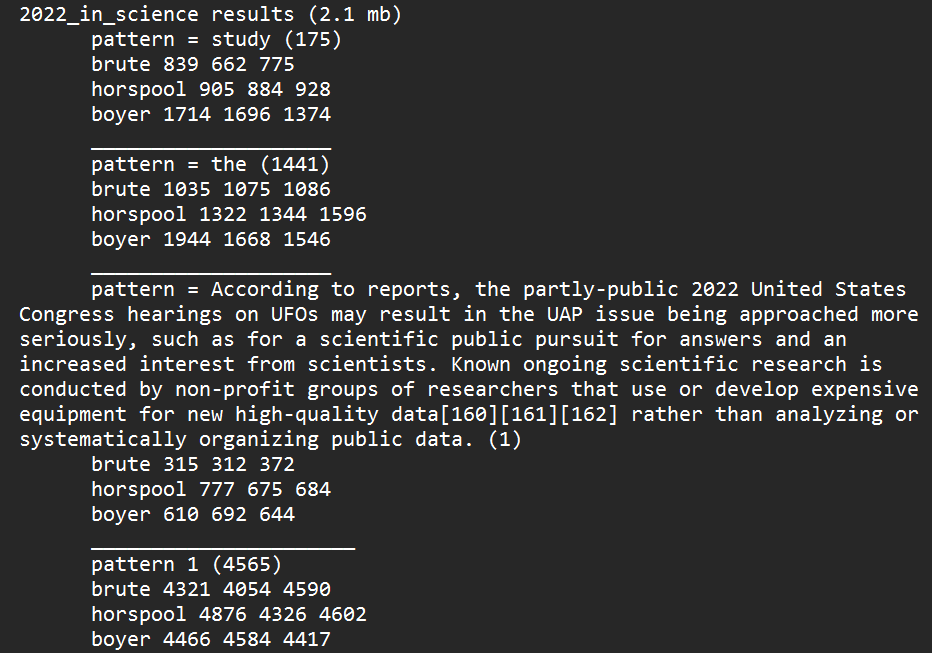


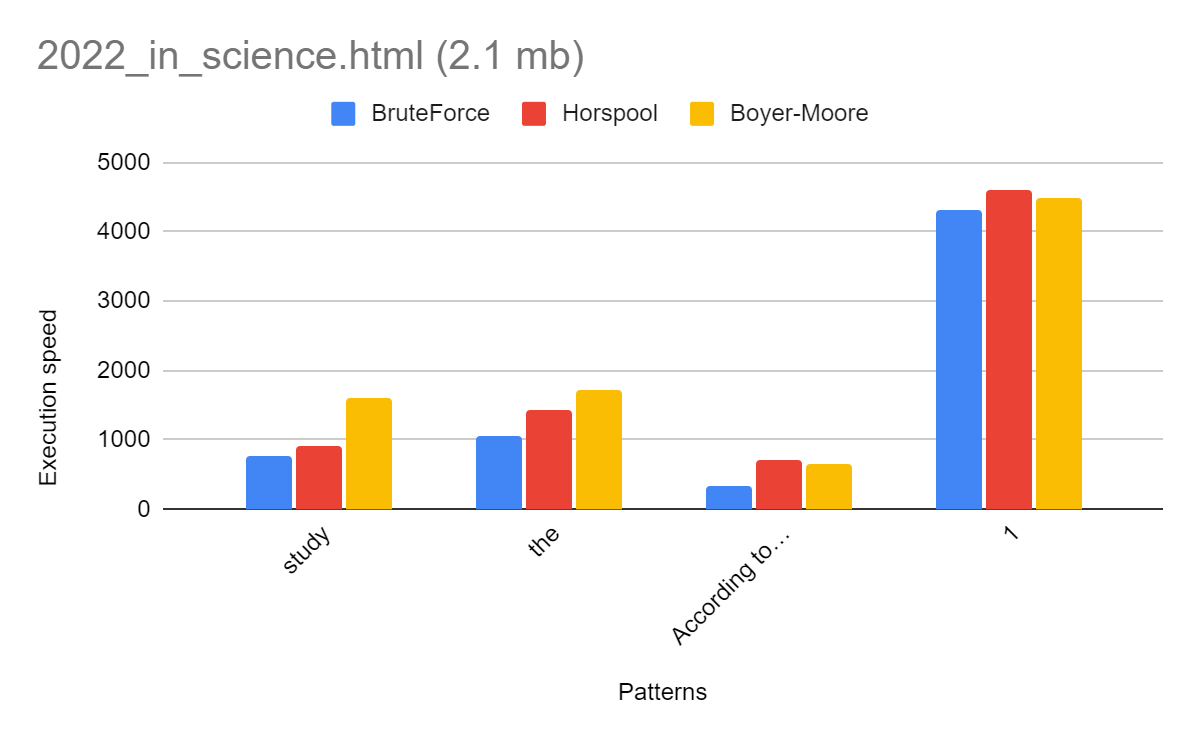
**3-)Illustrating and Analyzing Results:**

**(a) Illustrating the timing/complexity results in plots and/or tables and Comparing the performance of all the three algorithms for various HTML files and patterns.**

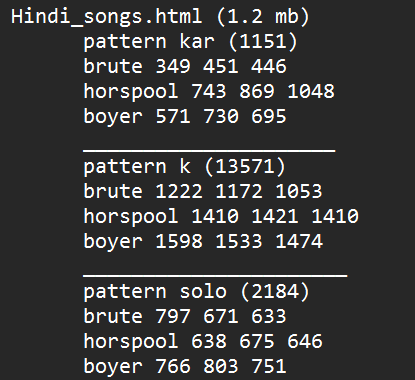
We executed 3 times for each algorithm and we took the average of them in order to get the most accurate result as much as possible.

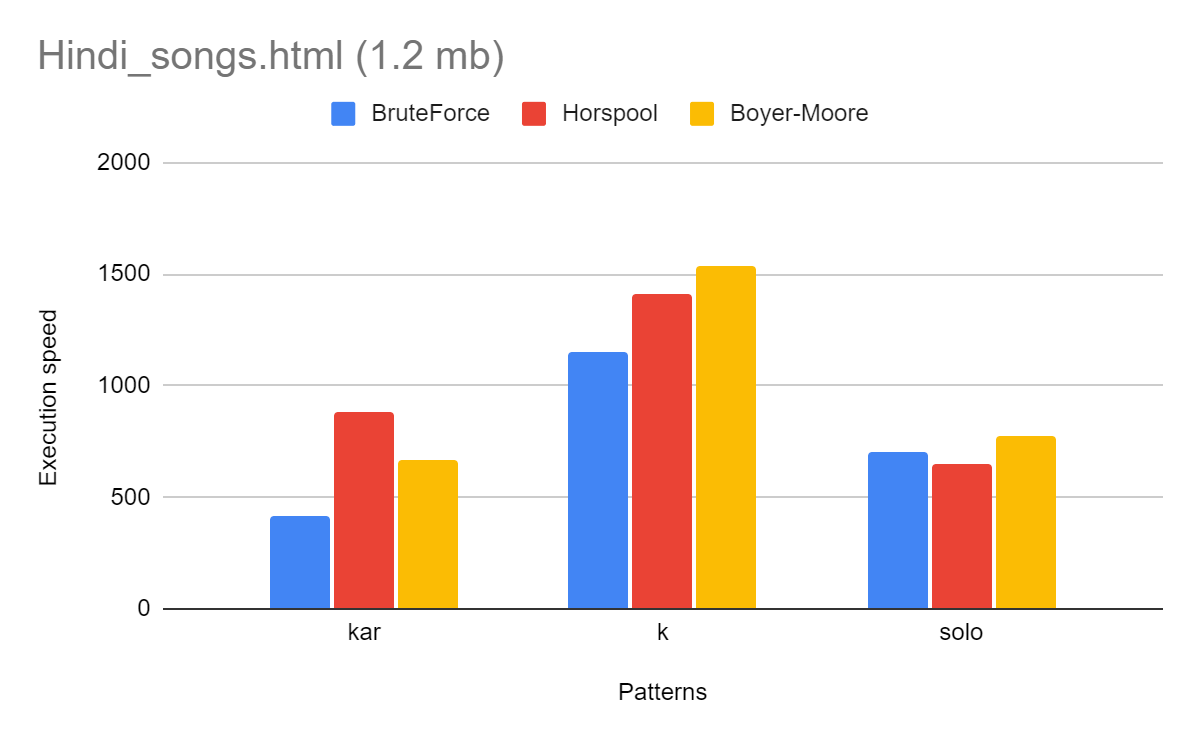
* **String based html files**

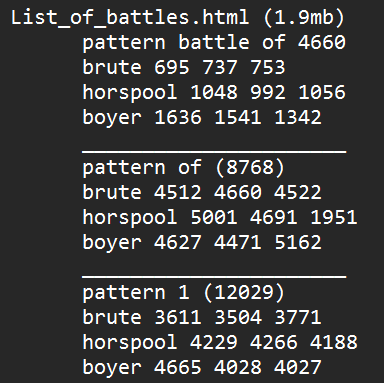


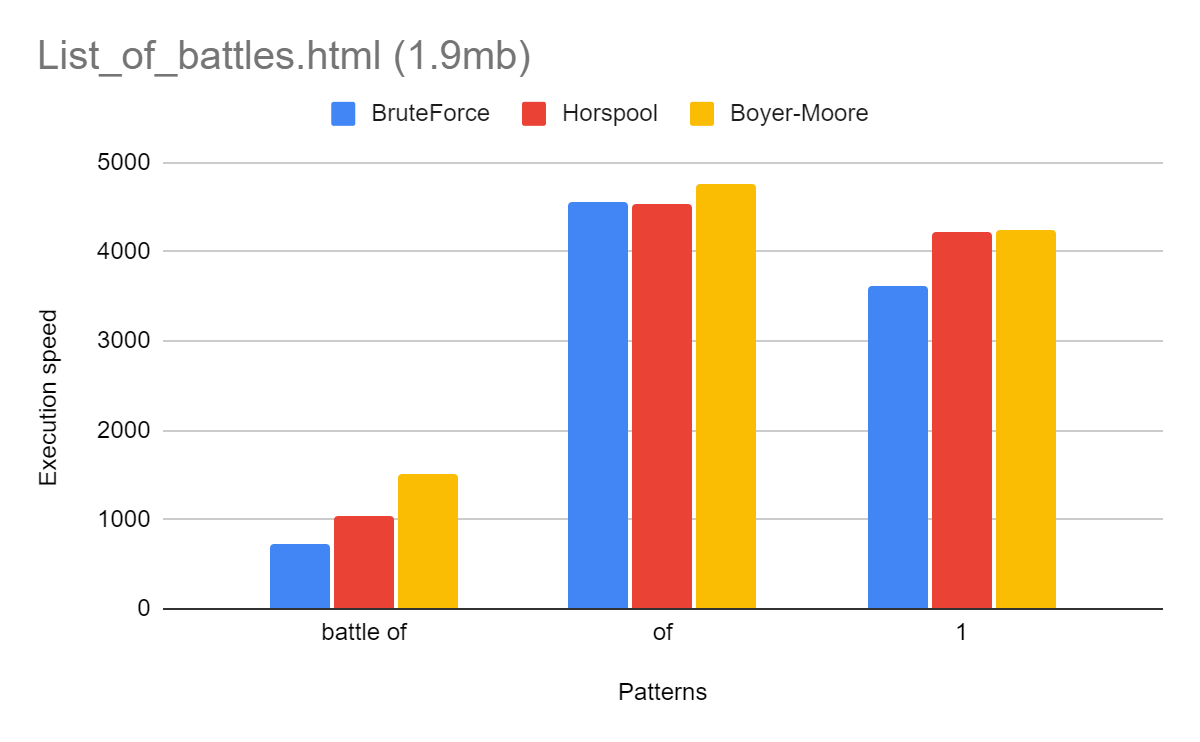


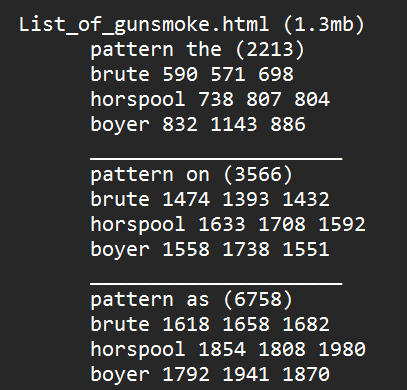
We executed 3 times for each algorithm and we took the average of them in order to get the most accurate result as much as possible.

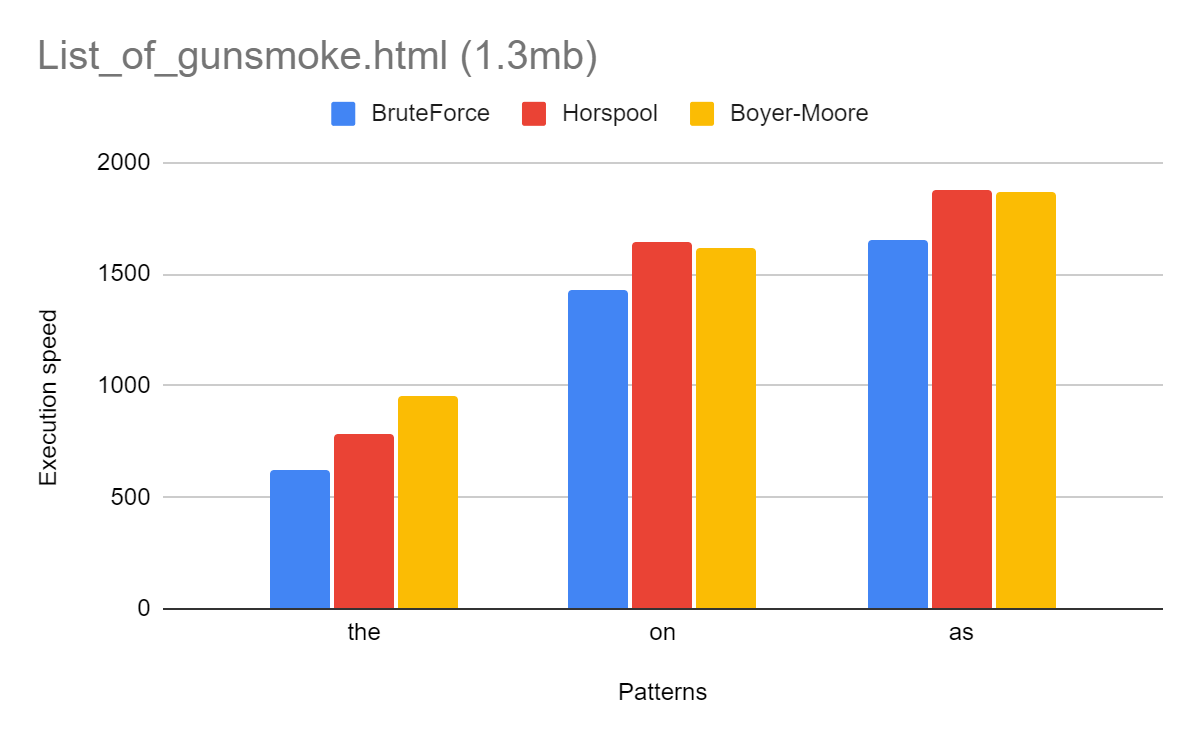


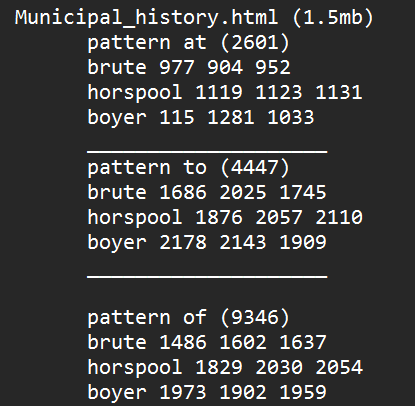


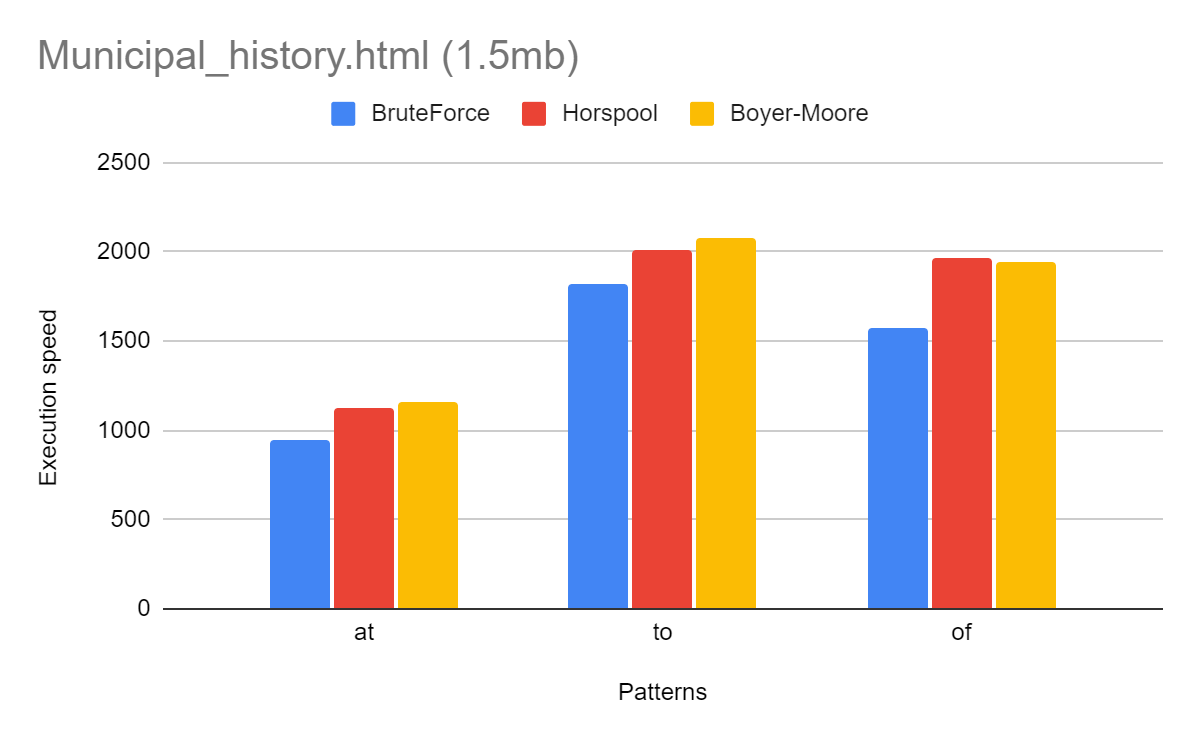


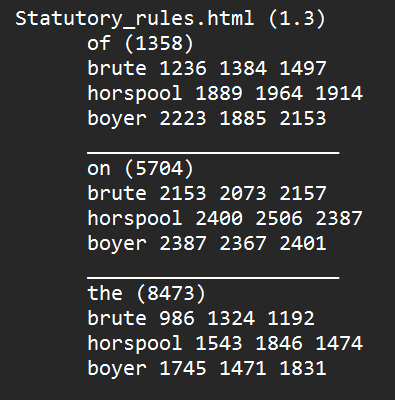


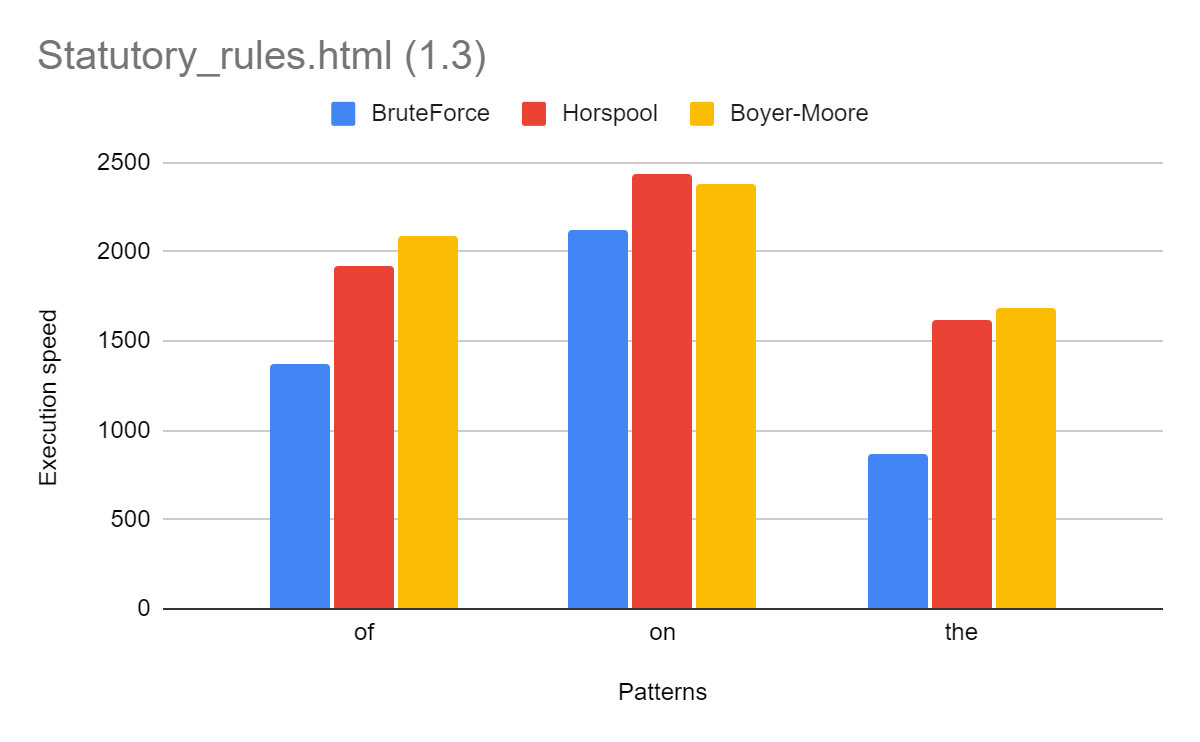




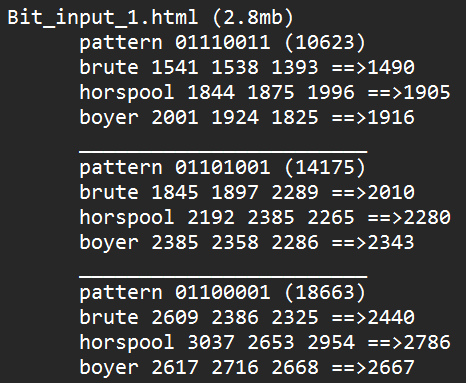


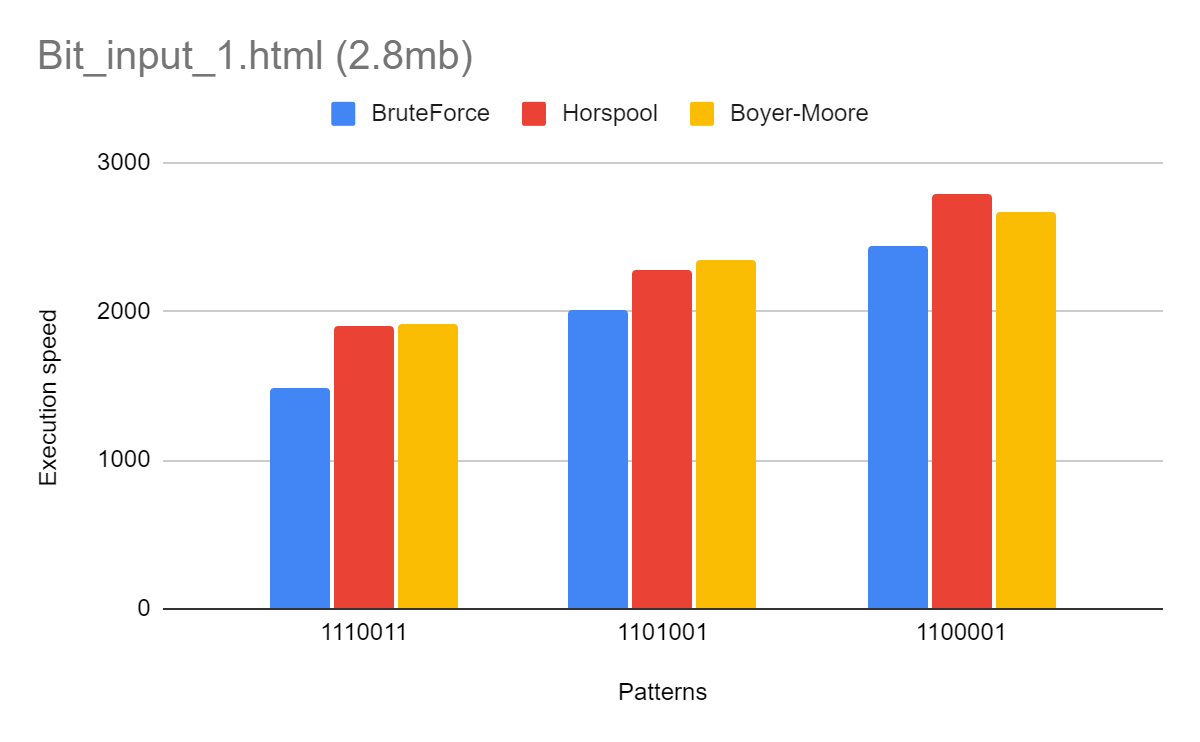


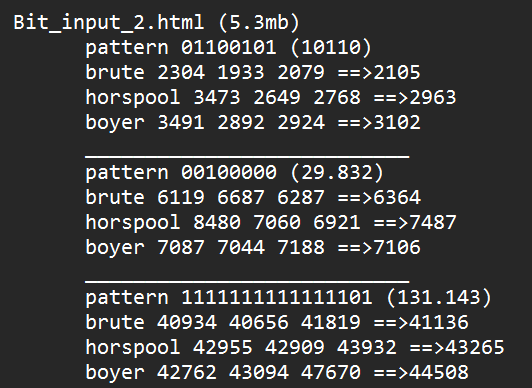


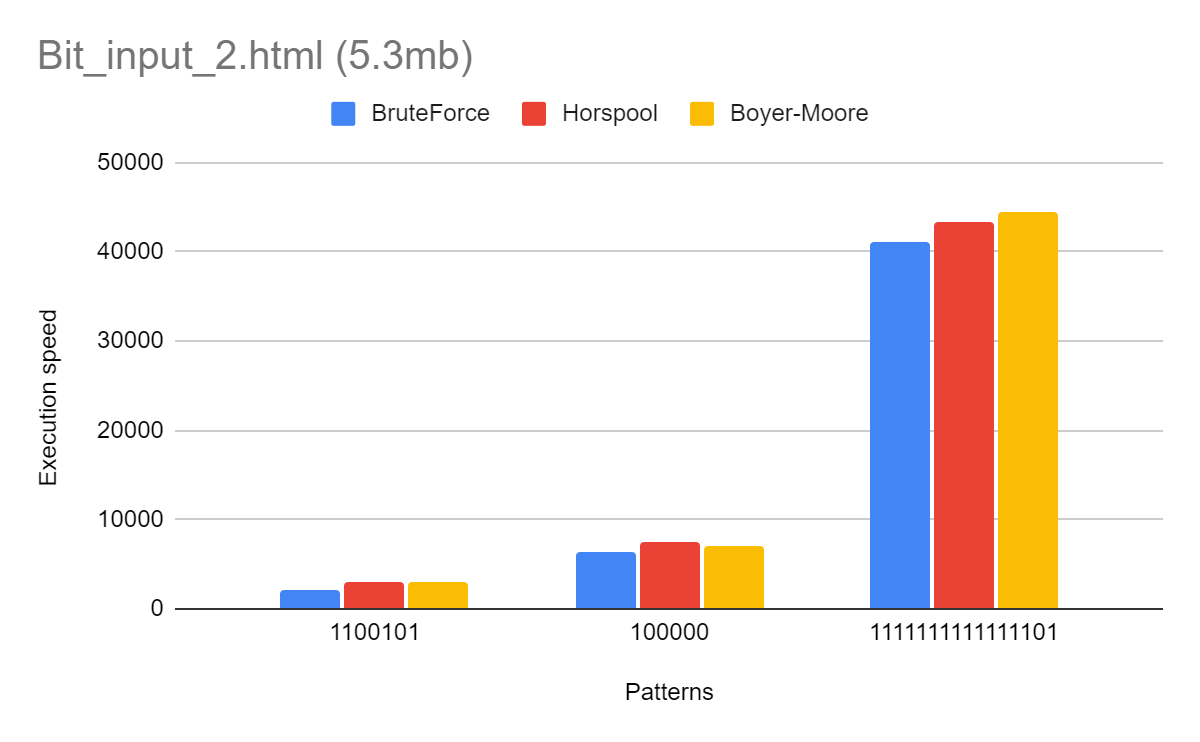


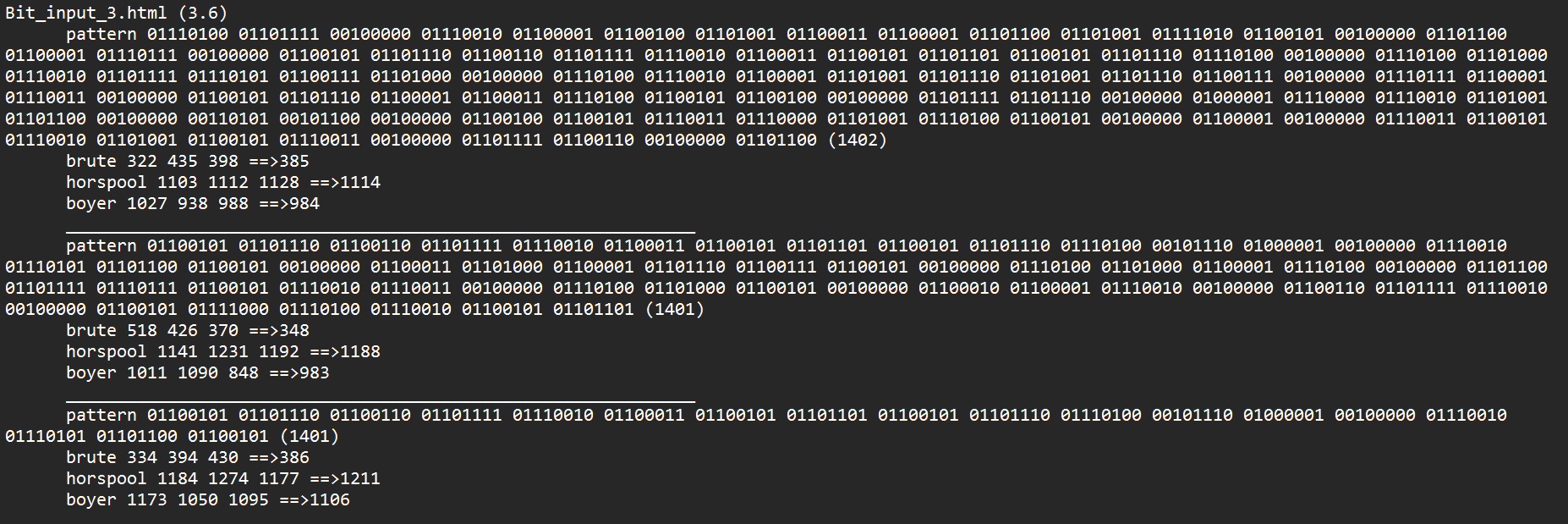
* **String bit html file result**

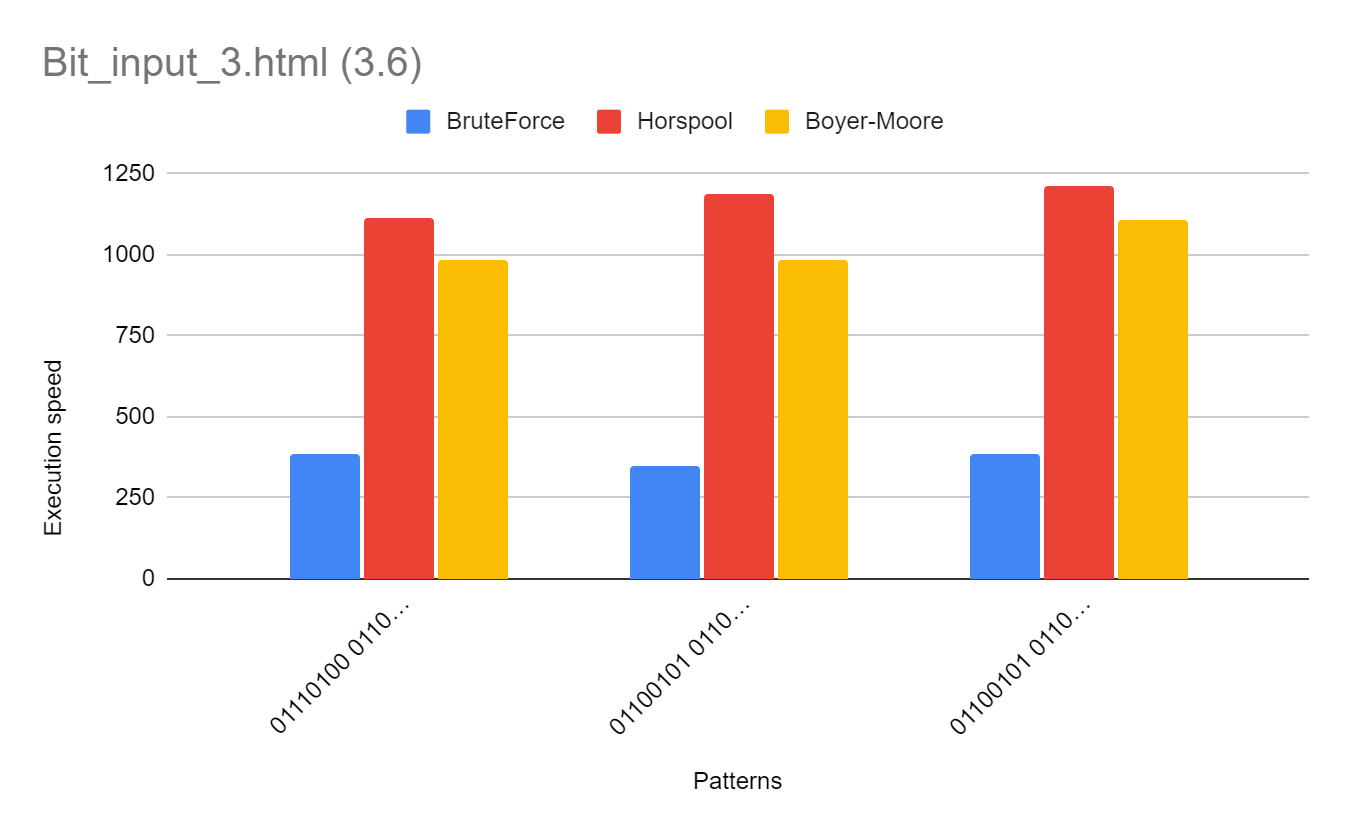












**(b)Analyzing result**

In this study, we compared the performance of three string matching algorithms: brute-moore, horspool, and boyer. We analyzed their execution times on multiple files with various patterns to draw meaningful conclusions.

Firstly, we found that the brute-moore algorithm consistently demonstrated competitive performance across different file sizes and patterns. It consistently provided efficient results and can be relied upon for most cases.

On the other hand, the horspool algorithm exhibited slightly higher execution times compared to brute-moore in most cases. Although it may not be the fastest algorithm, it could still be a suitable choice for smaller files or patterns with relatively higher frequencies.

The boyer algorithm showed varying performance throughout the experiments. In some cases, it outperformed the other algorithms, while in others, it lagged behind. Therefore, when considering the boyer algorithm, it is crucial to carefully assess the specific characteristics of the file and pattern being used.

Additionally, we observed that the number of occurrences of a pattern in a file varied significantly depending on the file and pattern chosen. Therefore, it is important to consider the frequency of the pattern relative to the file size when selecting an algorithm for string matching.

Moreover, the characteristics of the file itself played a vital role in determining the execution times of the algorithms. Larger files generally required more time to execute the algorithms, and the nature of the file content, such as HTML files versus bitstring files, also had some influence on algorithm performance.

To select the most appropriate algorithm for string matching, it is necessary to consider various factors, including file size, pattern frequency, and the desired execution time. The brute-moore algorithm emerged as a reliable choice, offering efficient performance across different file sizes and patterns. However, for specific cases involving smaller files or patterns with higher frequencies, the horspool algorithm could be considered. The boyer algorithm may yield competitive results in certain scenarios, but its suitability depends on the specific characteristics of the file and pattern.

In conclusion, when choosing a string matching algorithm, it is essential to have a comprehensive understanding of the file size, pattern frequency, and desired execution time. The findings of this study provide valuable insights into the performance of the brute-moore, horspool, and boyer algorithms, but further experimentation and analysis could deepen our understanding of their performance on different file types and patterns.

**4-) Implementation details for reading and parsing data**

We get the input from a file using BufferedReader and FileReader classes and we parse the input into a StringBuilder in order to get the appropriate result array or ArrayList would be hard for comparing strings.

When we get an index for occurrence we save all of them inside an Arraylist of Integers.

After getting all indexes for all occurrences we parsed <mark> at the beginning of index however there was a problem when parsing the <mark> tag inside StringBuilder instance when we add <mark> into it the size of StringBuilder changed. We added necessary implementation for that problem.

After adding mark tags is finished we parsed the StringBuilder into a file using BufferedWriter and FileWriter.