**MINI PROJECT ON**

IMPLEMENTATION OF LINEAR SEARCH ALGORITHM.

# 

**NAME : KUNAL DEEPAK BHOIR.**

**CLASS : SYBSC.CS**

**ROLL NO : 04**

**DIV: A**

**TOPIC : IMPLEMENTATION OF LINEAR SEARCH GUI BASED.**

**UNDER GUIDANCE : MRS.RASHAMI PRABHA MA’AM**

＊INDEX＊

|  |  |  |
| --- | --- | --- |
| **Sr.**  **No .** | **TOPIC** | **Page**  **No .** |
| **1** | **Introduction to LINEAR Search .** | **2** |
| **2** | **OBJECTIVE** | **6** |
| **3** | **IMPLEMENTATION DETAILS** | **7** |
| **4** | **Analysis of Results** | **11** |
| **5** | **CONCLUSION** | **12** |
| **6** | **FUTURE ENHANCEMENTS** | **18** |
| **7** | **OUTPUT** |  |

# **＊INTRODUCTION＊**

* **WHAT IS LINEAR SEARCH ?**

A linear search, also known as a sequential search, is a basic search algorithm used to find a specific item in a list or array. It works by checking each element in the list one by one until the desired item is found or all elements have been examined. It's like looking for a particular book in a library by going through the shelves one by one until you find it. Linear search is simple to understand and implement, but it can be slow for large lists since it checks each element in sequence, making it have a time complexity of O(n), where n is the number of elements in the list.

**The Linear Search Algorithm is a fundamental and intuitive searching technique used to find a specific element within a list or array. In this project, we aim to develop a Python application that not only implements the linear search algorithm but also provides users with a visual representation of how the algorithm operates. By doing so, we aim to create an educational tool that helps users understand the inner workings of this algorithm in a more intuitive and interactive way.**

* **OBJECTIVE:-**

**The objective of this program is to create a graphical user interface (GUI) application that enables users to perform a linear search on a list of elements. This program allows users to input a list of elements and a target element they want to find within that list. It then visually demonstrates the linear search process on a canvas, highlighting each element as it is checked for a match. The program provides immediate feedback on whether the target element is found and at which index if it exists in the list. Users can also switch between light and dark themes for the GUI and access a history of previously entered lists for reference. This program serves as an educational tool for understanding the linear search algorithm and offers a practical example of GUI development in Python.**

* **IMPLEMENTATION DETAILS:-**
* **Import Statements**

**Purpose:**

**The import statements at the beginning of the code serve the purpose of importing necessary libraries and modules to make the program functional and visually appealing.**

**Imports:**

* **customtkinter as ctk: This custom module provides enhanced GUI elements for the application.**
* **tkinter as tk: The standard tkinter library is used for basic GUI components.**

**time: The time module is used for adding delays to visualize the search process.**

* **CTkMessagebox: This custom module provides custom message boxes for displaying messages to the user.**
* **Global Variables**

**Purpose:**

**Global variables are used to store data that needs to be accessed and modified throughout the program's execution.**

* **Variables:**
* **stored\_user\_lists: This is a list that stores the lists of elements entered by the user. It allows us to keep track of previously entered lists.**
* **dark\_mode: A Boolean variable that keeps track of the current theme mode (light or dark).**
* **Linear Search Algorithm**

**Purpose:**

**The core of this program is the implementation of the linear search algorithm. It is responsible for searching for the target element within the user-entered list and visualizing the search process.**

**Function: linear\_search(arr, target, canvas, index=0)**

**This is a recursive function that performs the linear search.**

* **arr: The list in which the search is performed.**
* **target: The element being searched for.**
* **canvas: The tkinter canvas used for visualization.**
* **index: An optional parameter that keeps track of the current position in the list.**
* **Algorithm:**

**The function checks if the current index is equal to the length of the list. If so, it returns -1, indicating that the target element was not found.**

* **Visualization: The function updates the canvas to visualize the search process. It creates arrows, highlights elements, and waits for a short period to show each step.**
* **Comparison: The function compares the current element with the target.**

**If they match, it returns the index where the target was found.**

**If not, it continues the search for the next element recursively.**

**If the target is not found in the entire list, it returns -1.**

* **Draw Elements on Canvas**

**Purpose:**

**This function is responsible for drawing elements on the canvas to provide a visual representation of the search process.**

**Function: draw\_elements(canvas, elements, target)**

**canvas: The tkinter canvas where elements are drawn.**

**elements: The list of elements being searched.**

**target: The target element being searched for.**

* **Algorithm:**

**The function calculates the size and position of each element on the canvas to ensure they fit correctly.**

**It draws rectangles and labels for each element.**

**It also displays the index label at the bottom of each box.**

**The visualization is designed to provide a clear representation of the search process.**

* **Themes**

**Purpose:**

**The theme-related functions allow users to switch between light and dark themes, customizing the application's appearance.**

**Functions:**

**set\_light\_theme(): Sets the theme to light mode.**

**set\_dark\_theme(): Sets the theme to dark mode.**

**toggle\_theme(): Toggles between light and dark themes.**

**update\_ui(): Updates UI elements when the theme changes.**

* **Update Canvas Size**

**Purpose:**

**This function ensures that the canvas adapts to window resizing, providing a responsive user interface.**

**Function: update\_canvas\_size(event)**

**event: The event generated when the window is resized.**

* **Algorithm:**

**The function calculates the canvas width and height based on the window's size.**

**It updates the canvas dimensions to match the calculated size, ensuring elements fit within the canvas.**

* **Show Stored List**

**Purpose:**

**This function displays a new window showing the history of entered lists, allowing users to see previously entered data.**

**Function: show\_stored\_list()**

**This function opens a new window to display the history of user-entered lists.**

**It lists all the previously entered lists.**

* **Main Application Window**

**Purpose:**

**This section creates the main application window, including GUI elements such as entry fields, buttons, labels, and the canvas for visualization.**

**Components:**

* **Entry fields for entering elements and the target.**
* **Buttons for performing actions (search, clear, show list history, theme switch).**
* **Labels for displaying instructions and search results.**
* **The canvas for visualizing the search process.Event bindings to handle window resizing.The elements\_rectangles list to keep track of drawn rectangles on the canvas.**
* **Analysis of Results:-**

**Functionality and Search Algorithm:**

* **The program implements a linear search algorithm to find a target element in a user-provided list.**
* **It successfully performs the linear search, visualizing each step of the search process on the canvas.**
* **Visualization elements include arrows, color changes (yellow for comparison, green for found, and red for not found), and index labels.**
* **The recursive nature of the search algorithm is evident, as the program traverses the list step by step until the target is found or the end of the list is reached.**
* **User Interface:**

**The program offers a user-friendly graphical user interface (GUI) with clear entry fields and buttons.**

* **Entry fields allow users to input a list of elements (one per line) and specify the target element to search for.**
* **Buttons include "SEARCH" for initiating the search, "CLEAR" to reset the input fields, and "ENTERED LISTS HISTORY" to view previously entered lists.**
* **The canvas provides a visually appealing representation of the elements and the search process.**
* **Labels display the entered list, search results, and step-by-step search progress.**

**Program Execution:**

* **The program handles user input effectively, providing validation for input data.**

**During execution, it effectively visualizes the search process, making it easy for users to understand how the algorithm works.**

**The use of arrows and color changes helps users track the progress of the search.**

**Upon completing the search, the program displays a clear result message, indicating whether the target element was found and at which index.**

* **Buttons are appropriately enabled or disabled during execution to prevent multiple searches or data entry errors.**
* **Stored Lists History:**

**The program maintains a history of previously entered lists, allowing users to review them.**

* **The "ENTERED LISTS HISTORY" button provides easy access to this feature.**

**This functionality enhances user convenience and helps with record-keeping.**

* **Themes:**

**The program offers a theme-switching feature, allowing users to choose between light and dark themes.**

**Theme changes are reflected in the GUI elements and canvas appearance.**

**This feature enhances the program's user experience by catering to different preferences.**

* **Results and Observations:**

**The program effectively performs linear searches on user-provided lists.**

**Successful searches result in clear visual indications, with the target element highlighted in green, and the program displays the index at which the target was found.**

**Unsuccessful searches are similarly well-handled, with the target element highlighted in red to indicate that it was not found.**

**The step-by-step visualization of the search process aids in understanding the algorithm's operation.**

* **CONCLUSION:-**

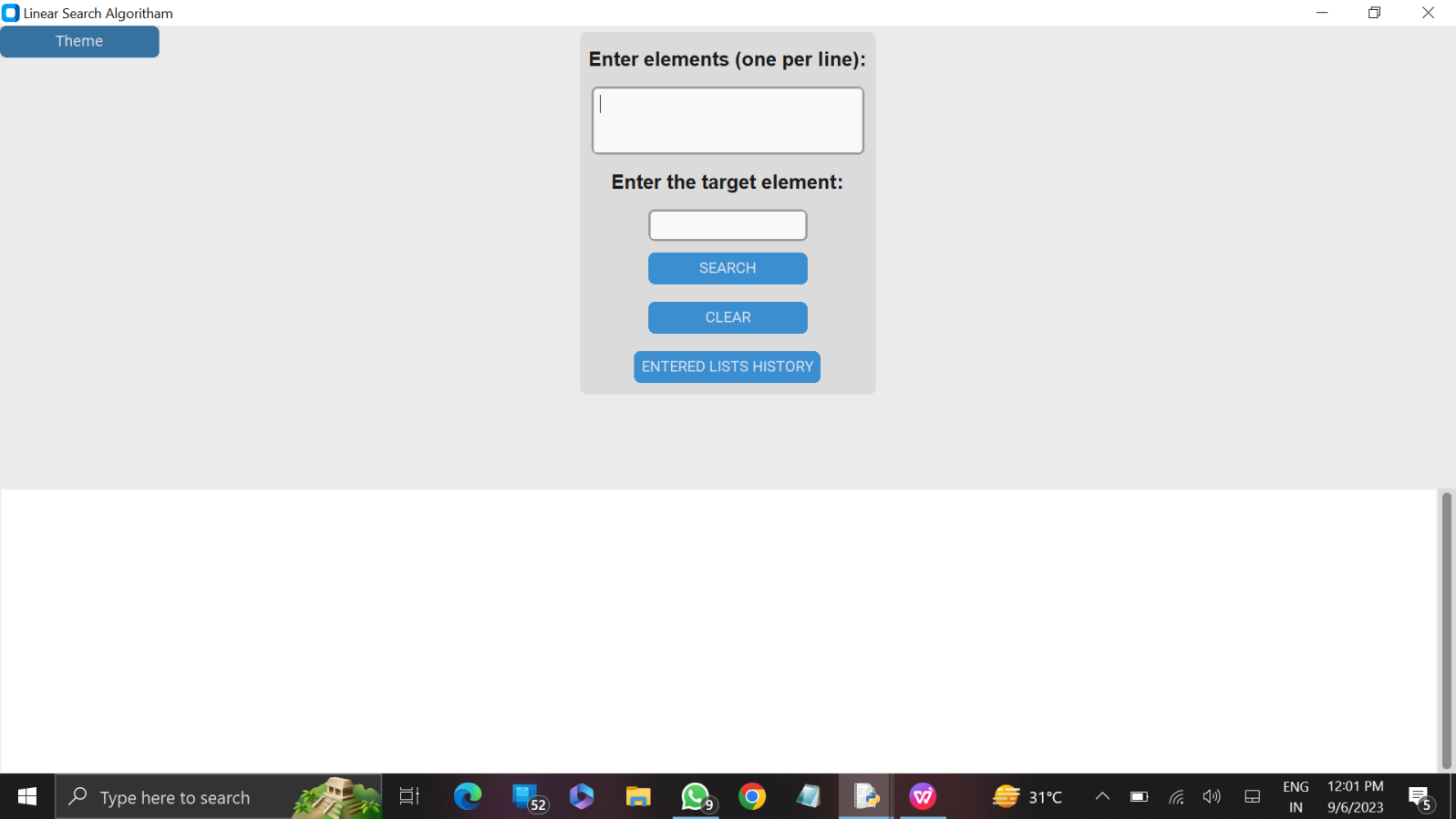
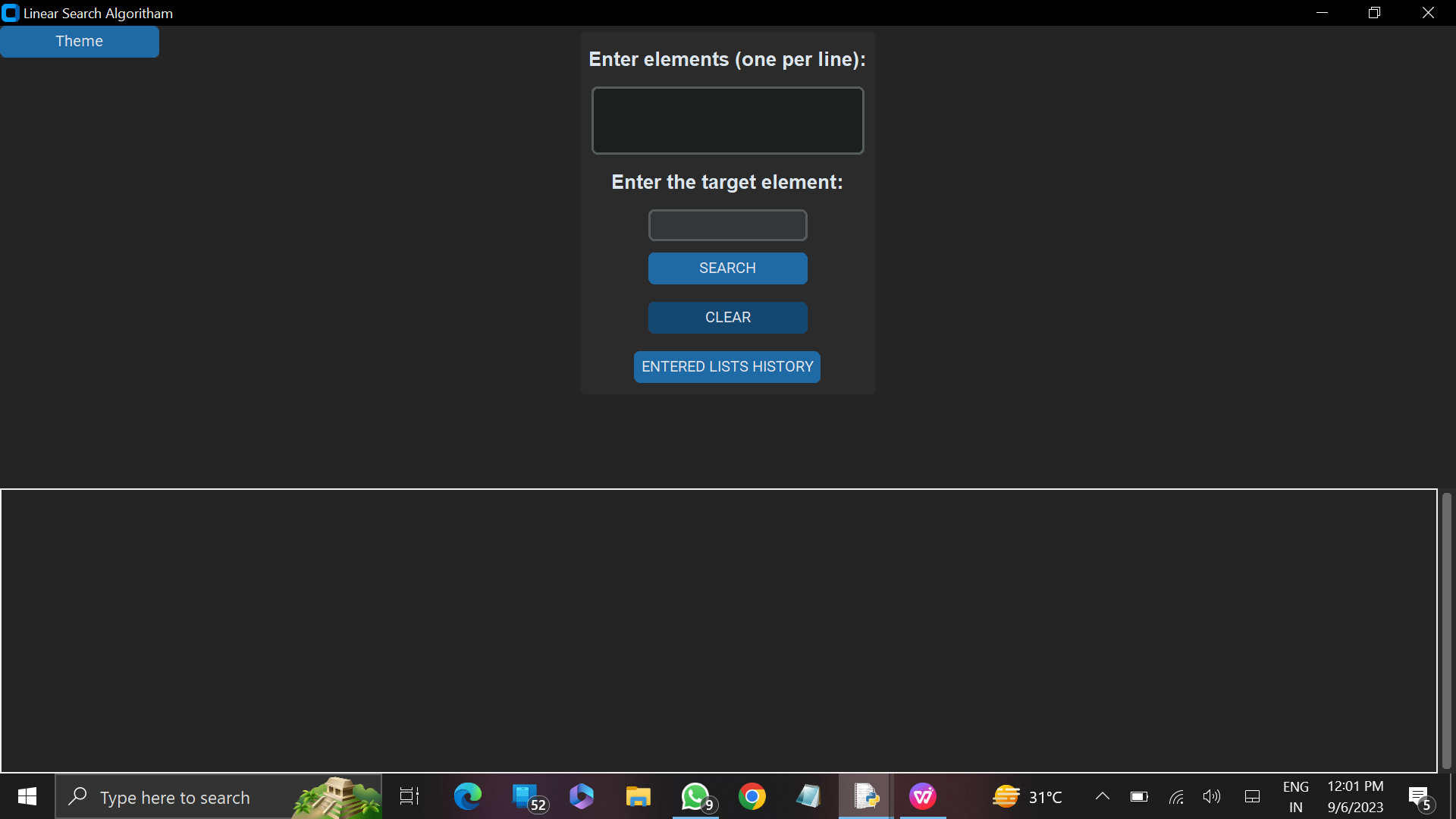
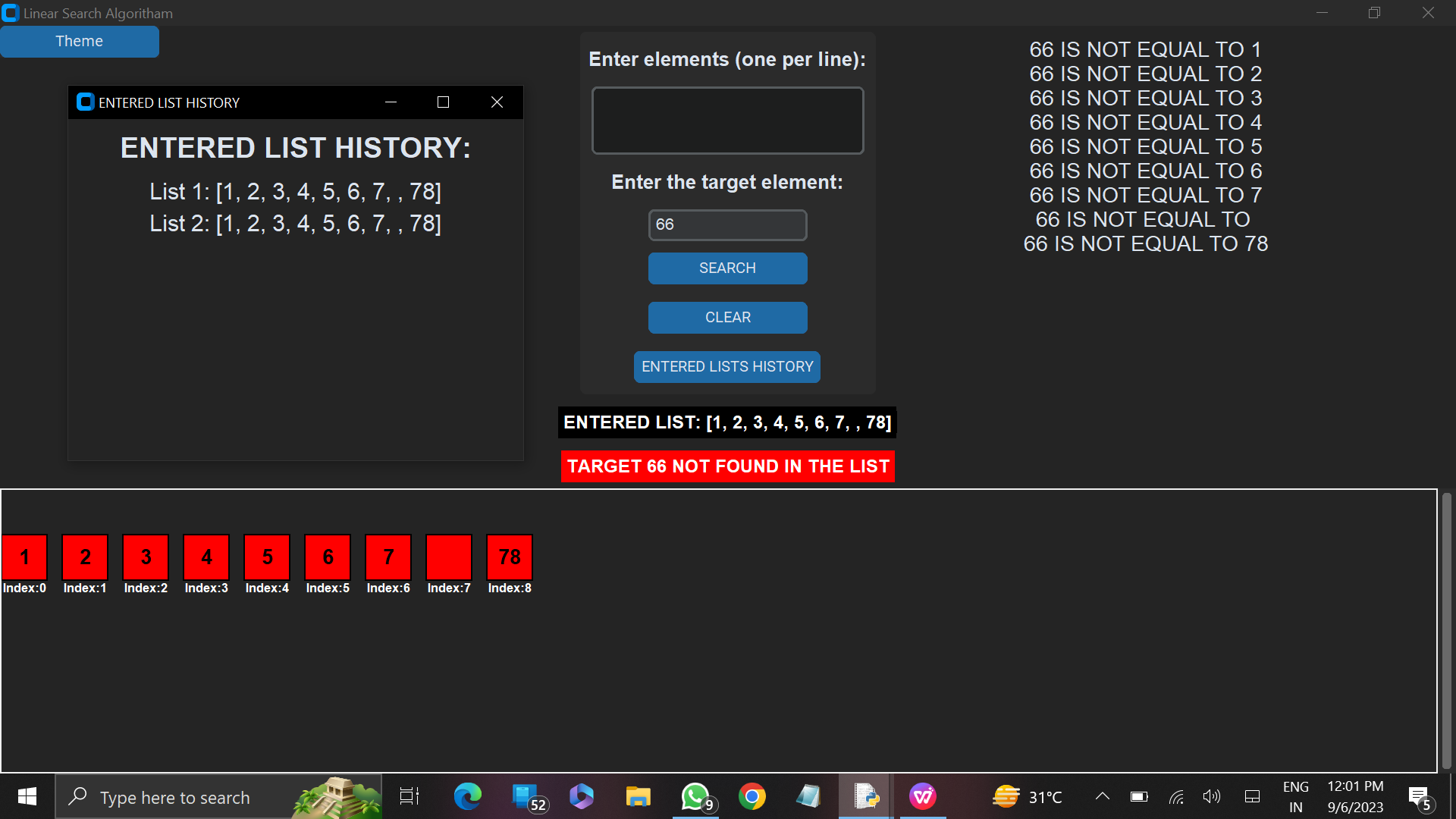
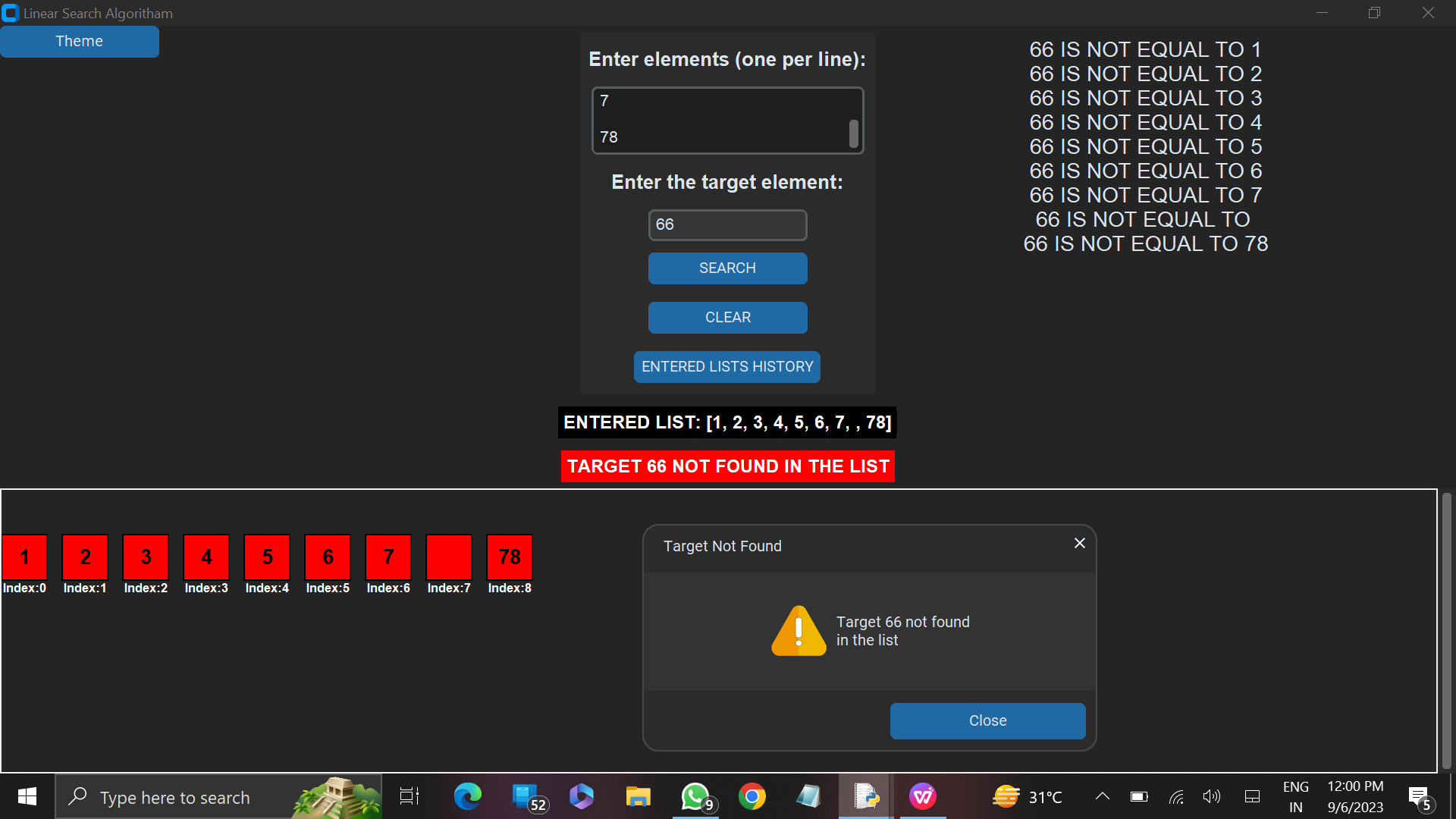
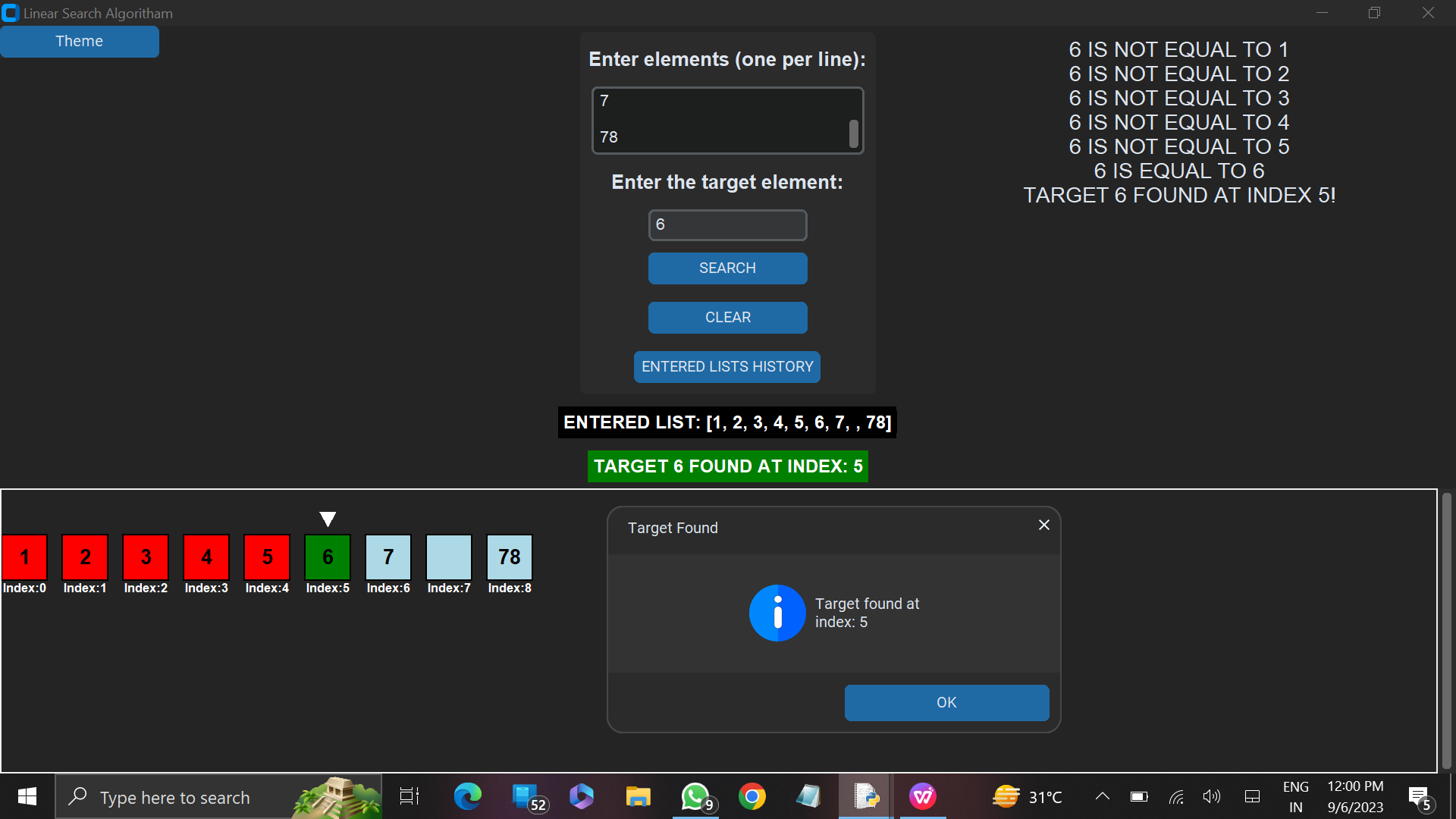
**Conclusion for Educational Purposes**

**In the context of education, the Linear Search Algorithm Program serves as a valuable resource for teaching and learning key concepts in computer science and algorithmic problem-solving. Here is a focused conclusion tailored to its educational significance:**

* **Enhancing Algorithm Comprehension: This program plays a pivotal role in enhancing students' comprehension of the linear search algorithm. By providing a real-time, visual representation of the algorithm's operation, it allows learners to witness how the algorithm iteratively scans a list to find a target element. This visual component aids in bridging the gap between abstract algorithmic concepts and practical implementation.**
* **Interactive Learning: The program's graphical user interface fosters an interactive learning environment. Students can actively participate in the search process by inputting lists and target elements. This hands-on approach reinforces their understanding of how the algorithm works in practice.**
* **Immediate Feedback: The program offers immediate feedback on search results, which is instrumental in the learning process. Whether a search is successful or unsuccessful, students receive visual cues and textual feedback, helping them understand the outcomes and reinforcing their grasp of the algorithm's behavior.**
* **Error Handling: Robust error handling mechanisms within the program teach students about the importance of input validation and error prevention. This educational aspect extends beyond the algorithm itself, imparting best practices for handling user input in software development.**
* **Customization and Themes: The program's theme-switching feature introduces the concept of user customization. This can spark discussions on user experience design principles and the importance of accommodating diverse user preferences, valuable topics in modern software development.**
* **Historical Reference: The ability to review previously entered lists provides students with a historical reference point for their learning journey. It encourages reflection and allows them to track their progress in understanding the algorithm over time.**

# FUTURE ENHANCEMEN:-

* **Speed Control: Allow users to control the speed of the search visualization for a better understanding of time complexity.**
* **Algorithm Variants: Include options to explore different search algorithms for comparative analysis.**
* **Complexity Analysis: Integrate tools for analyzing time and space complexity.**
* **Error Messages: Improve error messages and provide suggestions for input correction.**
* **Interactive Tutorials: Develop guided lessons for algorithmic concepts.**
* **Data Structure Integration: Extend the program to work with various data structures.**
* **User Profiles: Implement user profiles to track progress and history.**
* **Accessibility Features: Enhance accessibility with keyboard navigation and screen reader support.**
* **Multi-Platform Compatibility: Adapt the program for use on different platforms (web, mobile) to increase accessibility.**
* **OUTPUT**

****