Design and Analysis of Algorithm

Assignment 2

Report

Submitted by:

Name	Roll Number
Aqsa Fayaz	22i-1865
Muntaha Asim	22i-1868
Manal Aamir	22i-1940

Introduction:

Sorting algorithms are essential in computer science, impacting the efficiency of data processing in numerous applications. This report evaluates the performance of various sorting algorithms—Bubble Sort, Insertion Sort, Selection Sort, Merge Sort, Heap Sort, Quick Sort, Count Sort, Radix Sort, and Bucket Sort—by measuring their execution time on arrays of different sizes (100 to 100,000 elements).

Using Python, we analyze each algorithm's empirical time complexity through tables and logarithmic plots. Additionally, a web-based visualizer (bonus task) was developed to allow users to observe the sorting process in real time. This report provides insights into the scalability and efficiency of these algorithms, helping in the selection of the most suitable algorithm based on input size.

Choice of Language and why?

Python was chosen for implementing the sorting algorithms for several reasons:

1. **Simplicity and Readability**: Python's clean and easy-to-understand syntax makes it ideal for prototyping and demonstrating algorithms. It allows for clear implementation of sorting algorithms without excessive complexity.
2. **Rich Library Support**: Python offers a vast range of libraries such as numpy for efficient numerical operations and matplotlib for generating high-quality visualizations. These libraries are essential for analyzing and comparing sorting algorithms.
3. **Ease of Experimentation**: Python's dynamic nature and interactive environments (like Jupyter notebooks) allow quick testing, debugging, and visualization of results, making it perfect for exploring algorithm performance.
4. **Popularity and Versatility**: Python is widely used in academia and industry, particularly in data science and algorithm research. Its broad ecosystem ensures strong community support, tools, and resources.

These qualities make Python an ideal choice for both the implementation and analysis of sorting algorithms in this report.

2. Tools Used

a. Python Libraries

- **Numpy**:
  - Used for generating random arrays, which serve as the inputs for the sorting algorithms. Numpy's efficient handling of numerical operations allows for seamless generation and manipulation of large datasets, which is critical for testing sorting algorithms across different input sizes.
- **Matplotlib**:
  - Employed to plot time complexity graphs, allowing the visualization of sorting performance. This library helps present the empirical time complexity of the algorithms in a logarithmic scale, making it easier to analyze the efficiency of different sorting algorithms as the input size increases.
- **Time Module**:
  - The time module is utilized to measure the execution time of each sorting algorithm. By recording the start and end times of each algorithm's execution, it accurately captures the time taken to sort arrays of different sizes, providing essential data for analysis.

b. Graphical User Interface (Bonus Part)

- **HTML, CSS, and JavaScript**:
  - A front-end visualizer was created to make sorting algorithms more interactive and visually comprehensible.
  - **HTML** provides the structure of the web page, including dropdown menus, buttons, and the array container for visualization.
  - **CSS** is used to style the page, adding visual appeal through colors, fonts, and layout. It also helps in creating dynamic bar charts that change in height and color during the sorting process.
  - **JavaScript** is responsible for the interactive functionality. It handles the sorting logic, updates the visual representation of arrays in real-time, and manages user inputs (such as selecting an algorithm and adjusting array size or sorting speed).

JavaScript also controls the visualization of the sorting process, dynamically adjusting the height of bars as elements are sorted.

c. Sorting Algorithms Implemented

- **Bubble Sort**: A simple but inefficient algorithm with O(n^2) time complexity, used to demonstrate basic sorting operations.
- **Insertion Sort**: Another quadratic time algorithm, efficient for small datasets but not scalable.
- **Selection Sort**: Selects the minimum element and swaps it with the first unsorted element, similar in complexity to Bubble Sort and Insertion Sort.
- **Count Sort**: A linear time algorithm O($n + k$) ideal for sorting integers within a limited range.
- **Radix Sort**: A non-comparative sorting algorithm with linear complexity, efficient when sorting large datasets with numeric keys.
- **Bucket Sort**: Divides elements into multiple buckets and sorts each bucket individually, achieving linear time in the best case.
- **Merge Sort**: A divide-and-conquer algorithm with O($nlogn$) time complexity, consistently efficient for large datasets.
- **Quick Sort**: Known for its practical efficiency, Quick Sort uses a pivot element to partition the array, achieving O($nlogn$) time in average cases.
- **Heap Sort**: Builds a max-heap and repeatedly extracts the largest element, another O($nlogn$) algorithm used in practical applications.

3. Instructions to Run the Code

a. Running the Sorting Algorithms in Python

1. **Install Python**: Ensure that Python 3.x is installed on your machine. If not, download it from the official Python website.
2. **Install Required Libraries**:
    - Open a terminal or command prompt and run the following command to install the necessary Python libraries:

    pip install numpy matplotlib

    This will install Numpy for numerical operations and Matplotlib for plotting graphs.

3. **Run the Python Script**:
    - After installation, navigate to the directory where the provided Python script ***(task.py)*** is located.
    - Run the script using the command:

    python task.py

    - The script will execute all the sorting algorithms, measure their performance, and display the results in tabular format. It will also generate visual plots showing the empirical time complexity of each algorithm for different input sizes.
4. **Expected Output**:
    - A table summarizing the execution time (in milliseconds) of each sorting algorithm for arrays of sizes 100, 500, 1000, 5000, 10000, 50000, and 100000.
    - Logarithmic plots comparing the time complexity of the algorithms across these input sizes.

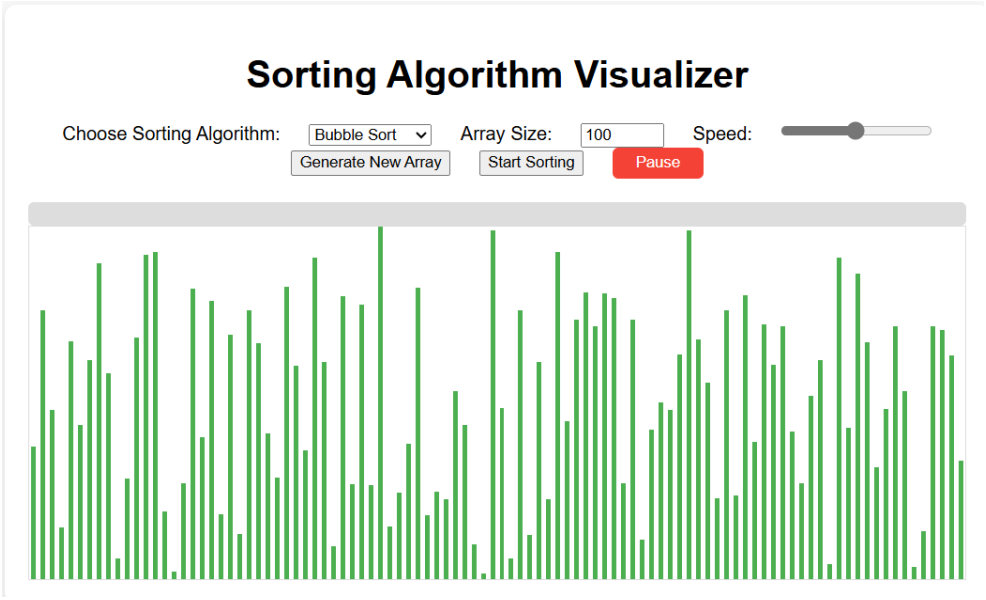
b. Running the GUI (Bonus Part)

1. **Open the HTML File**:
    - Locate the front.html file in your directory and open it using any modern web browser (e.g., Google Chrome, Mozilla Firefox).
2. **Using the Controls**:
    - **Select Sorting Algorithm**: Use the dropdown menu to select one of the sorting algorithms (e.g., Bubble Sort, Quick Sort, Merge Sort, etc.).
    - **Set Array Size**: Input the desired size of the array (between 5 and 100) using the size input field.
    - **Adjust Speed**: Use the slider to adjust the sorting speed from slow to fast.
3. **Start and Pause Sorting**:
    - Click on the "Start Sorting" button to begin the sorting process. The array bars will adjust dynamically as the sorting progresses.
    - You can pause the sorting process at any time by clicking the "Pause" button and resume sorting by clicking the same button again.
4. **Visualization**:
    - As the sorting progresses, the bars in the visualizer will change their height and color, representing the current state of the array. The GUI provides a real-time view of how each algorithm sorts the array step by step.

4. Screenshots

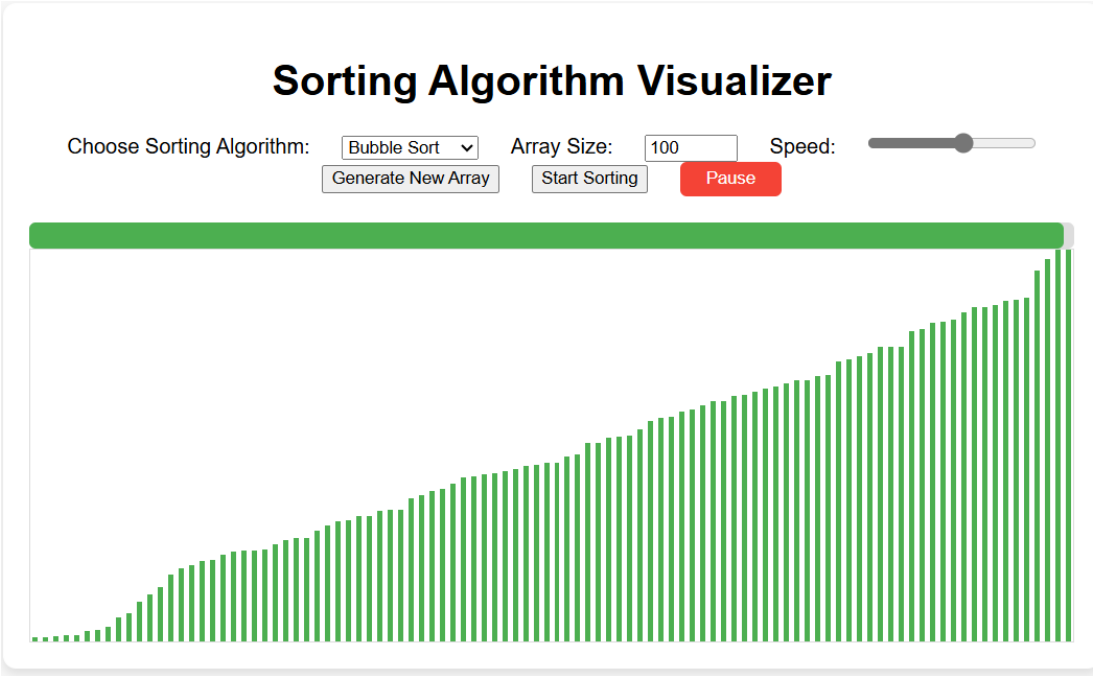
a. Initial Unsorted Arrays

- Before sorting begins, a randomly generated array of numbers is displayed as vertical bars of varying heights. Each bar’s height corresponds to the element's value in the array, providing a visual representation of the unsorted array.For example, with an array size of 100, the bars will range from very short (smallest numbers) to very tall (largest numbers), providing an intuitive understanding of the initial state before sorting.



b. Final Sorted Arrays for Each Algorithm

- Once an algorithm completes its sorting process, the final sorted array is displayed, with the bars arranged in ascending order of height. This visually demonstrates that the array has been successfully sorted from smallest to largest element.



c. Table and Plots

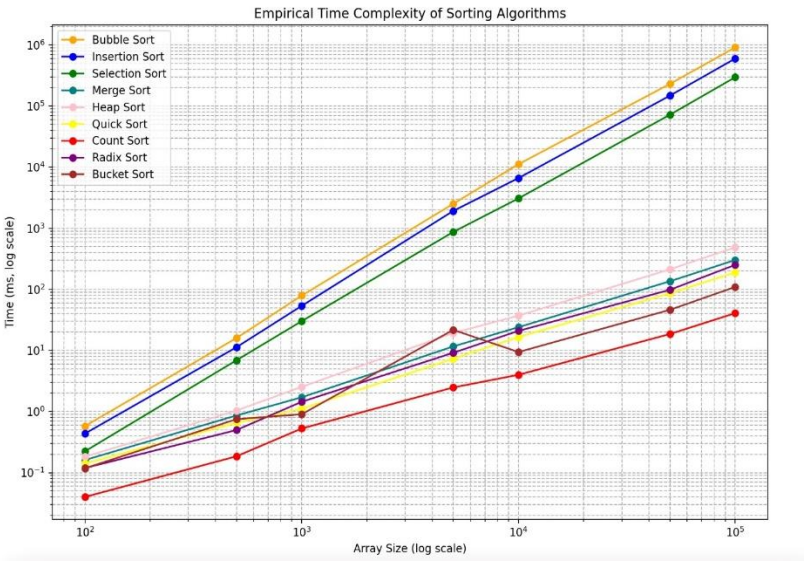
- **Table**: The table summarizes the performance of each sorting algorithm by listing the time (in milliseconds) it took to sort arrays of different sizes. These sizes range from 100 elements to 100,000 elements, allowing a clear comparison of how each algorithm scales.

Example:

- Bubble Sort takes 79 milliseconds to sort 1000 elements but over 912,000 milliseconds to sort 100,000 elements, highlighting its inefficiency for large datasets.
- Quick Sort and Merge Sort perform significantly better, maintaining reasonable execution times even as the input size increases.

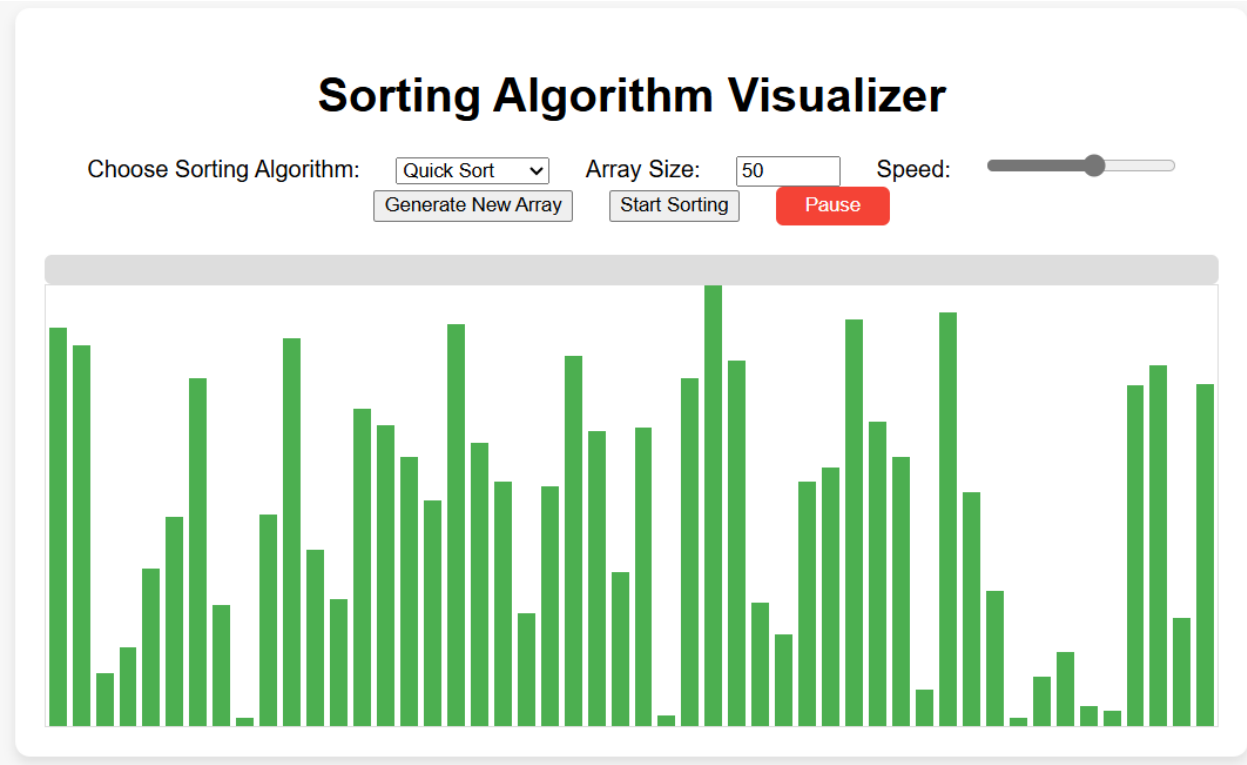
```
Array Size Bubble Sort (ms)    Insertion Sort (ms)  Selection Sort (ms)  Count Sort (ms)     Radix Sort (ms)     Bucket Sort (ms)    Merge Sort (ms)     Quick Sort (ms)     Heap Sort (ms)
-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
100         0.58                0.44                 0.22                 0.04                0.12                0.12                0.16                0.14                0.18
500         15.94               11.28                6.91                 0.19                0.50                0.75                0.86                0.65                1.04
1000        79.00               53.68                30.15                0.53                1.45                0.90                1.71                1.11                2.54
5000        2510.16             1917.78              868.38               2.48                9.16                21.73               11.64               7.31                19.29
10000       11154.50            6616.22              3075.68              3.99                20.88               9.42                24.01               16.59               37.11
50000       230470.77           148143.48            72125.19             18.59               98.38               46.24               135.66              86.03               212.53
100000      912313.18           598522.89            296027.79            40.76               251.55              109.57              302.05              189.53              486.83
manalaamir@Manals-MacBook-Pro assi_02 %
```

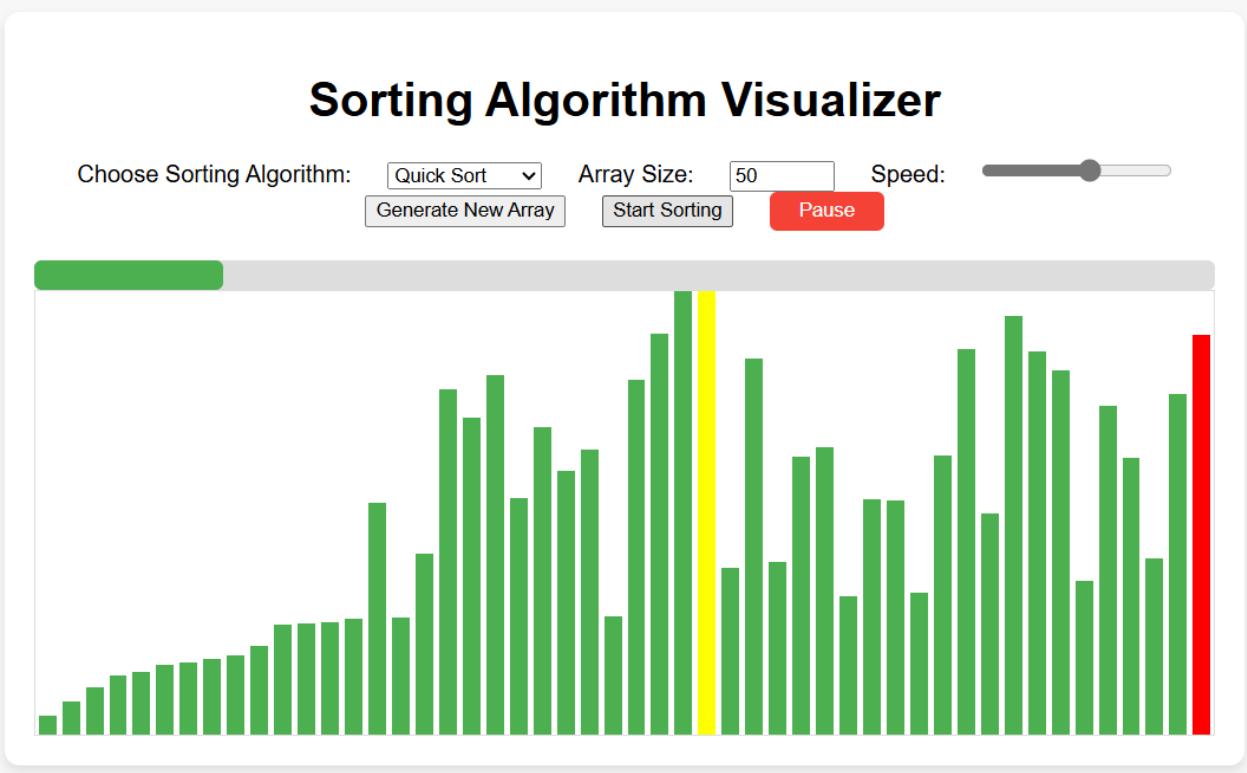
- **Plots**: The logarithmic plot visually compares the time complexity of each algorithm. The y-axis represents time (in milliseconds) on a logarithmic scale, while the x-axis represents the array size. This plot provides a clear understanding of how quickly the time required for sorting increases as the array size grows.

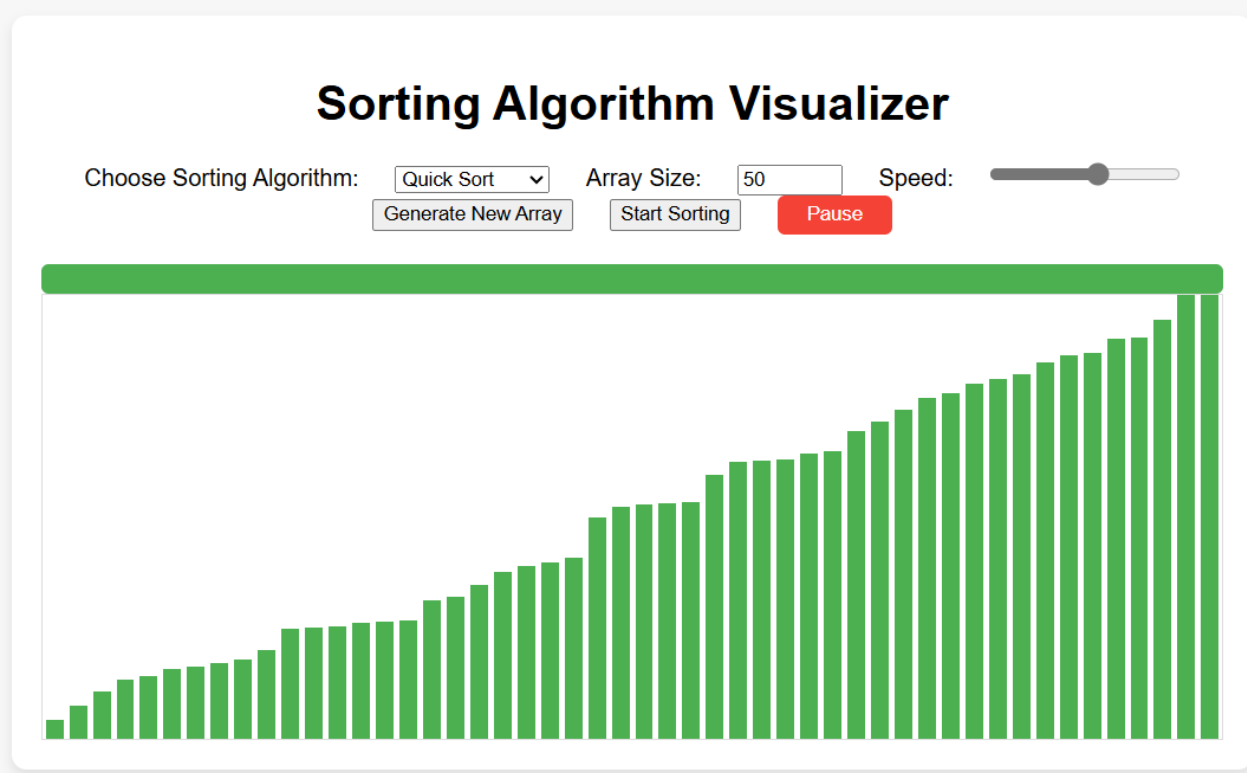


d. Visualization Screenshots of Sorting in Progress (Bonus Part)

- During the sorting process, the bars dynamically rearrange as the algorithm works through the array. Specific elements are highlighted in different colors (e.g., red for the currently compared elements) to give a clear visual representation of the algorithm's operations.
- For instance, in Quick Sort, the pivot element is highlighted, and the bars change color as the partitioning occurs, helping users visually grasp how the algorithm works.




Sorting Algorithm Visualizer
Choose Sorting Algorithm:
Quick Sort
Array Size:
50
Speed:
Generate New Array
Start Sorting
Pause


Sorting Algorithm Visualizer
Choose Sorting Algorithm:
Quick Sort
Array Size:
50
Speed:
Generate New Array
Start Sorting
Pause

5. References

- **Python Documentation**: Official Python documentation for understanding basic syntax, data structures, and standard modules like time that are used in this project.
    - https://docs.python.org/3/
- **Numpy Documentation**: Provides comprehensive guidance on generating and manipulating numerical data, which was essential for generating random arrays and performing array-based operations.
    - https://numpy.org/doc/
- **Matplotlib Documentation**: Offers detailed instructions on creating various types of visualizations, including the time complexity plots used in this project.
    - https://matplotlib.org/stable/contents.html
- **Sorting Algorithm Visualizer**: The front-end visualizer was implemented based on principles of HTML, CSS, and JavaScript to provide an interactive platform for understanding how sorting algorithms work in real time.