Word Search Application with Brute Force and KMP Algorithms

Group Members:

Aqsa Fayaz 22i-1865

Manal Amir 22i-1940

Muntaha Asim 22i-1868

Introduction:

In contemporary systems, searching for key terms in large volumes of textual content is vital. This word search application has been developed to enable effective keyword searching across several text files by implementing the Brute Force and Knuth-Morris-Pratt (KMP) string-searching algorithms. The interfaces of the application further allow users to enter search words and access matching options like whole word matching and case sensitivity to enhance search results. The project also includes monitoring of the algorithm efficiency in terms of searching speed which enhances the understanding of the Brute Force and KMP algorithms' effectiveness in different search conditions.

Choice of Language and Why?

Python was selected as the programming language for this project for several reasons:

1. **Readability and Simplicity**: Python's syntax is clean and highly readable, which is essential for implementing and maintaining complex algorithms like Brute Force and KMP without excessive complexity.
2. **Library Support**: Python has robust libraries like Tkinter for creating graphical user interfaces, time for performance measurement, and file-handling libraries that simplify text file management.
3. **Rapid Prototyping**: Python's dynamic nature allows for quick testing and debugging, which is beneficial when developing applications with real-time performance evaluation.
4. **Popularity in Data Processing**: Python's popularity in data analysis and text processing ensures strong community support, making it an ideal choice for this project.

Explanation of Algorithms:

1. **Brute Force Algorithm**:
    - **Concept**: The Brute Force algorithm searches for a pattern by examining each character in the text one-by-one until a match is found. This is done by aligning the start of the pattern with each character in the text and checking for a match character by character.
    - **Performance**: Although simple to implement, the Brute Force algorithm has a high time complexity, **O(m×n)**, where m is the length of the text and n is the length of the search term. This makes it less efficient, especially in large text files or when the search term appears frequently.
    - **Use Case in Application**: This algorithm is particularly useful in scenarios where simplicity and ease of implementation are prioritized over performance. It provides a baseline for comparing more advanced algorithms.

2. **Knuth-Morris-Pratt (KMP) Algorithm**:
    - **Concept**: The KMP algorithm optimizes the search process by preprocessing the search term to create a partial match table (or failure function). This table helps the algorithm skip certain parts of the text based on previously matched characters, reducing the number of comparisons needed.
    - **Performance**: The KMP algorithm has a time complexity of **O(m+n)**, making it more efficient than Brute Force, particularly for large text sizes. Its efficiency comes from its ability to avoid redundant comparisons, making it significantly faster in repeated search operations.
    - **Use Case in Application**: KMP is ideal for scenarios that require faster search speeds, especially when searching through large text files. It's implemented here to provide an optimized alternative to Brute Force, allowing the user to choose based on performance needs.

Functional Requirements:

1. **Search Functionality**:
    - **User Input**: The application provides a text field where users can enter any word or phrase they wish to search within a collection of files. Once the user inputs their search term and initiates the search, the application scans all specified files, streamlining the process of locating keywords across multiple documents.
    - **Multi-File Search**: This functionality allows the application to search across multiple files simultaneously. It simplifies the task for users who need to sift through extensive text or document sets, making it ideal for large datasets or directories with many files. Users no longer need to open and manually search each file; the application handles this automatically.
    - **Detailed Search Results**: For every occurrence of the search term found within a file, the application provides File Name,Row Number and Column Number.

2. **Advanced Matching Options**:
    - **Unchecked (Substring Match)**: When the whole word match option is disabled, the application will find the search term wherever it appears, even within larger words. For instance, if the user searches for "pack," the results may include words like "package," "backpack," and "packed." This option broadens the search scope, making it ideal for users who want to capture any instance related to the term, even if it is part of another word.

    - **Checked (Exact Match)**: When the whole word option is enabled, the application restricts results to exact matches. For example, a search for "pack" will only return instances where "pack" appears as a standalone word, excluding variations like "package" or "packed." This option is useful when users need precise matches, avoiding irrelevant results.

- **Unchecked (Case-Insensitive)**: With case sensitivity disabled, the application treats the search term as case-insensitive, meaning that "Pack," "pack," and "PACK" are all considered matches for a search term "pack." This is ideal for general searches where the user is not concerned with capitalization.
- **Checked (Case-Sensitive)**: When case sensitivity is enabled, the application only returns matches that exactly match the case of the search term. For instance, if the search term is "Pack," results will not include "pack" or "PACK." This feature is beneficial for cases where capitalization impacts meaning or context.

3. **Performance Evaluation**:
    - The application supports both **Brute Force** and **Knuth-Morris-Pratt (KMP)** algorithms to perform the search, each offering different performance benefits.
    - **Execution Time Display**: After each search, the application records and displays the time taken by both algorithms, allowing users to compare their performance. This feature provides valuable insights into how each algorithm handles the search task, with KMP typically being faster due to its optimized structure, especially in larger files or high-frequency searches.
    - **Comparative Insights**: By presenting execution times for both algorithms, the application allows users to make informed decisions on which algorithm is more efficient for their specific dataset or search requirements. For instance, while Brute Force might suffice for small files, KMP is usually more efficient for larger text searches.

Sample Images:

Below are sample images that demonstrate the application's interface, input/output functionality, and performance evaluation features:

1. **User Interface - Main Screen**
    - *Description*: This screenshot shows the application's main interface, including the search term input field, file selection area, and options for enabling whole word match and case sensitivity.

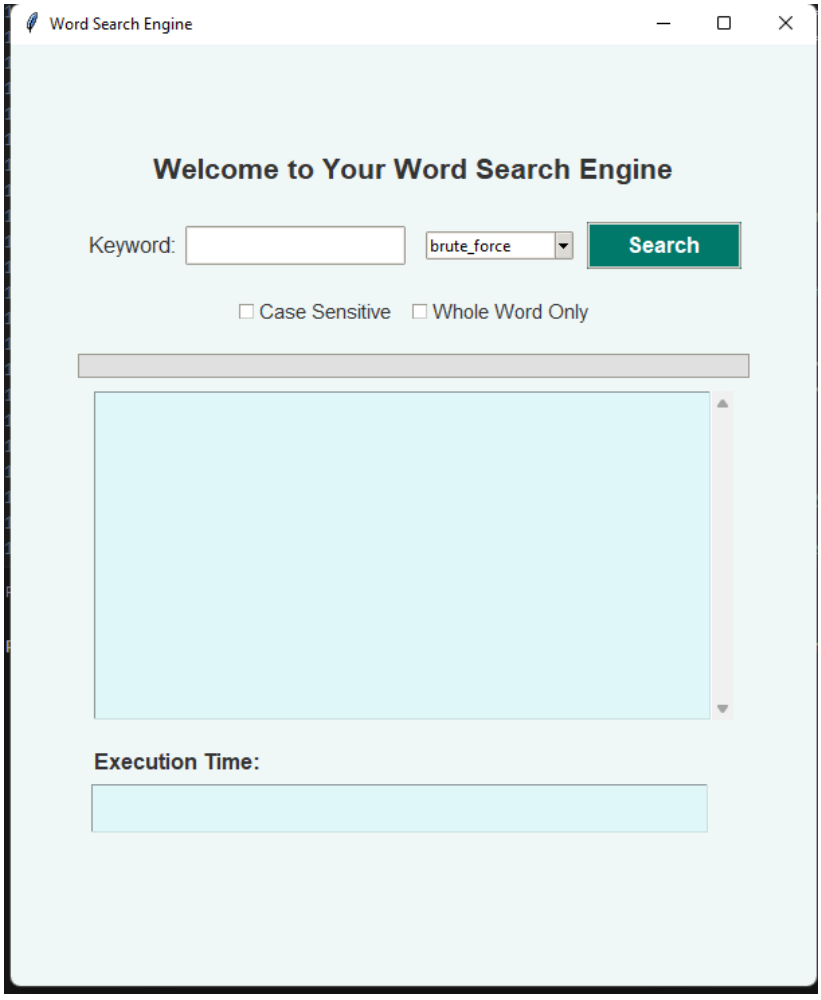

Figure 1 Main Screen

2. **Search Results Display**
    - *Description*: This image captures the application's display of search results, showing file names, row numbers, and column positions for each occurrence of the search term.

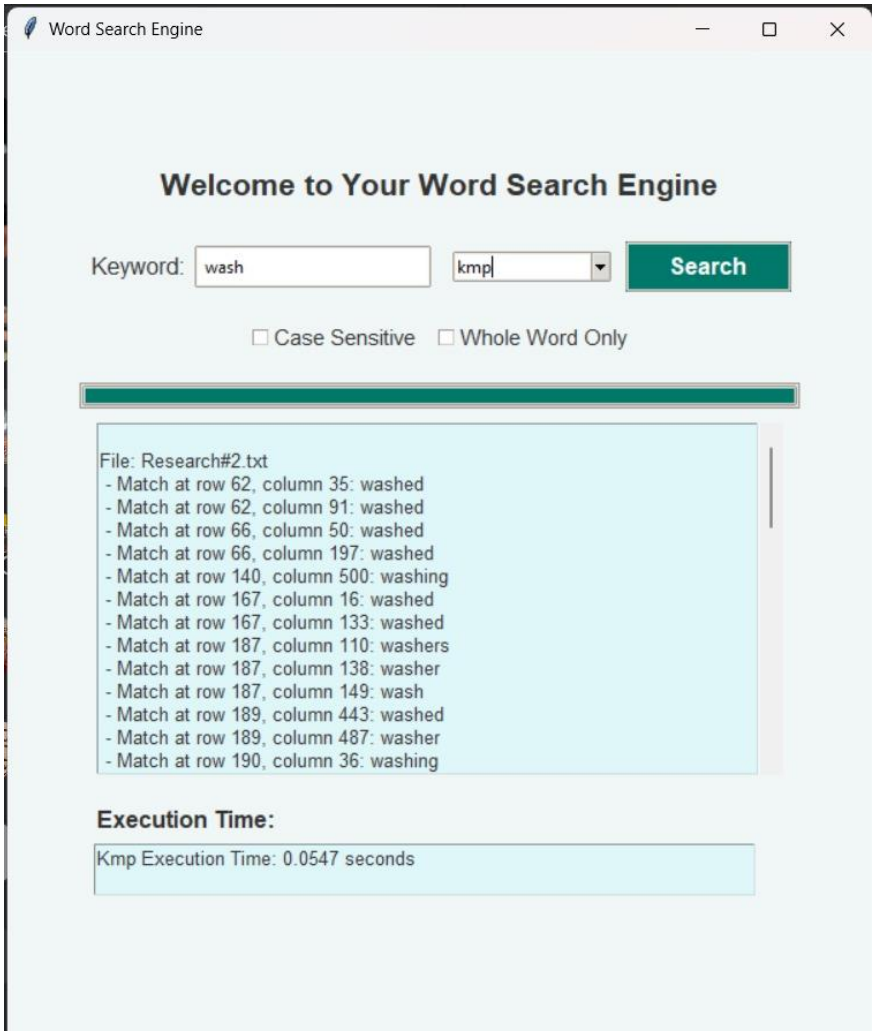

Figure 2 Result display

3. **Performance Evaluation - Brute Force Algorithm Only**
    - *Description*: This screenshot showcases the performance evaluation for the Brute Force algorithm, displaying the execution time taken to complete the search. It provides insights into the efficiency of Brute Force when used without comparison to other algorithms.

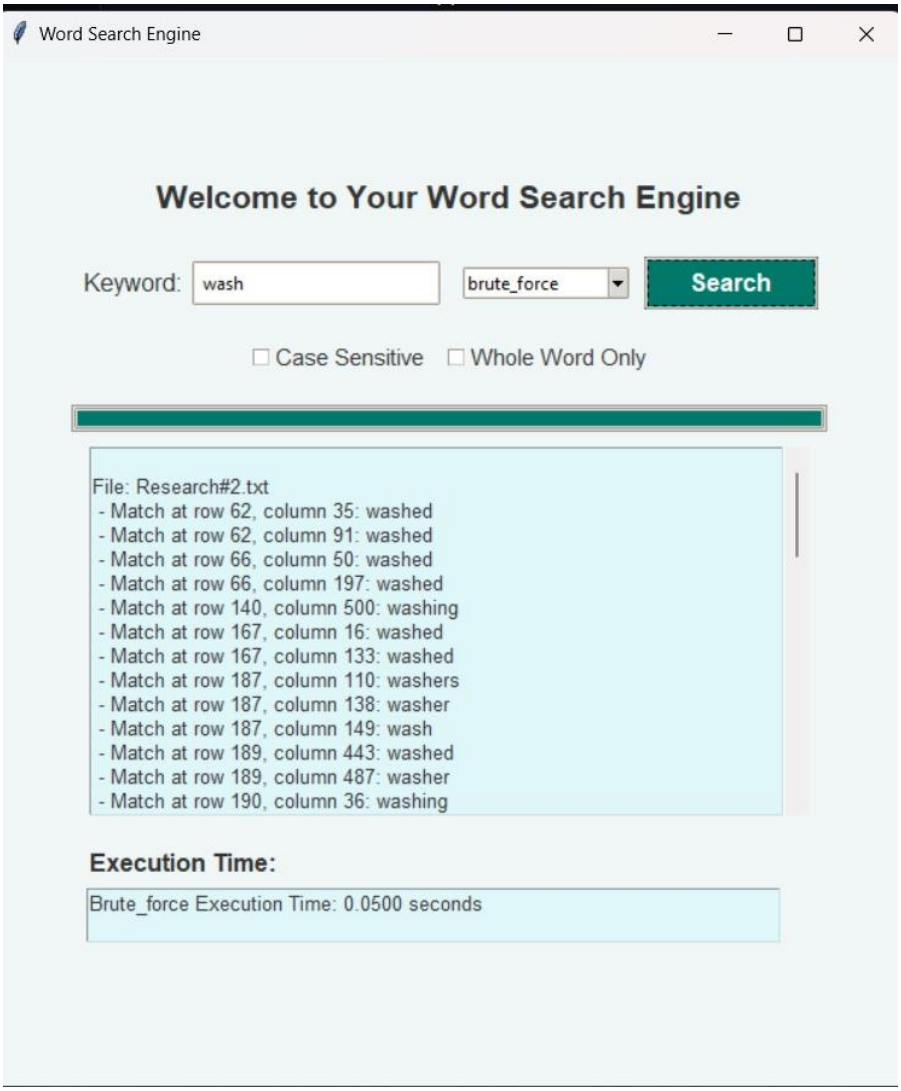

Figure 3 Brute Force Algorithm

4. **Performance Evaluation - KMP Algorithm Only**
    - *Description*: This screenshot highlights the performance of the KMP algorithm exclusively, displaying the time taken to execute the search. This allows users to see the efficiency of KMP on its own, particularly in scenarios where optimized search speed is essential.

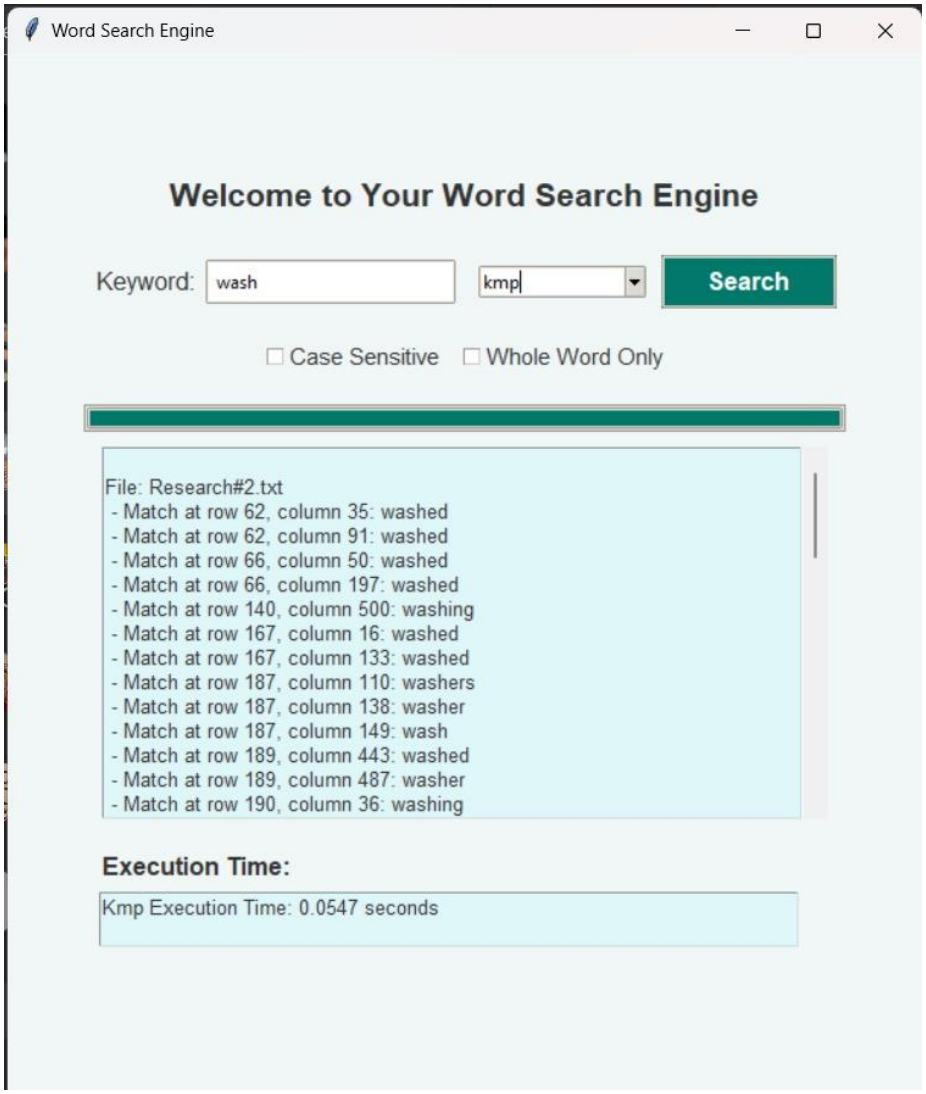

Figure 4KMP Algorithm

5. **Performance Evaluation Comparison - Brute Force vs KMP**
    - *Description*: This screenshot shows the performance comparison for Brute Force and KMP algorithms, including the time taken by each algorithm to complete the search. It allows users to see the efficiency of each method.

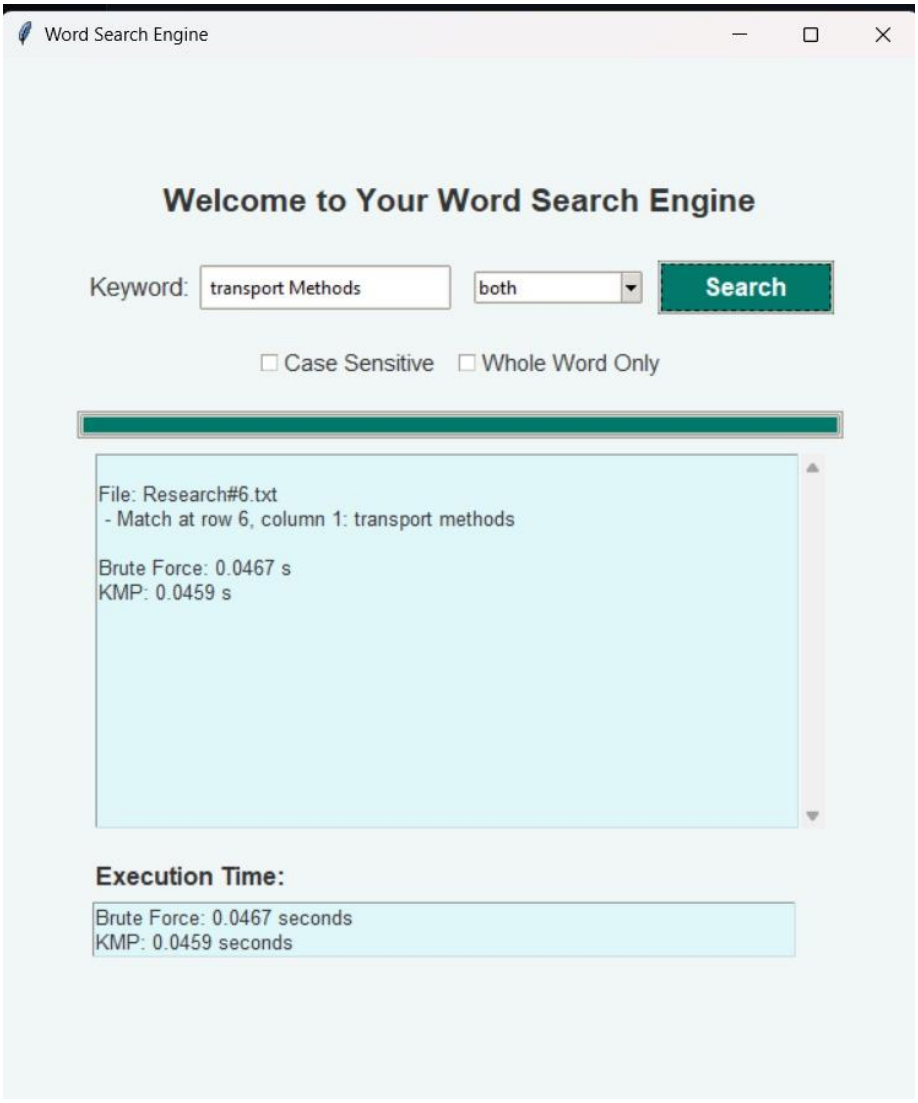

Figure 5 Comparing Brute force with KMP

6. **Whole Word Match - Exact Matches Only**
    - *Description*: This image showcases the search results when the "Whole Word Only" option is enabled, displaying only exact matches of the search term within files.
    -

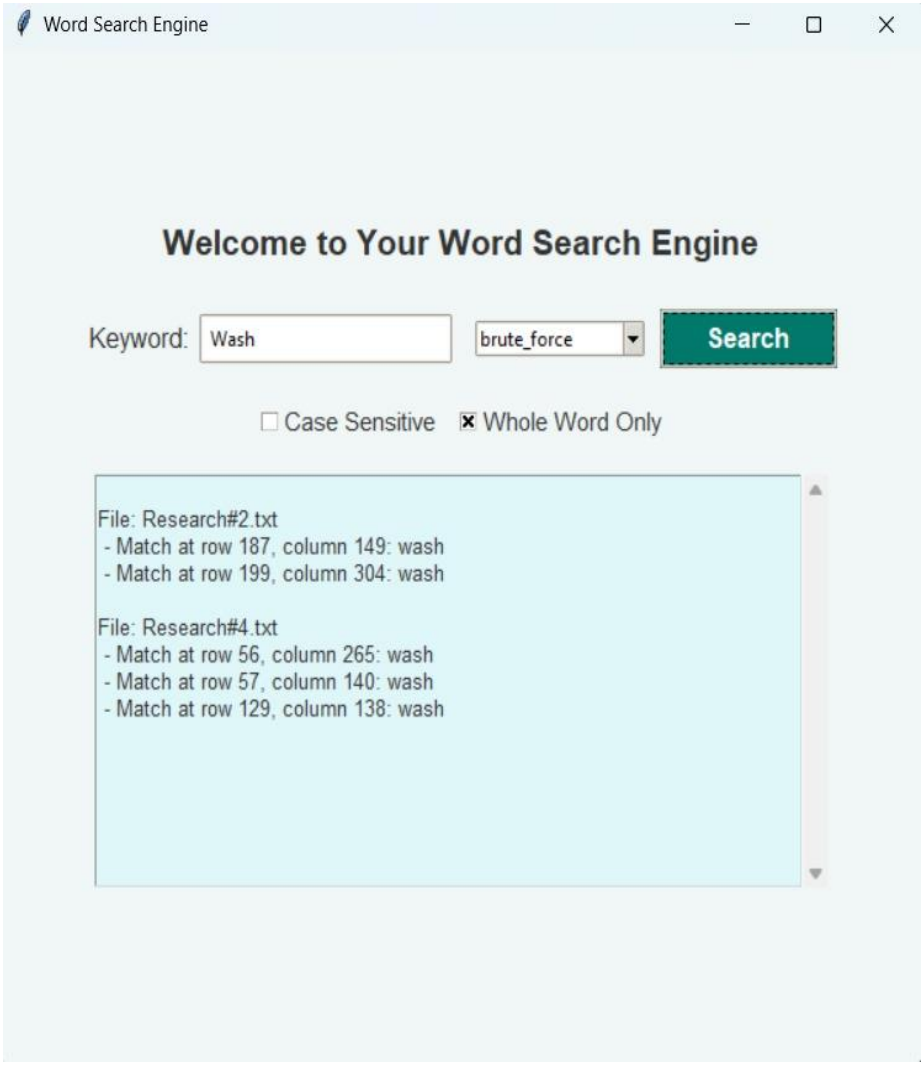

Figure 6 Whole word match

7. **Case Sensitive Search - Precise Results**
    - *Description*: This screenshot displays the results of a case-sensitive search, where only matches with the exact casing of the search term are shown, demonstrating the application's filtering capability.

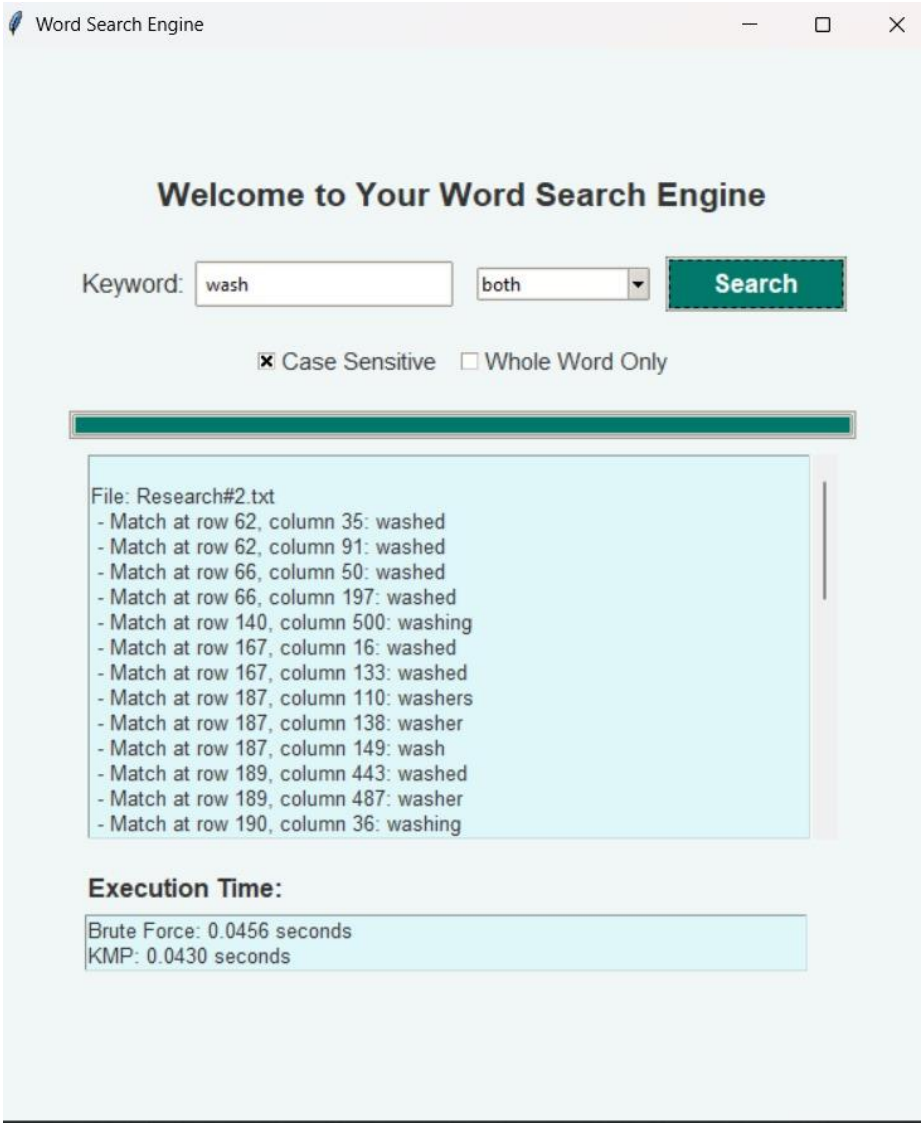

Figure 7 Case Sensitive Search

8. **Both Case Sensitive and Whole Word Only Options Enabled - Results Display**
    - *Description*: This screenshot demonstrates the search results when both "Case Sensitive" and "Whole Word Only" options are enabled. It ensures that only exact matches with correct casing and whole word boundaries are shown, providing highly filtered search results.

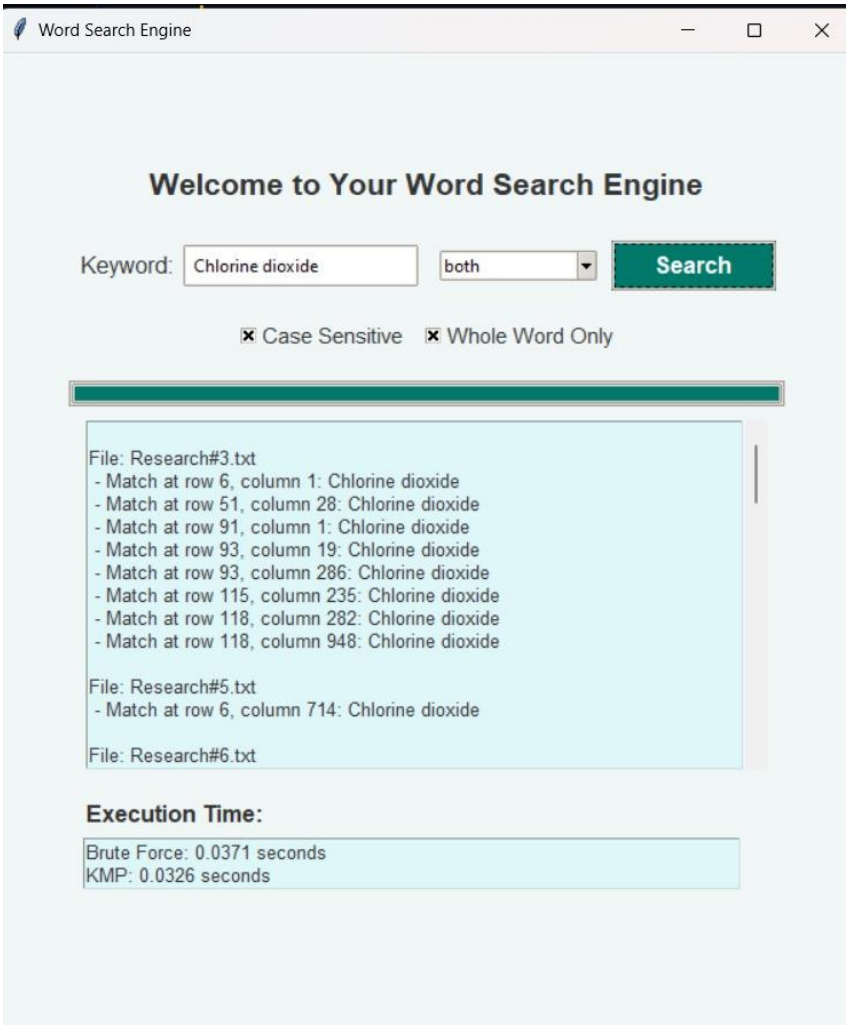

Figure 8 Both constraints

Conclusion:

The word search application fulfills its objective by providing a flexible and efficient text-search tool with a user-friendly GUI. The inclusion of both Brute Force and KMP algorithms allows users to choose based on performance needs, demonstrating the strengths and limitations of each approach. This project highlights the importance of algorithm selection in text-processing applications, providing a practical comparison of their efficiencies.