PROJECT REPORT

NOORULAIN(14663)

SEMESTER : 3RD (A)

PROJECT NAME: ALGORITHM VISUALIZER

SUBJECT: DATA STRUCTURE AND ALGORITHM

SUBMITTED TO : SIR JAMAL ABDUL AHAD

YEAR : 2024 - 2025

PROJECT REPORT

Table of Contents

INTRODUCTION: 3
ABSTRACT: 3
OBJECTIVE : 3
OVERVIEW: 4
FEATURES : 4
CODE WORKFLOW: 4
ALGORITHMS 5
DEMONSTRARTION: 7
VISUAL REPRESENTATION : 8
EXAMPLE INPUT AND OUTPUT 10
APPLICATIONS 10
STRENGTHS OF THE CODE 11
CHALLENGES AND SOLUTIONS 11
LIMITATIONS 12
RECOMMENDATIONS 12
CONCLUSION 13

INTRODUCTION:

The Algorithm Visualizer project is designed to provide an intuitive and interactive platform for understanding sorting algorithms. This project focuses on three fundamental sorting techniques: Bubble Sort, Selection Sort, and Insertion Sort. Developed in Python using the Flet library, the visualizer enables users to observe the sorting process step-by-step or automatically, enhancing their comprehension of algorithm behavior. By incorporating user-friendly input methods, random array generation, and dynamic graphical representation, this tool serves as an educational resource for students, developers, and enthusiasts.

ABSTRACT:

This project aims to develop an Algorithm Visualizer that demonstrates the inner workings of Insertion Sort, Bubble Sort, and Selection Sort using the Python programming language and the Flet library. The visualizer will allow users to input arrays manually, or generate random arrays of different sizes. Once an array is selected, users can choose from the three sorting algorithms to visualize the sorting process in real time. The tool will feature two modes for visualization: auto-sort mode, where the sorting happens automatically, and step-wise mode, where the user can proceed through each sorting step manually. The visualizer will also provide a graph displaying the array elements, and the time complexity of the sorting process will be displayed dynamically, allowing users to analyze how each algorithm performs with various input sizes. This project is aimed at helping students, developers, and educators understand sorting algorithms in a clear and interactive manner.

OBJECTIVE :

The primary objective of this project is :

- To help users understand sorting algorithms through visual demonstrations.
- To allow real-time analysis of algorithm performance using graphical and numerical representations.
- To offer an interactive learning experience with user-controlled sorting options like auto-sort and step-wise sorting.
- To demonstrate time complexity and its impact on sorting efficiency with varying array sizes.

Overview:

The code is written in Python and utilizes the Flet library for creating an interactive UI. Users can select a sorting algorithm, input or generate an array, and visualize the sorting process in real-time.

FEATURES :

- **Array Input Options:**
    - Manual input through text fields for custom arrays.
    - Random array generation with adjustable sizes.
- **Algorithm Selection:**
    - Bubble Sort, Selection Sort, and Insertion Sort.
- **Visualization Modes:**
    - Step-wise Sorting: Users can navigate through each sorting step.
    - Auto-Sorting: The entire sorting process is animated automatically.
- **Graphical Representation:**
    - Array elements are represented as bars in a graph, dynamically updated at each step.
- **Time Complexity Display:**
    - Displays the Best Case and Worst Case time complexity for the selected algorithm.
- **Interactive UI:**
    - A clean, responsive interface built using the Flet library, enabling easy navigation and user interaction.

Code Workflow:

- **Input Handling:**
    - Users can manually enter the array or generate a random array using the random array generator.
    - The size of the array can be adjusted through sliders.
- **Algorithm Selection:**

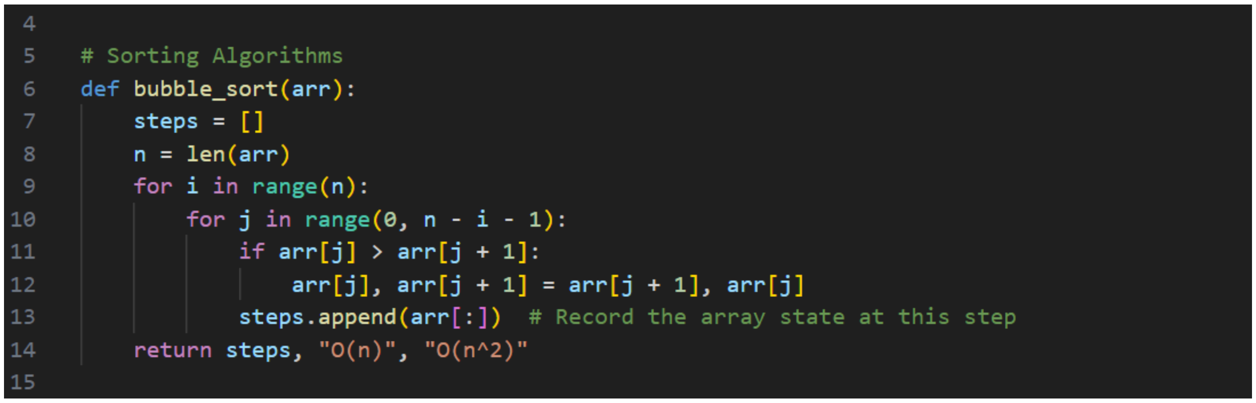
    - Users select one of the three sorting algorithms: Bubble Sort, Insertion Sort, or Selection Sort.

- **Sorting Logic:**
    - Based on the selected algorithm, the corresponding sorting function is called.
    - Sorting happens step-by-step for step-wise mode or automatically in auto-sort mode.
- **Visualization Engine:**
    - The array is displayed as a bar graph using the Flet library.
    - The graph updates dynamically to show changes in the array after each sorting step.
- **Output Display:**
    - The sorted array is displayed alongside the graph.
    - Time complexity for the selected algorithm is also shown.

ALGORITHMS

Bubble Sort:

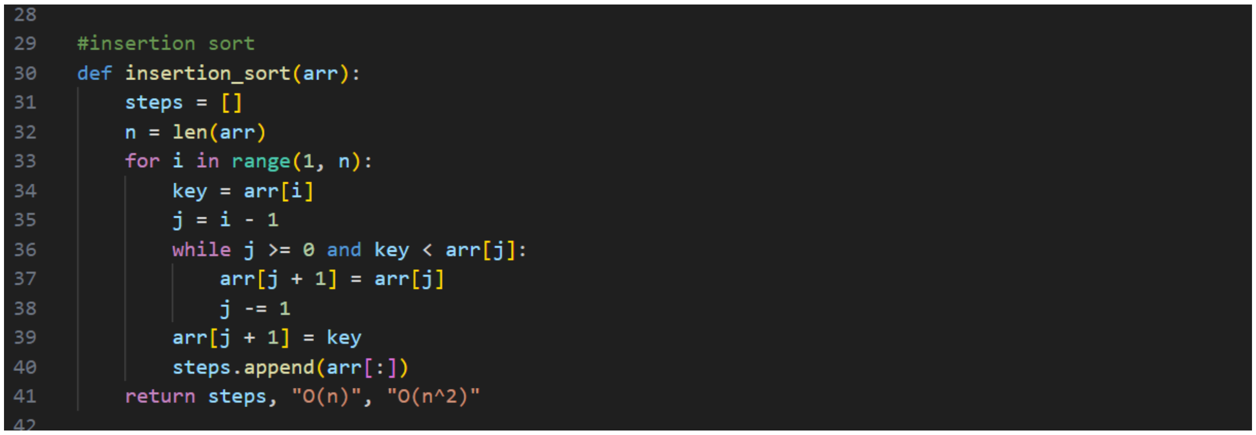
- Compares adjacent elements and swaps them if they are in the wrong order.
- Repeats the process for each element until the array is sorted.
- **Time Complexity:**
    - Best Case: O(n)
    - Worst Case: O(n^2)
- **Implementation :**



```python

# Sorting Algorithms
def bubble_sort(arr):
    steps = []
    n = len(arr)
    for i in range(n):
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
            steps.append(arr[:])  # Record the array state at this step
    return steps, "O(n)", "O(n^2)"

```

Insertion Sort:

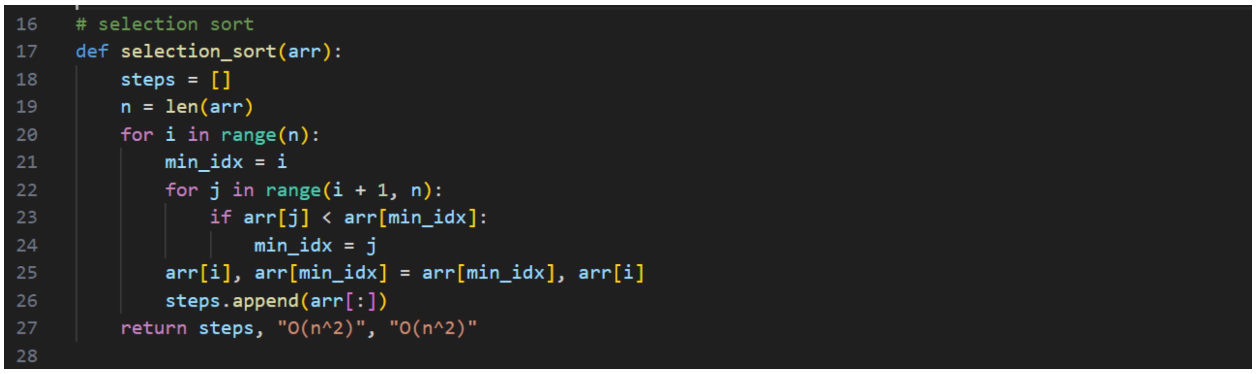
- Builds a sorted array by picking elements one by one and inserting them into the correct position.
- **Time Complexity:**
  - Best Case: O(n)
  - Worst Case: O(n^2)
- **Implementation :**



```python
#insertion sort
def insertion_sort(arr):
    steps = []
    n = len(arr)
    for i in range(1, n):
        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
        steps.append(arr[:])
    return steps, "O(n)", "O(n^2)"
```

Selection Sort:

- Finds the smallest element in the unsorted portion and swaps it with the first unsorted element.
- Repeats until the entire array is sorted.
- **Time Complexity:**
  - Best Case: O(n^2)
  - Worst Case: O(n^2)
- **Implementation :**



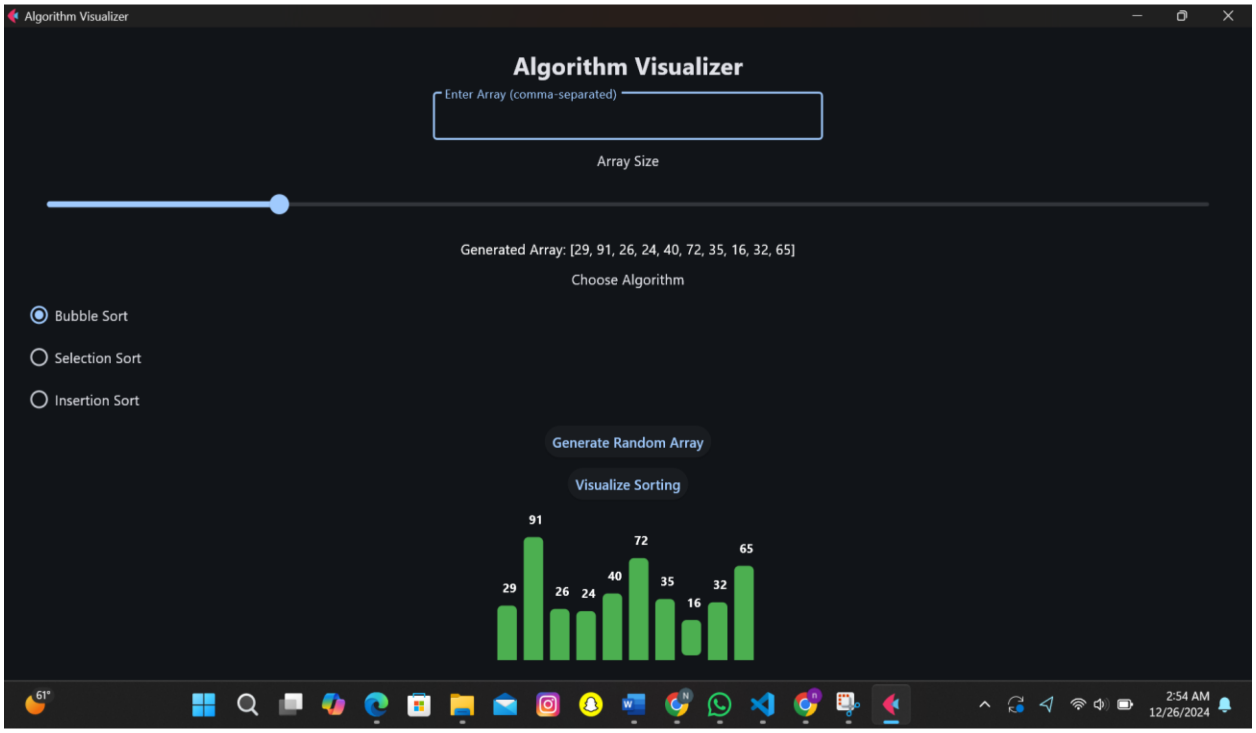
```python
# selection sort
def selection_sort(arr):
    steps = []
    n = len(arr)
    for i in range(n):
        min_idx = i
        for j in range(i + 1, n):
            if arr[j] < arr[min_idx]:
                min_idx = j
        arr[i], arr[min_idx] = arr[min_idx], arr[i]
        steps.append(arr[:])
    return steps, "O(n^2)", "O(n^2)"

```

DEMONSTRARTION:

- **Input Options**
    - Manually enter array values or generate a random array using the "Generate Random Array" button.
- **Choose Algorithm:**
    - Select one of the three sorting algorithms from the provided options.
- **Start Sorting:**
    - Click "Visualize Sorting" to begin.
    - Use "Next Step" to move through each step or "Auto Sort" to view the sorting process dynamically.
- **Observe Results:**
    - View the dynamically updated graph and time complexity details displayed below the sorting visualization.

VISUAL REPRESENTATION :

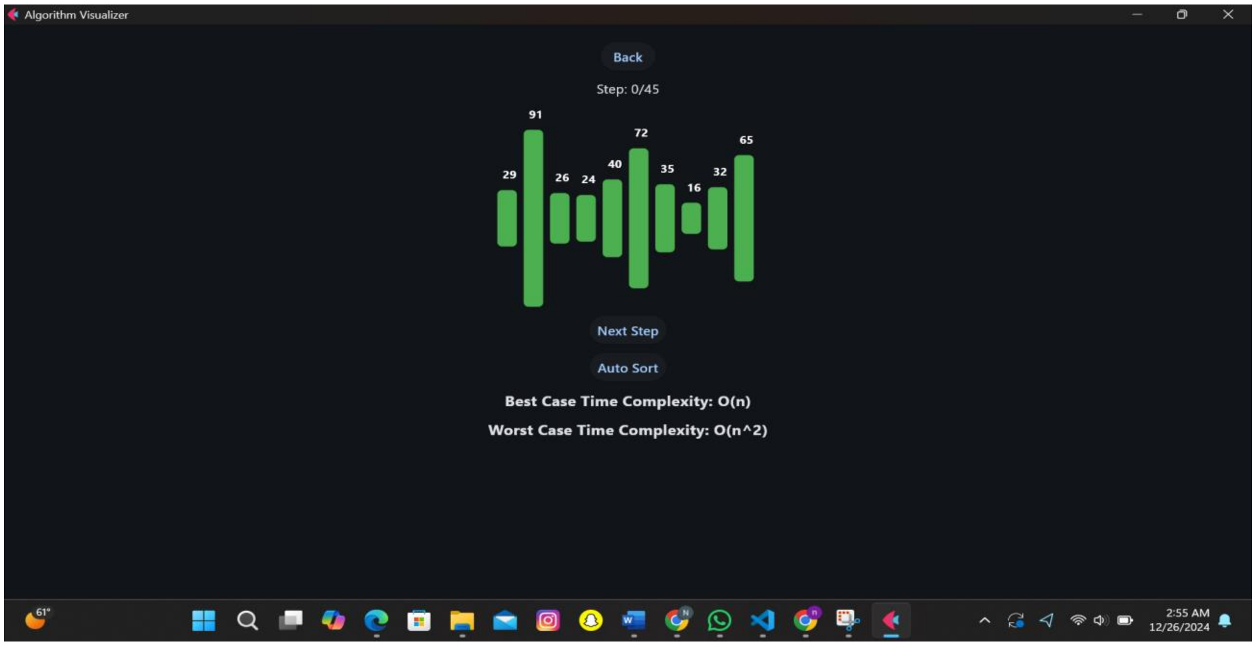
STEP1:



Generated Array: [29, 91, 26, 24, 40, 72, 35, 16, 32, 65]

Bubble Sort [✓]

STEP 2:



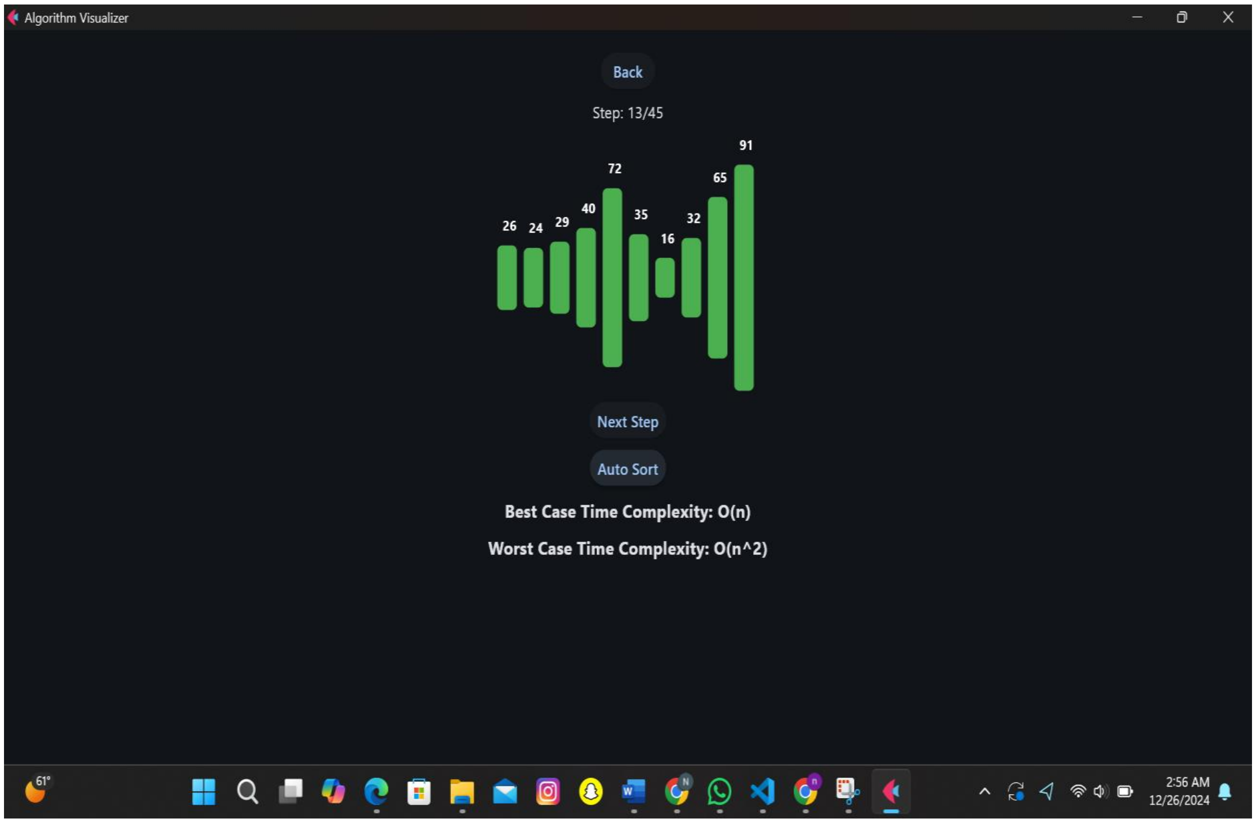
Step: 0/45

29 91 26 24 40 72 35 16 32 65

Best Case Time Complexity: O(n)

Worst Case Time Complexity: O(n^2)

STEP 3:



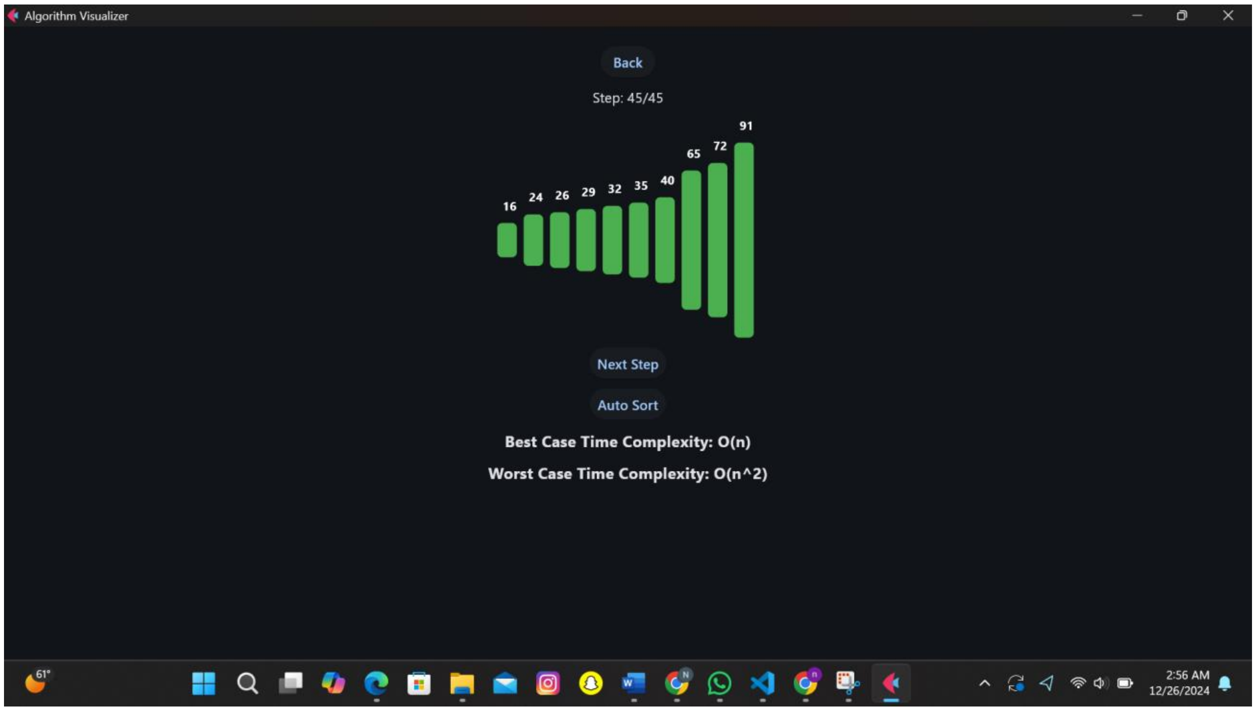
Step: 13/45

26 24 29 40 72 35 16 32 65 91

Best Case Time Complexity: O(n)

Worst Case Time Complexity: O(n^2)

STEP 4:



Best Case Time Complexity: O(n)

Worst Case Time Complexity: O(n^2)

EXAMPLE INPUT AND OUTPUT

Input:

- **Array:** [8, 5, 2, 9]
- **Algorithm:** Bubble Sort

Visualization Steps:

- [5, 8, 2, 9] (swap 8 and 5)
- [5, 2, 8, 9] (swap 8 and 2)
- [2, 5, 8, 9] (swap 5 and 2)

Output:

- **Sorted Array:** [2, 5, 8, 9]
- **Time Complexity:** O(n^2)

APPLICATIONS

- **Educational Tool:**
    - Helps students and educators visualize sorting algorithms in real-time.
- **Algorithm Comparison:**

    - Assists developers in comparing the performance of different sorting techniques.

- **Programming Tutorials:**
    - Enhances learning in online coding platforms or classrooms.

STRENGTHS OF THE CODE

- **Interactive Design:**
    - Allows users to customize inputs and control the sorting process.
- **Comprehensive Visualization:**
    - Step-by-step and auto-sorting modes enhance learning.
- **Time Complexity Analysis:**
    - Provides insights into the efficiency of sorting algorithms.
- **Scalability:**
    - Supports arrays of various sizes.

CHALLENGES AND SOLUTIONS

- **Challenge 1: Understanding Algorithm Behavior**
    - **Description:** Users may find it difficult to grasp how sorting algorithms process arrays step by step and how elements are swapped or moved.
    - **Solution:** The visualizer provides graphical visualizations of each sorting step, using dynamic bar heights to represent array values. This makes abstract concepts tangible. Detailed explanations can also be included for each step, highlighting the key operations being performed.
- **Challenge 2: Performance Insights**
    - **Description:** Users might struggle to understand the efficiency of algorithms, especially in terms of best-case and worst-case time complexities.
    - **Solution:** The application displays time complexity (best and worst cases) for the selected algorithm. Including additional comparisons between algorithms for the same array input could further enhance understanding.
- **Challenge 3: Interactive Learning**

- **Description:** Providing a flexible, engaging learning environment where users can experiment with different array sizes and algorithms can be challenging.
- **Solution:** The visualizer includes features like adjustable array sizes, random array generation, and manual input options. Adding customization for speed control during auto-sort or the ability to pause/resume the visualization could improve user experience further.

LIMITATIONS

- **Limited Algorithms:**
  - Currently supports only Bubble Sort, Insertion Sort, and Selection Sort.
- **Array Size Constraints:**
  - Performance may decrease for very large arrays due to real-time visualization overhead.
- **No Parallel Execution:**
  - Cannot visualize multiple sorting algorithms simultaneously for direct comparison.

RECOMMENDATIONS

- **Expand Algorithm Library:**
  - Add more sorting algorithms such as Merge Sort, Quick Sort, or Heap Sort. This will provide users with a broader understanding of different approaches to sorting.
- **Real-Time Explanations:**
  - Include text or tooltips that explain each step as it happens, such as "Swapping 3 and 5" or "Placing the smallest element at index 0."
- **Comparison Mode:**
  - Enable users to compare multiple algorithms side by side on the same array. This would help them understand how different sorting techniques perform under similar conditions.
- **Customization Options:**

  - Provide more flexibility in visualization settings, such as changing colors, adjusting the speed of auto-sorting, or customizing the size and range of array elements.
- **Mobile and Web Compatibility:**
  - Ensure the application is optimized for mobile devices or web browsers, making it accessible to a larger audience.
- **Educational Content:**
  - Include brief theoretical explanations of each algorithm, their use cases, and where they perform best or worst. This could be displayed as a pop-up or side panel.
- **Error Handling and Feedback:**
  - Improve error messages and add helpful tips if a user enters invalid input, such as non-numeric values or array sizes beyond the allowed range.
- **Gamified Learning:**
  - Introduce challenges or quizzes where users predict the number of steps an algorithm will take or choose the most efficient algorithm for a given scenario.
- **Performance Metrics:**
  - Show execution time for each algorithm based on the input array, helping users connect theoretical complexities with practical performance.
- **Step Control Enhancements:**
  - Add options to rewind or jump to specific steps in the visualization for a more detailed review of critical moments.

CONCLUSION

The Algorithm Visualizer project is a powerful educational tool that combines interactivity and visualization to make sorting algorithms more accessible and engaging. By showcasing the mechanics and performance of Bubble Sort, Insertion Sort, and Selection Sort, this tool bridges the gap between theoretical concepts and practical understanding. Despite its limitations, the visualizer lays a strong foundation for future expansions, including additional algorithms and enhanced functionality.

PROJECT LINK:

Noorulain63/Algorithm-Visualizer