SORTING ALGORITHM VISUALIZER

Submitted by

SWAPAN KUMAR MONDAL

UID: 25MCI10301

in partial fulfilment for the award of the degree of

MASTER OF COMPUTER APPLICATIONS

IN

ARTIFICIAL INTELLIGENCE & MACHINE LEARNING

Chandigarh University

NOVEMBER 2025

ACKNOWLEDGEMENT

The successful completion of this Company Management System project would not have been possible without the guidance and support of several individuals. I would like to express my heartfelt gratitude to the following people who made this project a reality.

First and foremost, I would like to thank Chandigarh University for providing me with the opportunity to apply theoretical knowledge to a practical real-world project. I am grateful to my institution, University Institute of Computing, and our HOD, Dr. Krishan Tuli sir, for their support and encouragement throughout the MCA – AI/ML program.

I would like to extend my sincere thanks to Prof. Gurpreet Kaur Madam, Class Teacher for MCA department, University Institute of Computing, for providing me with the opportunity to work on this project and for her valuable guidance and support.

I am deeply grateful to my internal guide, Prof. Gurpreet Kaur Madam, University Institute of Computing, for her timely advice, necessary guidance, and valuable suggestions that helped me complete the project successfully.

Finally, I would like to thank my parents and other faculty members who have been a constant source of inspiration and support throughout the project.

1. Introduction: -

In the modern educational and computational environment, understanding the working mechanisms of sorting algorithms is essential for developing strong problem-solving and programming skills. However, visualizing how these algorithms operate step-by-step can be challenging without the support of interactive tools. To address this challenge, the **Sorting Algorithm Visualization System** provides an automated and web-based platform that allows users to explore and compare different sorting techniques through dynamic visualization.

This project is built using **Python (Flask framework)** for the backend, ensuring efficient data handling and real-time interaction between the client and server. The system receives user inputs, such as the array of numbers and the selected sorting algorithm, processes the data in Python, and returns step-by-step sorting progress as JSON responses. The backend implements multiple sorting algorithms—including **Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Quick Sort, and Heap Sort**—each designed to record intermediate sorting steps for clear visualization.

The primary objective of this system is to help students, developers, and educators understand the internal operations of popular sorting algorithms through animated demonstrations. By showing each transformation in the array as the algorithm progresses, the system provides a transparent view of how data is rearranged, making it easier to grasp algorithmic logic and efficiency.

By integrating **Flask-based APIs**, **Python logic**, and a **web-based visualization interface**, the Sorting Algorithm Visualization System offers an interactive and educational tool for algorithm learning. It bridges theoretical understanding and practical observation, promoting deeper insight into data structures and algorithmic behavior while fostering effective learning through visualization.

2. Background: -

In today's rapidly advancing digital world, understanding the internal functioning of algorithms is essential for developing strong computational thinking and analytical problem-solving skills. Sorting algorithms form the foundation of computer science and data processing, as they are used extensively in organizing, searching, and optimizing data operations. However, learners and developers often find it difficult to comprehend how these algorithms work internally, since the sorting process occurs quickly and invisibly in most programming environments.

To overcome this challenge, the **Sorting Algorithm Visualization System** was developed as an interactive, automated, and educational tool that demonstrates how various sorting techniques operate step by step. The system utilizes Python's **Flask framework** for backend logic and **JSON-based APIs** for seamless data communication with the frontend. It supports multiple algorithms—including **Bubble Sort, Selection Sort, Insertion Sort,**

Merge Sort, Quick Sort, and Heap Sort—each implemented to record and return every intermediate state of the sorting process.

By integrating algorithmic computation with real-time visualization, the project bridges the gap between theoretical understanding and practical demonstration. Each algorithm processes user-input arrays and returns the sorting sequence, allowing learners to observe the element comparisons, swaps, and data transformations that occur at each stage. This approach simplifies complex algorithmic concepts and helps users visually grasp how sorting efficiency and logic vary across different methods.

Unlike traditional static explanations or code-only examples, this system provides a **dynamic, web-based learning experience** accessible to students, educators, and enthusiasts. It encourages active engagement, enhances conceptual clarity, and fosters deeper understanding of algorithm behavior and time complexity. By combining modern web technology, backend computation, and interactive data representation, the **Sorting Algorithm Visualization System** contributes to smarter and more effective learning in the field of computer science education.

3. Objective: -

- **Step-by-Step Algorithm Visualization:** Capture and display each intermediate state of various sorting algorithms to facilitate a clear understanding of their internal workings
- **Support Multiple Algorithms:** Implement multiple sorting techniques including Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Quick Sort, and Heap Sort for comparative learning.
- **Interactive User Input:** Enable users to input custom arrays and select sorting algorithms to observe the sorting process dynamically.
- **Automated Step Recording:** Track all element comparisons and swaps during sorting to provide detailed step-by-step output for visualization.
- **Real-Time Feedback:** Provide instant feedback and sorting sequences through a web-based interface using Flask and JSON-based APIs.
- **Educational Insight:** Help learners and educators analyze the behavior, efficiency, and time complexity of different sorting algorithms.
- **Dynamic Result Generation:** Automatically return sorted sequences and all intermediate steps to support animation and learning.
- **Error Handling:** Manage invalid inputs or unsupported algorithm selections gracefully, ensuring a robust user experience.
- **Scalable System Design:** Structure the system to easily add new algorithms or expand functionality for advanced algorithmic demonstrations.
- **Web Integration and Visualization:** Utilize Flask for backend processing and seamless communication with a frontend interface for interactive algorithm visualization.

4. Technology used: -

- **Python:** The primary programming language used to implement the sorting algorithms, providing simplicity, flexibility, and powerful computational capabilities.
- **Flask:** A lightweight Python web framework utilized to build the backend, handle HTTP routes, process user requests, and communicate with the frontend through JSON-based APIs.
- **HTML, CSS, and JavaScript:** Employed to design a responsive and interactive web interface that allows users to input arrays, select sorting algorithms, and visualize sorting steps dynamically.
- **JSON:** Used for structured data exchange between the frontend and backend, transmitting user inputs and algorithm step sequences efficiently.
- **Python Standard Libraries:** Built-in modules such as copy, json, and os are used for array manipulation, data handling, and ensuring smooth backend operations.
- **Algorithm Implementation Techniques:** Step-tracking logic is applied to record every intermediate state of the array during sorting, enabling accurate visualization of comparisons, swaps, and recursive operations.
- **RESTful API Integration:** Facilitates real-time communication between the client-side interface and the Flask backend for immediate response and interactive experience.
- **Cross-Platform Compatibility:** The system is capable of running on Windows, macOS, or Linux environments where Python and Flask are supported, ensuring broad accessibility.
- **Data Visualization Readiness:** The backend structure supports integration with frontend visualization tools, allowing step-by-step animations of algorithm execution for educational purposes.
- **Scalable Architecture:** Designed to accommodate additional sorting algorithms or enhancements, making it suitable for future expansion and advanced algorithm demonstrations.

5. Requirement: -

Hardware Requirements:

1. **Computer or Laptop**: A machine capable of running Python and the necessary libraries.
2. **Processor (CPU)**: Intel i3 or higher / AMD equivalent (dual-core minimum, quad-core recommended for faster processing).
3. **Storage (HDD/SSD)**: Minimum 100 GB free storage (SSD preferred for faster read/write operations, especially with databases and Python scripts).
4. **Internet Connection**: Real time login and store the date in to the server database.

Software Requirements:

1. **Python**: Version 3.x installed on the machine as the programming language for development.
2. **Flask:** Lightweight Python web framework used to handle routing, API creation, and backend communication.
3. **HTML, CSS, and JavaScript:** Used to design the interactive web interface, allowing users to input arrays, select sorting algorithms, and view step-by-step visualization.
4. **JSON Module:** Handles structured data exchange between frontend and backend for transmitting user inputs and algorithm step sequences.
5. **Python Standard Libraries:** Includes copy, os, and json for array manipulation, data handling, and backend operations.
6. **Code Editor / IDE:** Tools such as VS Code, PyCharm, or Jupyter Notebook for writing, debugging, and running the Flask application efficiently.
7. **Web Browser:** Required to access and test the interactive web interface (e.g., Google Chrome, Mozilla Firefox, or Microsoft Edge).
8. **Operating System:** Compatible with Windows, macOS, or Linux that supports Python, Flask, and the required frontend technologies.
9. **Optional Web Server Environment:** For local testing, platforms like XAMPP or WAMP can be used if database logging or extended features are integrated in the future.

6. System Design

1. Architecture Overview

The **Sorting Algorithm Visualization System** follows a layered architecture that integrates the user interface, backend processing, and algorithm logic components:The system is built using a layered architecture that includes:

- **Client Layer (Frontend):** Built using HTML, CSS, and JavaScript, this layer provides an interactive interface where users can input an array of numbers and select a sorting algorithm. It communicates with the backend through AJAX and JSON-based API calls to fetch step-by-step sorting results for visualization.
- **Backend Layer (Flask Server):** Developed using Flask (Python framework), this layer handles all server-side logic, including input validation, sorting algorithm execution, and preparation of sorting steps for frontend visualization. Each algorithm is implemented to record intermediate steps, which are returned as JSON objects.
- **Algorithm Logic Layer (Python):** Contains implementations of Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Quick Sort, and Heap Sort, each designed to track and return every state change in the array. This allows dynamic observation of element comparisons, swaps, and recursive operations for educational purposes.
- **Data Communication:** The system uses RESTful API endpoints (e.g., /sort) to enable seamless and secure exchange of user input arrays and sorting step sequences between the frontend and backend.
- **Visualization Readiness:** The backend structure ensures that the JSON output can be easily consumed by frontend scripts to animate sorting steps as vertical bars, lists, or other graphical representations.

2. Workflow Diagram

The workflow of the Sorting Algorithm Visualization System can be summarized in the following steps:

1. **User Access:** User opens the web application → lands on the homepage with options to input an array and select a sorting algorithm.
2. **Array Input:** User enters a custom array of numbers through the frontend interface.
3. **Algorithm Selection:** User selects a sorting algorithm from the provided list.
4. **Data Submission:** The array and selected algorithm are sent via AJAX as a JSON request to the Flask backend endpoint /sort.
5. **Backend Processing:** Flask receives the request → executes the selected sorting algorithm in Python → records all intermediate sorting steps.
6. **Step Sequence Generation:** The backend returns a JSON response containing the complete step-by-step sequence of array states.
7. **Result Display:** The frontend dynamically visualizes the sorting process, animating comparisons and swaps to illustrate algorithm behavior.
8. **Error Handling:** If invalid input or unsupported algorithm is submitted, the backend returns an error message → frontend displays appropriate notification to the user.
9. **System Scalability:** New sorting algorithms or enhancements can be added to the backend without affecting the existing interface, ensuring extensibility.
10. **Interactive Learning:** Users can re-run the algorithm with different inputs or select alternative algorithms → backend processes the new request → visualization updates dynamically.

7. Code: -

Python Code: -

```
from flask import Flask, request, jsonify, send_file

app = Flask(__name__)

@app.route('/')
def index():
    return send_file('index.html')

# ---------------- 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.copy())
    return steps
```

```
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.copy())
    return steps

def insertion_sort(arr):
    steps = []
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
            steps.append(arr.copy())
        arr[j + 1] = key
        steps.append(arr.copy())
    return steps

def merge_sort_steps(arr):
    steps = []
    def merge_sort(a, start, end):
        if end - start > 1:
            mid = (start + end)//2
            merge_sort(a, start, mid)
            merge_sort(a, mid, end)
            L, R = a[start:mid], a[mid:end]
            i = j = 0
            k = start
            while i < len(L) and j < len(R):
                if L[i] < R[j]:
                    a[k] = L[i]
                    i += 1
                else:
                    a[k] = R[j]
                    j += 1
                k += 1
                steps.append(a.copy())
                while i < len(L):
                    a[k] = L[i]
                    i += 1
                    k += 1
                    steps.append(a.copy())
                while j < len(R):
                    a[k] = R[j]
                    j += 1
                    k += 1
                    steps.append(a.copy())
        merge_sort(arr, 0, len(arr))
        return steps

    def quick_sort_steps(arr):
        steps = []
        def quick_sort(a, low, high):
            if low < high:
                p = partition(a, low, high)
                quick_sort(a, low, p - 1)
                quick_sort(a, p + 1, high)
        def partition(a, low, high):
            pivot = a[high]
            i = low - 1
            for j in range(low, high):
                if a[j] <= pivot:
                    i += 1
                    a[i], a[j] = a[j], a[i]
                    steps.append(a.copy())
            a[i + 1], a[high] = a[high], a[i + 1]
            steps.append(a.copy())
            return i + 1
        quick_sort(arr, 0, len(arr) - 1)
        return steps

    def heap_sort_steps(arr):
        steps = []
        def heapify(n, i):
            largest = i
            l = 2 * i + 1
            r = 2 * i + 2
            if l < n and arr[l] > arr[largest]:
                largest = l
            if r < n and arr[r] > arr[largest]:
                largest = r
            if largest != i:
```

```
            arr[i], arr[largest] = arr[largest], arr[i]
            steps.append(arr.copy())
            heapify(n, largest)
    n = len(arr)
    for i in range(n // 2 - 1, -1, -1):
        heapify(n, i)
    for i in range(n - 1, 0, -1):
        arr[i], arr[0] = arr[0], arr[i]
        steps.append(arr.copy())
        heapify(i, 0)
    return steps

@app.route('/sort', methods=['POST'])
def sort_data():
    data = request.json
    arr = data.get("array", [])
    algo = data.get("algorithm", "bubble")

    algorithms = {
        "bubble": bubble_sort,
        "selection": selection_sort,
        "insertion": insertion_sort,
        "merge": merge_sort_steps,
        "quick": quick_sort_steps,
        "heap": heap_sort_steps
    }

    if algo not in algorithms:
        return jsonify({"error": "Invalid algorithm"}), 400

    steps = algorithms[algo](arr.copy())
    return jsonify({"steps": steps})

if __name__ == '__main__':
    app.run(debug=True)
```

Html, CSS, JavaScript Code:

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Sorting Algorithm Visualizer</title>
<style>
```

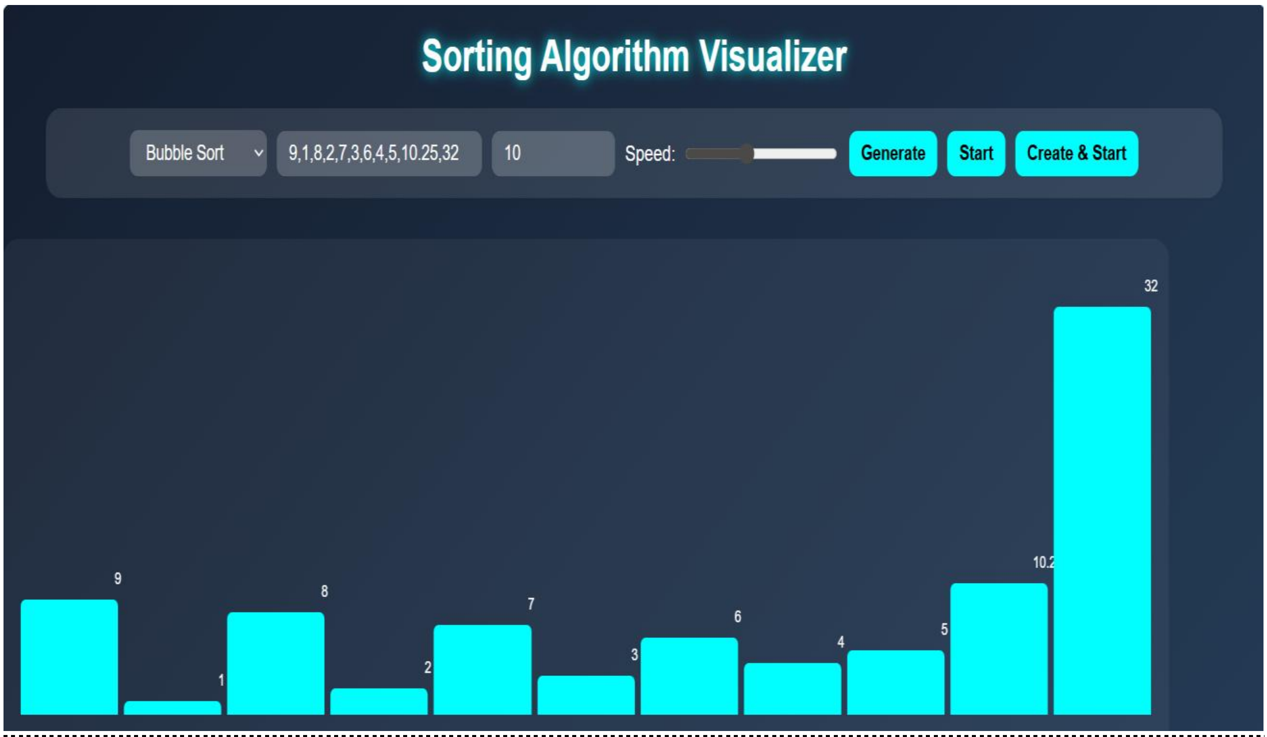
```
body {
  font-family: "Poppins", sans-serif;
  background: linear-gradient(135deg, #141e30, #243b55);
  color: white;
  margin: 0;
  padding: 0;
  text-align: center;
}

h1 {
  margin: 20px 0;
  text-shadow: 0 0 10px cyan;
  font-size: 2em;
}

.controls {
  display: flex;
  flex-wrap: wrap;
  justify-content: center;
  align-items: center;
  gap: 10px;
  background: rgba(255,255,255,0.1);
  padding: 15px 20px;
  border-radius: 15px;
  width: 90%;
  margin: 10px auto;
}

select, input, button {
  padding: 8px 12px;
  border: none;
  border-radius: 8px;
  font-size: 15px;
  outline: none;
}

select, input[type="number"], input[type="text"] {
  background: rgba(255,255,255,0.2);
  color: white;
  min-width: 100px;
}

button {
  background: cyan;
  color: black;
  font-weight: bold;
  cursor: pointer;
  transition: 0.3s;
}

button:hover {
  background: #00bcd4;
  color: white;
}

.bar-container {
  display: flex;
  align-items: flex-end;
  justify-content: center;
  flex-wrap: nowrap;
  margin-top: 30px;
  height: 350px;
  width: 90%;
  background: rgba(255, 255, 255, 0.05);
  border-radius: 15px;
  padding: 15px;
  overflow-x: auto;
}

.bar {
  flex: 1;
  margin: 0 3px;
  background: cyan;
  border-radius: 5px 5px 0 0;
  transition: height 0.3s ease, background 0.3s ease;
  position: relative;
  text-align: center;
}

.bar span {
  position: absolute;
  top: -22px;
  width: 100%;
  font-size: 12px;
  color: white;
}

.speed-control {
  display: flex;
  align-items: center;
```

```
    gap: 8px;
    color: white;
  }

  @media (max-width: 600px) {
    .controls {
      flex-direction: column;
      gap: 8px;
    }
    h1 {
      font-size: 1.5em;
    }
  }
</style>
</head>
<body>

<h1>Sorting Algorithm Visualizer</h1>

<div class="controls">
  <select id="algorithm">
    <option value="bubble">Bubble Sort</option>
    <option value="selection">Selection Sort</option>
    <option value="insertion">Insertion Sort</option>
    <option value="merge">Merge Sort</option>
    <option value="quick">Quick Sort</option>
    <option value="heap">Heap Sort</option>
  </select>

  <input type="text" id="userInput" placeholder="Enter numbers (e.g. 90,44,78)">
  <input type="number" id="size" min="5" max="50" value="10">

  <div class="speed-control">
    <label>Speed:</label>
    <input type="range" id="speed" min="50" max="1000" value="400">
  </div>

  <button onclick="generateArray()">Generate</button>
  <button onclick="startSorting()">Start</button>
  <button onclick="createAndStart()">Create & Start</button>
</div>

<div class="bar-container" id="barContainer"></div>

<script>
```

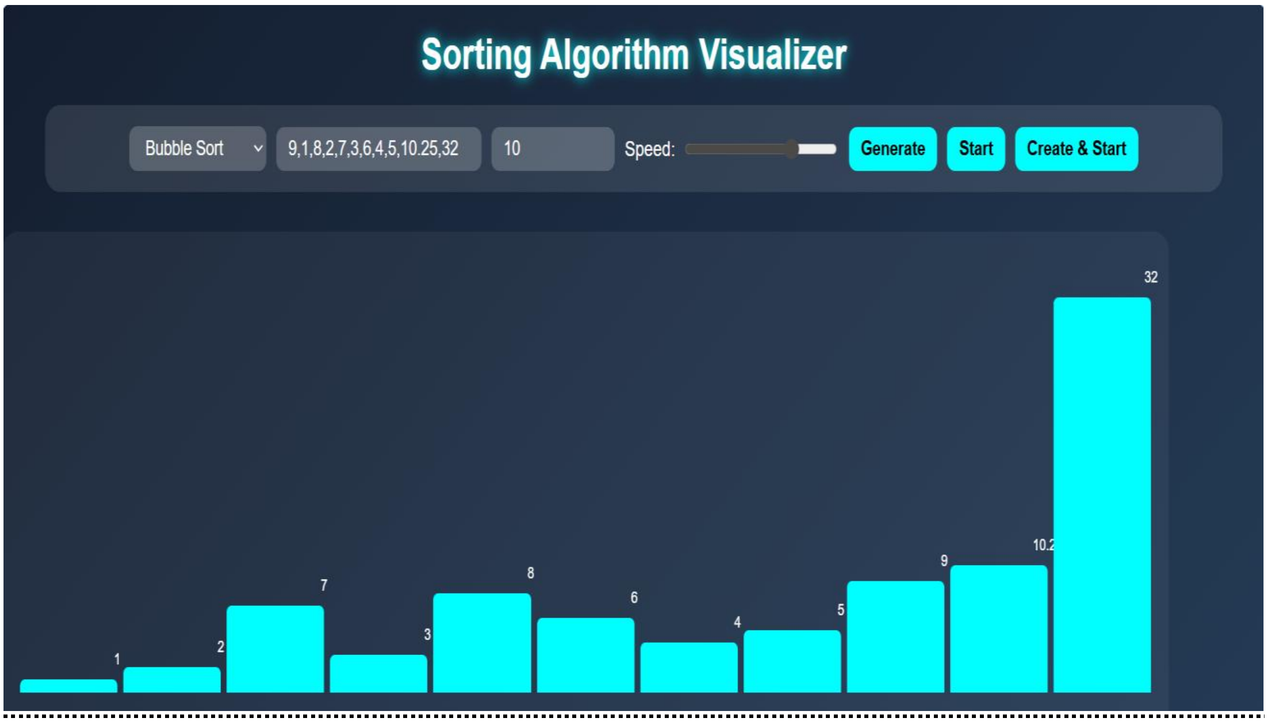
```
let array = [];
let delay = 400;

function generateArray() {
  const userInput = document.getElementById("userInput").value.trim();
  if (userInput) {
    array = userInput.split(",").map(Number);
  } else {
    const size = document.getElementById('size').value;
    array = Array.from({ length: size }, () => Math.floor(Math.random() * 100) + 5);
  }
  displayArray(array);
}

function displayArray(arr) {
  const container = document.getElementById('barContainer');
  container.innerHTML = '';
  const max = Math.max(...arr);
  const barWidth = Math.max(20, Math.floor(800 / arr.length));

  arr.forEach(num => {
    const bar = document.createElement('div');
    bar.classList.add('bar');
    bar.style.height = `${(num / max) * 300}px`;
    bar.style.minWidth = `${barWidth}px`;
    bar.innerHTML = `<span>${num}</span>`;
    container.appendChild(bar);
  });
}

async function startSorting() {
  const algorithm = document.getElementById('algorithm').value;
  delay = document.getElementById('speed').value;
  if (array.length === 0) {
    alert("Please generate or enter an array first!");
    return;
  }

  const response = await fetch('/sort', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ array, algorithm })
  });

  const data = await response.json();
  if (data.steps) await visualizeSteps(data.steps);
}

async function visualizeSteps(steps) {
  for (const step of steps) {
    displayArray(step);
    await new Promise(r => setTimeout(r, delay));
  }
}

async function createAndStart() {
  generateArray();
  await new Promise(r => setTimeout(r, 500));
  startSorting();
}

window.onload = generateArray;
</script>
</body>
</html>
```

8. OUTPUT:

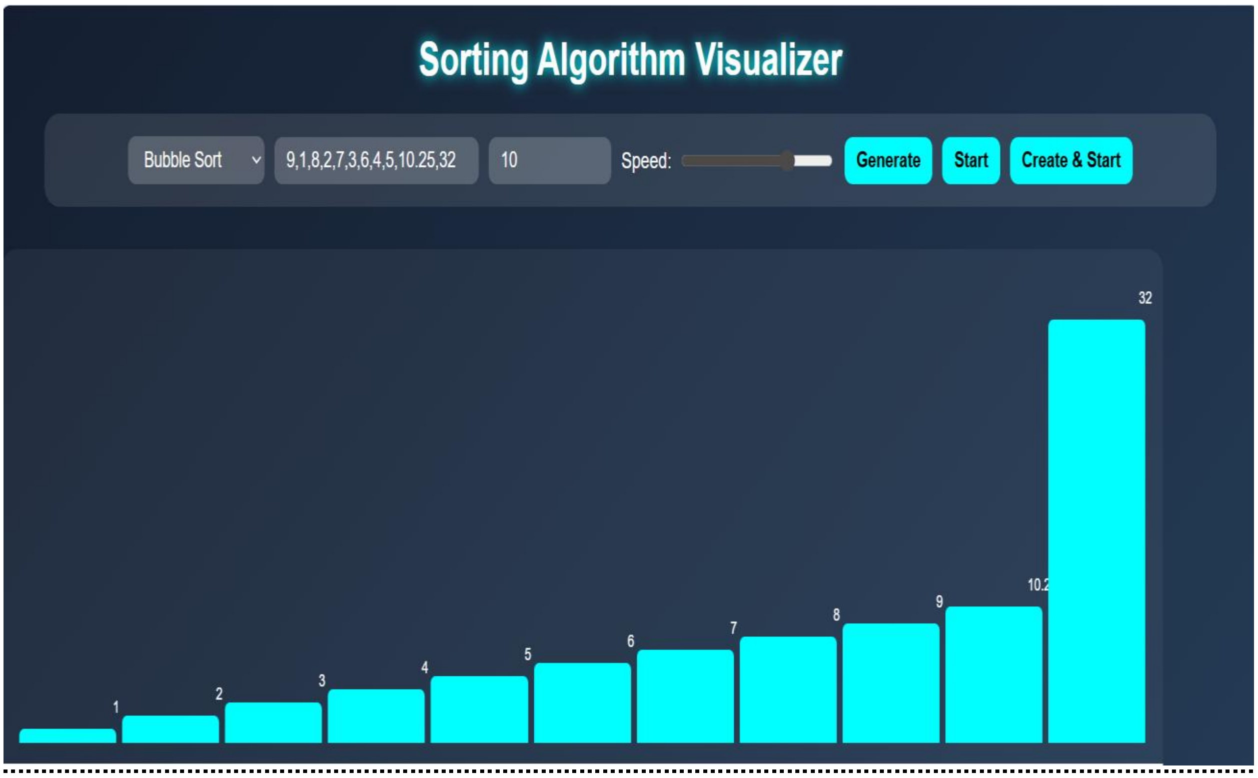
User Interface:



User Interface after Giving the data:



After Process the Data:



9. Conclusion

- User Authentication: A secure login and registration system ensures that only authorized users can access the application, protecting sensitive data and maintaining privacy.
- Efficient Data Storage: Disease and symptom data are efficiently managed using a MySQL database, ensuring fast retrieval and reliable storage.
- Dynamic Diagnosis System: The system dynamically processes user-selected symptoms and predicts the most probable disease in real time using data-driven comparison logic.
- Interactive Web Interface: Designed with HTML, CSS, and JavaScript, the interface provides a smooth and engaging user experience with clear symptom selection and diagnosis display.
- Backend Processing with Flask: Flask efficiently manages data flow between the frontend and database, ensuring quick response times and seamless communication through RESTful APIs.
- Data Accuracy and Reliability: The implemented algorithm ensures accurate disease prediction by analyzing symptom similarity and matching it with the database.
- Scalability and Extensibility: The modular design allows for easy integration of additional diseases, symptoms, or external medical APIs in future updates.
- Cross-Platform Compatibility: The system can be deployed on multiple operating systems like Windows, macOS, or Linux, ensuring flexibility for developers and users.
- Automation and Efficiency: Automated processing reduces manual workload and enhances the system's overall efficiency in diagnosing and data management.
- Practical Impact: This project demonstrates how data-driven technology and Python-based web development can assist in early disease prediction, promoting smarter healthcare solutions.

10.Reference: -

- https://github.com/Swapankumar2002/daaproject
- Python Software Foundation. Python Programming Language – Official Documentation. https://www.python.org/doc/
- Pallets Projects. Flask Web Framework – Official Documentation. https://flask.palletsprojects.com/
- MDN Web Docs. HTML, CSS, and JavaScript Guides. https://developer.mozilla.org/
- W3Schools. Web Development Tutorials *(*HTML, CSS, JavaScript, Flask Integration*)*. https://www.w3schools.com/
- Chandigarh university LMS Webpage.
- Get Semester YouTube Channels.