MODULE 3

1.Expression Evaluation (Infix to Postfix Conversion): Implement a calculator that converts infix expressions to postfix notation using stacks. Evaluate the postfix expression to return the result. Handle complex expressions with parentheses and operator precedence efficiently

HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Infix to Postfix Calculator</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div class="calculator">
        <h1>Postfix Calculator</h1>
        <input type="text" id="input" class="input-field" placeholder="Enter expression" />
        <div class="buttons">
            <button class="btn" onclick="appendToExpression('1')">1</button>
            <button class="btn" onclick="appendToExpression('2')">2</button>
            <button class="btn" onclick="appendToExpression('3')">3</button>
            <button class="btn" onclick="appendToExpression('+')">+</button>
            <button class="btn" onclick="appendToExpression('4')">4</button>
            <button class="btn" onclick="appendToExpression('5')">5</button>
            <button class="btn" onclick="appendToExpression('6')">6</button>
            <button class="btn" onclick="appendToExpression('-')">-</button>
            <button class="btn" onclick="appendToExpression('7')">7</button>
            <button class="btn" onclick="appendToExpression('8')">8</button>
            <button class="btn" onclick="appendToExpression('9')">9</button>
            <button class="btn" onclick="appendToExpression('*')">*</button>
```

```
      <button class="btn" onclick="appendToExpression('0')">0</button>
          <button class="btn" onclick="appendToExpression('.')">.</button>
          <button class="btn" onclick="appendToExpression('(')">(</button>
          <button class="btn" onclick="appendToExpression(')')">)</button>

          <button class="btn clear" onclick="clearExpression()">C</button>
          <button class="btn" onclick="appendToExpression('/')">/</button>
          <button class="btn" onclick="evaluateExpression()">=</button>
      </div>
      <div class="output">
          <h3>Postfix Expression: <span id="postfix-output"></span></h3>
          <h3>Result: <span id="result-output"></span></h3>
      </div>
   </div>
 <script src="script.js"></script>
</body>
</html>
```

CSS:

```
body {
    font-family: Arial, sans-serif;
    background-color: #f4f7f6;
    margin: 0;
    padding: 0;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
}
 .calculator {
    background: white;
    padding: 20px;
    border-radius: 10px;
    box-shadow: 0 10px 20px rgba(0, 0, 0, 0.1);
    width: 300px;
    text-align: center;
}
 h1 {
    margin-bottom: 10px;
    color: #333;
}
 .input-field {
    width: 100%;
    padding: 10px;
    font-size: 18px;
```

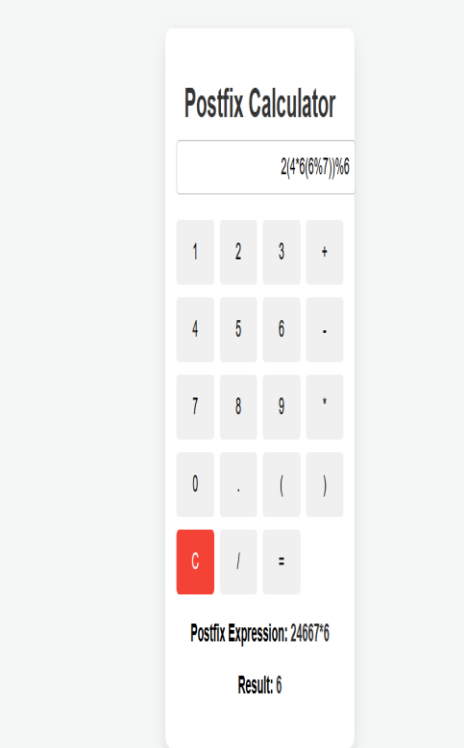
```
    margin-bottom: 20px;
    border: 1px solid #ccc;
    border-radius: 5px;
    text-align: right;
}
 .buttons {
    display: grid;
    grid-template-columns: repeat(4, 1fr);
    grid-gap: 10px;
}
 .btn {
    padding: 15px;
    font-size: 18px;
    border: none;
    border-radius: 5px;
    background-color: #f0f0f0;
    cursor: pointer;
    transition: background-color 0.3s;
}
 .btn:hover {
    background-color: #ddd;
}

.clear {
    background-color: #f44336;
    color: white;
}
 .clear:hover {
    background-color: #d32f2f;
}

.output {
    margin-top: 20px;
}
 #postfix-output,
#result-output {
    font-weight: bold;
    color: #333;
}
```

JavaScript:

```
function appendToExpression(value) {
   document.getElementById('input').value += value;
}
 function clearExpression() {
 document.getElementById('input').value = '';
    document.getElementById('postfix-output').textContent = '';
    document.getElementById('result-output').textContent = '';
}
function infixToPostfix(infix) {
    let stack = [];
```

```
    let postfix = [];
let precedence = {'+': 1, '-': 1, '*': 2, '/': 2, '(': 0};


    for (let i = 0; i < infix.length; i++) {
        let char = infix[i];


        if (/[0-9.]/.test(char)) {
            postfix.push(char);
        } else if (char === '(') {
            stack.push(char);
        } else if (char === ')') {
            while (stack.length && stack[stack.length - 1] !== '(') {
                postfix.push(stack.pop());
            }
            stack.pop();
        } else if (/[+\-*/]/.test(char)) {
            while (stack.length && precedence[stack[stack.length - 1]] >= precedence[char]) {
                postfix.push(stack.pop());
            }
            stack.push(char);
        }
    }
     while (stack.length) {
        postfix.push(stack.pop());
    }
     return postfix.join('');
}
function evaluatePostfix(postfix) {
    let stack = [];
    for (let i = 0; i < postfix.length; i++) {
        let char = postfix[i];
        if (/[0-9.]/.test(char)) {
            stack.push(parseFloat(char));
        }
    else {
            let b = stack.pop();
            let a = stack.pop();
        if (char === '+') stack.push(a + b);
   else if (char === '-') stack.push(a - b);
    else if (char === '*') stack.push(a * b);
   else if (char === '/') stack.push(a / b);
        }
    }
    return stack.pop();
}
function evaluateExpression() {
    let infix = document.getElementById('input').value.replace(/\s+/g, '');
    if (!infix) {
        alert("Please enter a valid expression");
        return;
```

```
    }

    let postfix = infixToPostfix(infix);
    let result = evaluatePostfix(postfix);

    document.getElementById('postfix-output').textContent = postfix;
    document.getElementById('result-output').textContent = result;
}
```



2.Online Ticketing System (Priority Queue): Design an online ticketing system using a priority queue where VIP customers are served first. Regular customers are served based on their order of arrival. Simulate ticket booking, cancellation, and serve operations, ensuring the system works under heavy traffic conditions.

HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Online Ticketing System</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div class="ticketing-system">
        <h1>Online Ticketing System</h1>
        <div class="input-section">
            <input type="text" id="customer-name" placeholder="Enter Customer Name" class="input-field">
```

```
        <select id="customer-type" class="input-field">
            <option value="VIP">VIP</option>
            <option value="Regular">Regular</option>
        </select>
        <button class="btn" onclick="addCustomer()">Book Ticket</button>
    </div>
    <div class="queue-section">
        <h2>Queue</h2>
        <ul id="queue-list"></ul>
    </div>
  <div class="control-buttons">
        <button class="btn" onclick="serveCustomer()">Serve Customer</button>
        <button class="btn" onclick="cancelTicket()">Cancel Ticket</button>
    </div>
    <div class="output-section">
  <h3>  Currently Serving: <span id="currently-serving">None</span></h3>
    </div>
  </div>

  <script src="script.js"></script>
</body>
</html>
```

CSS:

```
body {
  font-family: Arial, sans-serif;
  background-color: #f4f7f6;
  margin: 0;
  padding: 0;
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100vh;
}
.ticketing-system {
  background-color: white;
  padding: 40px;
  border-radius: 10px;
  box-shadow: 0 10px 20px rgba(0, 0, 0, 0.1);
  width: 450px;    text-align: center;
  margin: 20px;
}
h1 {
  color: #333;
  margin-bottom: 30px; }
.input-section {
```

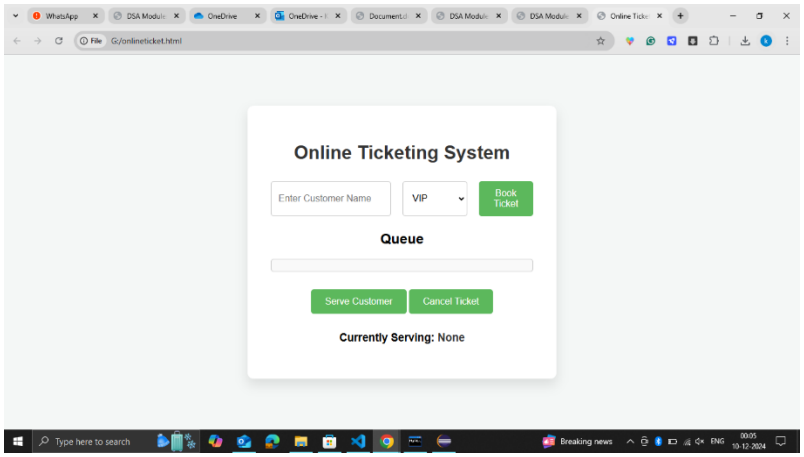
```
    display: flex;
    gap: 20px;
    margin-bottom: 25px;
    justify-content: center;
}
 .input-field {
    padding: 12px;
    font-size: 16px;
    border: 1px solid #ccc;
    border-radius: 5px;
    width: 180px;
}

select {
    padding: 12px;
    font-size: 16px;
    border: 1px solid #ccc;
    border-radius: 5px;
    width: 180px;
}

.btn {
    padding: 12px 24px; /* Keeping buttons the same size as before */
    font-size: 16px;
    background-color: #5cb85c;
    color: white;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    transition: background-color 0.3s;
}
 .btn:hover {
    background-color: #4cae4c;
}
 .queue-section {
    margin-bottom: 30px; /* Increased margin to make the section more spacious */
}
 ul {
    list-style-type: none;
    padding: 0;
    margin: 0;
    max-height: 200px;
    overflow-y: auto;
    border: 1px solid #ccc;
    border-radius: 5px;
    padding: 10px;
    background-color: #f9f9f9;
}
 li {
    padding: 10px;
    border-bottom: 1px solid #ddd;
}
```

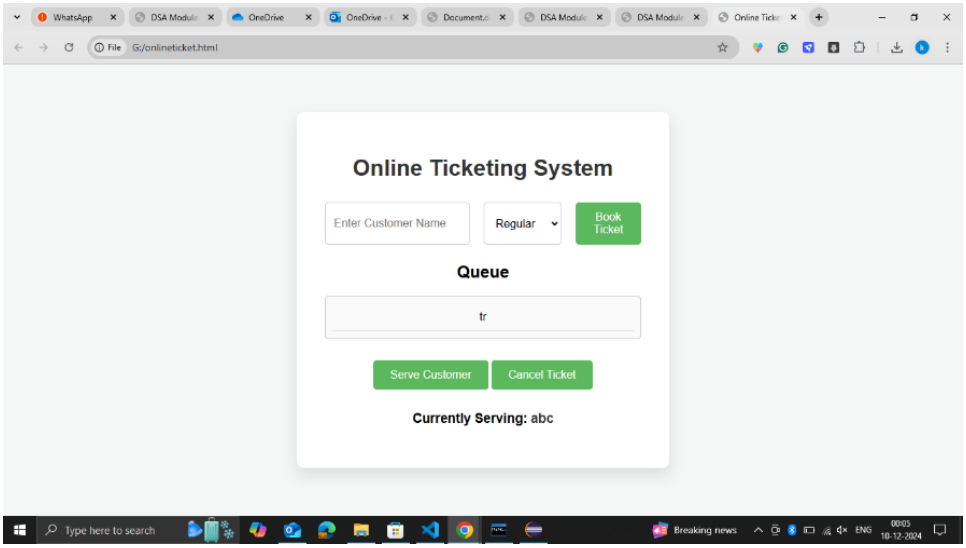
```
li.vip {
    color: #d9534f;
    font-weight: bold;
}
.control-buttons {
    margin-bottom: 25px; /* Added more spacing below the control buttons */
}
.output-section {
    margin-top: 30px; /* Increased top margin for better separation */
}
#currently-serving {
    font-weight: bold;
    color: #333;
}
```

JavaScript:

```
let queue = [];

function addCustomer() {
    const customerName = document.getElementById('customer-name').value;
    const customerType = document.getElementById('customer-type').value;

    if (customerName.trim() === '') {
        alert('Please enter a valid customer name.');
        return;
    }

    const customer = {
        name: customerName,
        type: customerType
    };
    if (customerType === 'VIP') {
        queue.unshift(customer);
    } else {
        queue.push(customer);
    }
  document.getElementById('customer-name').value = '';
updateQueueList();
}

function updateQueueList() {
    const queueList = document.getElementById('queue-list');
    queueList.innerHTML = '';
    queue.forEach((customer) => {
        const li = document.createElement('li');
        li.textContent = customer.name;
        if (customer.type === 'VIP') {
            li.classList.add('vip');
```

```
        }
        queueList.appendChild(li);
    });
}

function serveCustomer() {
    if (queue.length === 0) {
        alert('No customers in the queue to serve.');
        return;
    }

    const customer = queue.shift();
    document.getElementById('currently-serving').textContent = customer.name;
    updateQueueList();
}

function cancelTicket() {
    const customerName = document.getElementById('customer-name').value;
    if (customerName.trim() === '') {
        alert('Please enter a valid customer name to cancel.');
        return;
    }
const index = queue.findIndex((customer) => customer.name === customerName);
    if (index === -1) {
        alert('Customer not found in the queue.');
        return;
    }
    queue.splice(index, 1);
    updateQueueList();
}
```


Online Ticketing System
Enter Customer Name
VIP
Book Ticket
Queue
Serve Customer
Cancel Ticket
Currently Serving: None


Online Ticketing System
Enter Customer Name
Regular
Book Ticket
Queue
tr
Serve Customer
Cancel Ticket
Currently Serving: abc

3. Undo-Redo Functionality for a Code Editor: Create an undo-redo feature using two stacks to track changes made in a code editor. As the user performs actions (e.g ., writing, deleting text), track each action and allow them to undo or redo changes.

HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Code Editor with Undo/Redo</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <div class="editor-container">
    <h1>Code Editor with Undo/Redo</h1>

    <!-- Buttons for Undo/Redo -->
    <div class="controls">
      <button class="btn" id="undo-btn" onclick="undo()">Undo</button>
      <button class="btn" id="redo-btn" onclick="redo()">Redo</button>
    </div>

    <textarea id="code-editor" class="editor" oninput="trackChanges()" placeholder="Start coding here..."></textarea>

    <div class="status">
      <p><strong>Current Action:</strong> <span id="action-status">None</span></p>
    </div>
  </div>

  <script src="script.js"></script>
</body>
</html>
```

CSS:

```
body {
  font-family: Arial, sans-serif;
  background-color: #f4f7f6;
  margin: 0;
  padding: 0;
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100vh;
}
```

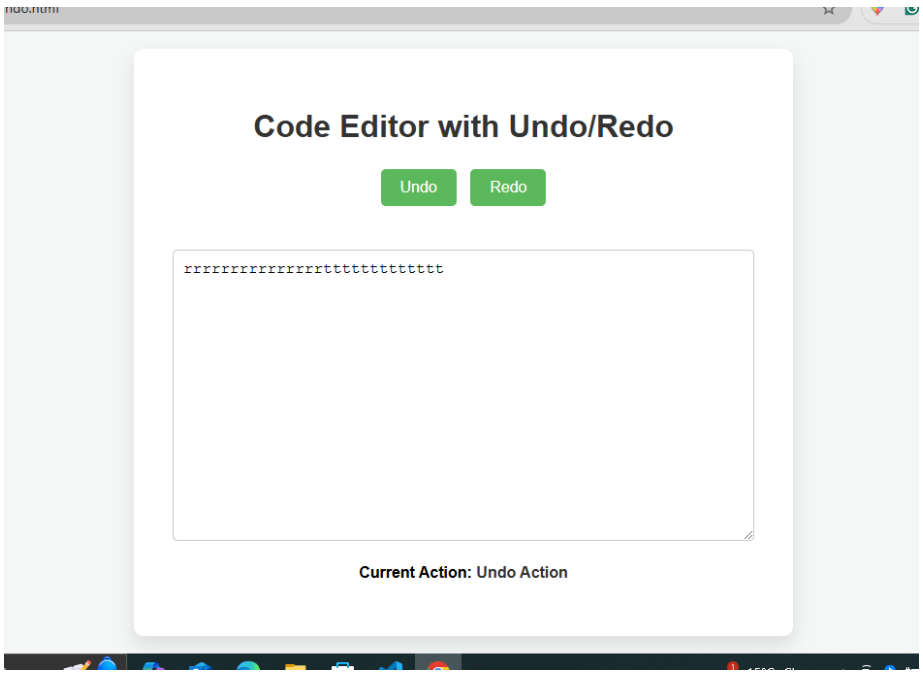
```
.editor-container {
    background-color: white;
    padding: 40px;
    border-radius: 10px;
    box-shadow: 0 10px 20px rgba(0, 0, 0, 0.1);
    width: 600px;
    text-align: center;
}

h1 {
    color: #333;
    margin-bottom: 20px;
}

.controls {
    margin-bottom: 20px;
}

.btn {
    padding: 10px 20px;
    font-size: 16px;
    background-color: #5cb85c;
    color: white;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    margin: 5px;
    transition: background-color 0.3s;
}

.btn:hover {
    background-color: #4cae4c;
}

.editor {
    width: 100%;
    height: 300px;
    font-family: 'Courier New', Courier, monospace;
    font-size: 16px;
    padding: 10px;
    border: 1px solid #ccc;
    border-radius: 5px;
    box-sizing: border-box;
    margin-top: 20px;
}

.status {
    margin-top: 20px;
}

#action-status {
    font-weight: bold;
```

```
    color: #333;
}
```

JAVASCRIPT:

```
let undoStack = [];
let redoStack = [];

const editor = document.getElementById('code-editor');
const actionStatus = document.getElementById('action-status');
 function trackChanges() {
    undoStack.push(editor.value);
        redoStack = [];
      actionStatus.textContent = "Text Edited";
      document.getElementById('undo-btn').disabled = undoStack.length === 0;

    // Enable the redo button if there are actions in the redo stack
    document.getElementById('redo-btn').disabled = redoStack.length === 0;
}
function undo() {
    if (undoStack.length > 0) {
      redoStack.push(editor.value);
        editor.value = undoStack.pop();
            actionStatus.textContent = "Undo Action";
       document.getElementById('undo-btn').disabled = undoStack.length === 0;
          document.getElementById('redo-btn').disabled = redoStack.length === 0;
    }
}
 function redo() {
    if (redoStack.length > 0) {
       // Save the current content to undo stack before redo
       undoStack.push(editor.value);

       // Pop from redo stack and set the editor value to the last action
       editor.value = redoStack.pop();

       // Update the action status
       actionStatus.textContent = "Redo Action";

       // Disable redo button if the redo stack is empty
       document.getElementById('redo-btn').disabled = redoStack.length === 0;

       // Enable undo button since we've redone an action
```

```
        document.getElementById('undo-
btn').disabled = undoStack.length === 0;
    }
}
```



4.Job Queue System: Simulate a job processing system where jobs (like printing documents) are queued. Implement the queue with the ability to dynamically prioritize certain jobs (e.g ., emergency print requests) using a priority queue.

HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Job Queue System</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div class="job-queue-system">
        <h1>Job Queue System</h1>

        <!-- Form to Add a New Job -->
        <div class="form-section">
            <input type="text" id="job-name" class="input-field" placeholder="Job Name">
            <select id="job-priority" class="input-field">
                <option value="emergency">Emergency</option>
                <option value="high">High</option>
                <option value="normal">Normal</option>
            </select>
            <button id="add-job-btn" class="btn" onclick="addJob()">Add Job</button>
        </div>

        <!-- Section for Queued Jobs -->
        <div class="queued-jobs">
            <h3>Queued Jobs</h3>
            <ul id="queue-list">
                <!-- Jobs will appear here dynamically -->
            </ul>
        </div>

        <!-- Process Job Button -->
        <button id="process-job-btn" class="btn" onclick="processJob()">Process Job</button>

        <!-- Processed Jobs Section -->
        <div class="processed-jobs">
            <h3>Processed Jobs</h3>
            <ul id="processed-list">
                <!-- Processed jobs will appear here -->
```

```
        </ul>
      </div>
  </div>

  <script src="script.js"></script>
</body>
</html>
```

CSS:

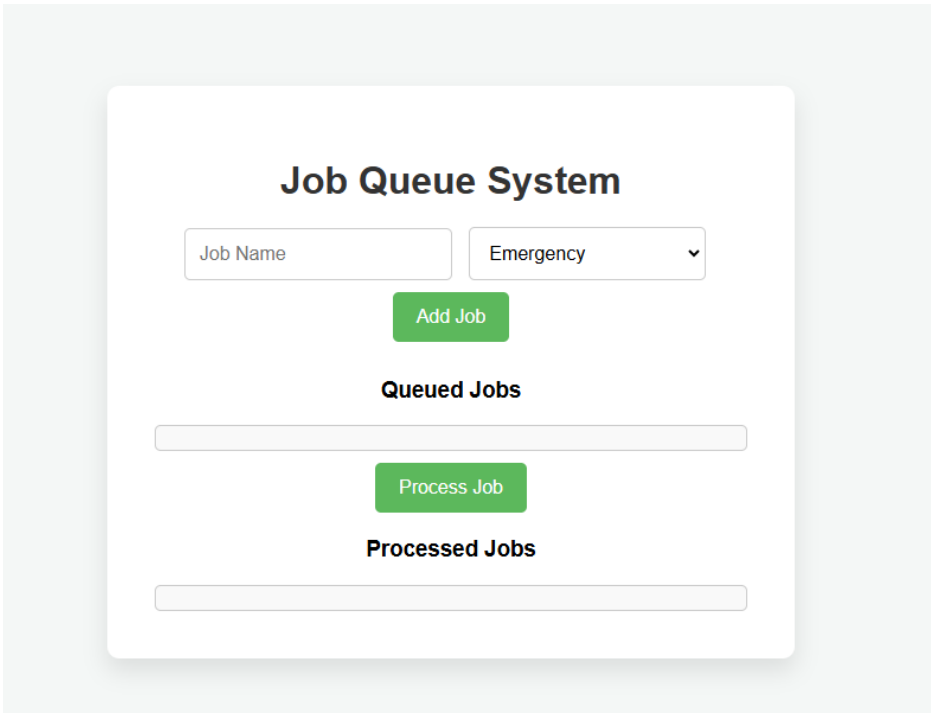
```
body {
  font-family: Arial, sans-serif;
  background-color: #f4f7f6;
  margin: 0;
  padding: 0;
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100vh;
}

.job-queue-system {
  background-color: white;
  padding: 40px;
  border-radius: 10px;
  box-shadow: 0 10px 20px rgba(0, 0, 0, 0.1);
  width: 500px;
  text-align: center;
}

h1 {
  color: #333;
  margin-bottom: 20px;
}

.form-section {
  margin-bottom: 30px;
}

.input-field {
  padding: 12px;
  font-size: 16px;
  border: 1px solid #ccc;
  border-radius: 5px;
  margin-right: 10px;
  width: 200px;
}

select {
  padding: 12px;
  font-size: 16px;
  border: 1px solid #ccc;
  border-radius: 5px;
  width: 200px;
```

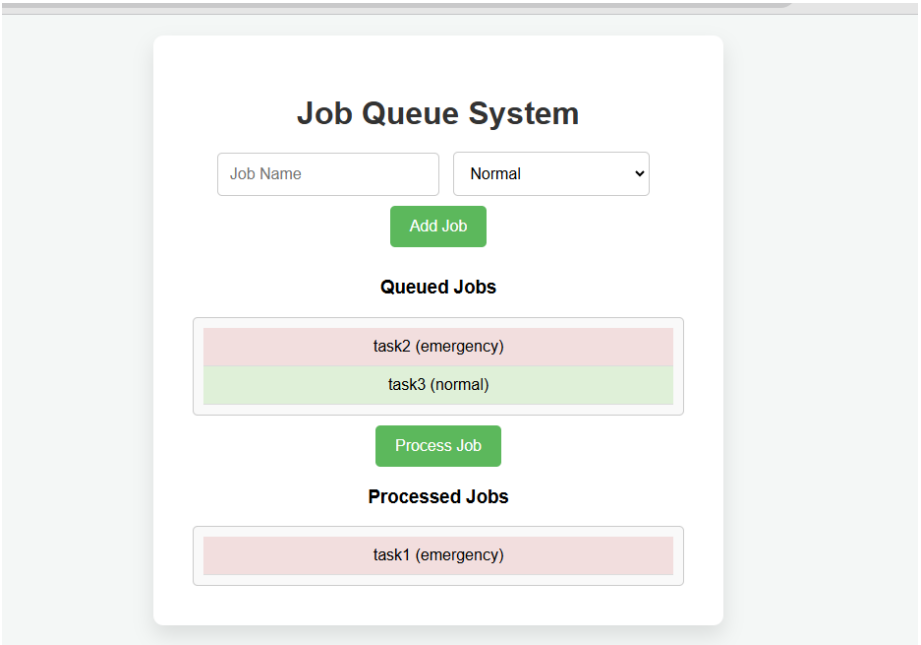
```
}

.btn {
    padding: 12px 20px;
    font-size: 16px;
    background-color: #5cb85c;
    color: white;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    margin-top: 10px;
}

.btn:hover {
    background-color: #4cae4c;
}

.queued-jobs, .processed-jobs {
    margin-top: 20px;
}

ul {
    list-style-type: none;
    padding: 0;
    margin: 0;
    max-height: 200px;
    overflow-y: auto;
    border: 1px solid #ccc;
    border-radius: 5px;
    padding: 10px;
    background-color: #f9f9f9;
}

li {
    padding: 10px;
    border-bottom: 1px solid #ddd;
}

li.emergency {
    background-color: #f2dede;
}

li.high {
    background-color: #fcf8e3;
}

li.normal {
    background-color: #dff0d8;
}

button:disabled {
    background-color: #ccc;
    cursor: not-allowed;
}
```

JAVASCRIPT:

```
// Priority Queue Implementation
class PriorityQueue {
    constructor() {
        this.queue = [];
    }

    // Add job to the queue with priority
    enqueue(job) {
        let added = false;
        for (let i = 0; i < this.queue.length; i++) {
            if (this.comparePriority(job.priority, this.queue[i].priority)) {
                this.queue.splice(i, 0, job);
                added = true;
                break;
            }
        }
        if (!added) {
            this.queue.push(job);
        }
    }

    // Compare priorities (emergency > high > normal)
    comparePriority(priority1, priority2) {
        const priorities = ['normal', 'high', 'emergency'];
        return priorities.indexOf(priority1) > priorities.indexOf(priority2);
    }

    // Remove and return the highest priority job
    dequeue() {
        return this.queue.shift();
    }

    // Check if the queue is empty
    isEmpty() {
        return this.queue.length === 0;
    }
}

// Global Variables
const queue = new PriorityQueue();
const queueList = document.getElementById('queue-list');
const processedList = document.getElementById('processed-list');
const jobNameInput = document.getElementById('job-name');
const jobPriorityInput = document.getElementById('job-priority');
```

```
// Add Job to the Queue
function addJob() {
    const jobName = jobNameInput.value.trim();
    const jobPriority = jobPriorityInput.value;

    if (jobName === '') {
        alert('Please enter a job name!');
        return;
    }

    const job = { name: jobName, priority: jobPriority };
    queue.enqueue(job);
    displayQueue();
    jobNameInput.value = '';  // Clear input field
}

// Process Job (Highest Priority First)
function processJob() {
    if (queue.isEmpty()) {
        alert('No jobs to process!');
        return;
    }

    const job = queue.dequeue();
    displayProcessedJob(job);
    displayQueue();
}

// Display the Queue
function displayQueue() {
    queueList.innerHTML = '';
    queue.queue.forEach(job => {
        const li = document.createElement('li');
        li.textContent = `${job.name} (${job.priority})`;
        li.classList.add(job.priority);
        queueList.appendChild(li);
    });
}

// Display Processed Job
function displayProcessedJob(job) {
    const li = document.createElement('li');
    li.textContent = `${job.name} (${job.priority})`;
    li.classList.add(job.priority);
    processedList.appendChild(li);
}
```


Job Queue System
Job Name
Emergency
Add Job
Queued Jobs
Process Job
Processed Jobs


Job Queue System
Job Name
Normal
Add Job
Queued Jobs
task2 (emergency)
task3 (normal)
Process Job
Processed Jobs
task1 (emergency)

5.0: Stock Span Problem: Solve the Stock Span Problem using a stack, where for each day's stock price, you calculate the number of consecutive days the price was less than or equal to today's price.

HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Stock Span Problem</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div class="container">
        <h1>Stock Span Problem</h1>

        <!-- Input field for stock prices -->
        <input type="text" id="stock-prices" class="input-field" placeholder="Enter stock prices separated by commas" />
        <button id="calculate-btn" class="btn" onclick="calculateStockSpan()">Calculate Stock Span</button>

        <!-- Output Section -->
        <div id="output" class="output">
            <h3>Stock Span:</h3>
            <ul id="stock-span-list"></ul>
        </div>
    </div>

    <script src="script.js"></script>
</body>
</html>
```

CSS:

```
body {
    font-family: Arial, sans-serif;
    background-color: #f4f7f6;
    margin: 0;
    padding: 0;
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
}

.container {
    background-color: white;
    padding: 40px;
    border-radius: 10px;
```

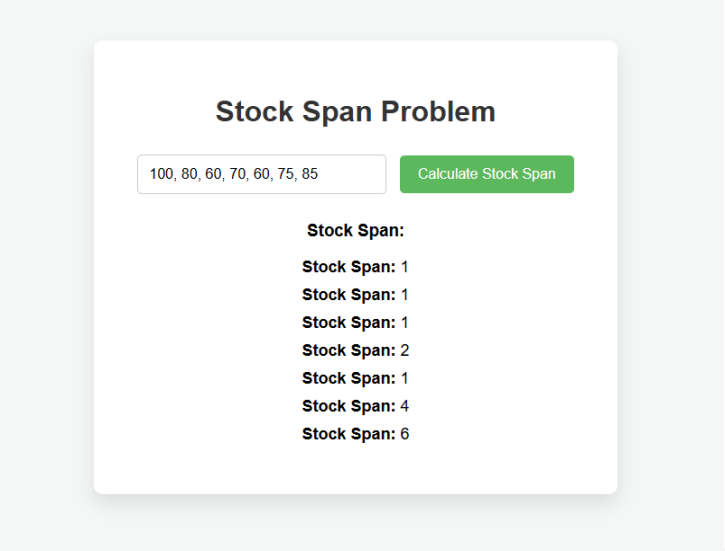
```
    box-shadow: 0 10px 20px rgba(0, 0, 0, 0.1);
    width: 500px;
    text-align: center;
}

h1 {
    color: #333;
    margin-bottom: 20px;
}

.input-field {
    padding: 12px;
    font-size: 16px;
    border: 1px solid #ccc;
    border-radius: 5px;
    margin-right: 10px;
    width: 250px;
}

.btn {
    padding: 12px 20px;
    font-size: 16px;
    background-color: #5cb85c;
    color: white;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    margin-top: 10px;
}

.btn:hover {
    background-color: #4cae4c;
}

.output {
    margin-top: 30px;
}

ul {
    list-style-type: none;
    padding: 0;
}

li {
    font-size: 18px;
    margin: 10px 0;
}

li span {
    font-weight: bold;
}

button:disabled {
```

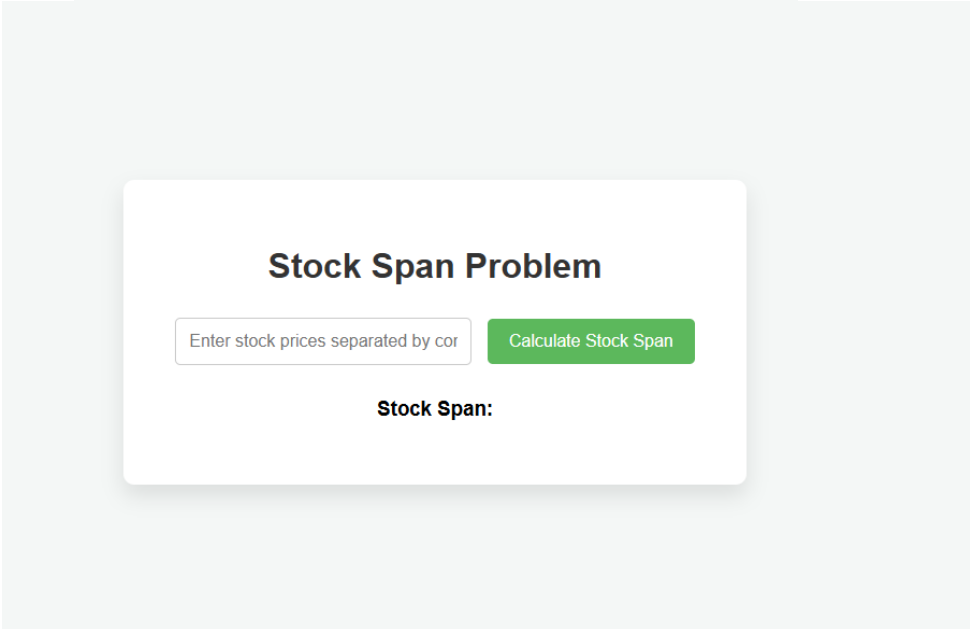
```
    background-color: #ccc;
    cursor: not-allowed;
}
```

JavaScript:

```
// Function to calculate Stock Span using a stack
function calculateStockSpan() {
    // Get the input stock prices
    let pricesInput = document.getElementById('stock-prices').value;
    let prices = pricesInput.split(',').map(price => parseInt(price.trim()));

    // Validate input
    if (prices.some(isNaN) || prices.length === 0) {
        alert('Please enter valid stock prices separated by commas.');
        return;
    }

    // Array to store the stock span
    let stockSpan = [];
    let stack = [];

    // Loop over each stock price
    for (let i = 0; i < prices.length; i++) {
        // Pop from stack while the current price is greater than or equal to the price at stack's top
        while (stack.length > 0 && prices[stack[stack.length - 1]] <= prices[i]) {
            stack.pop();
        }

        // Calculate span
        let span = (stack.length === 0) ? (i + 1) : (i - stack[stack.length - 1]);
        stockSpan.push(span);

        // Push the current index onto the stack
        stack.push(i);
    }

    // Display the stock span result
    displayStockSpan(stockSpan);
}
// Function to display the Stock Span results
function displayStockSpan(stockSpan) {
    const output = document.getElementById('stock-span-list');
    output.innerHTML = '';

    stockSpan.forEach(span => {
        let li = document.createElement('li');
```

```
        li.innerHTML = `<span>Stock
Span:</span> ${span}`;
        output.appendChild(li);
    });
}
```





6. Bank ATM Queue Simulation: Implement a bank ATM queue where customers are queued for transactions. Simulate different types of transactions (deposit, withdrawal, balance check) with varying processing times. Use a deque (double-ended queue) to allow priority transactions at either end

HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Bank ATM Queue Simulation</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div class="container">
        <h1>Bank ATM Queue Simulation</h1>


        <!-- Form to Add Customer to Queue -->
        <div class="form-section">
            <input type="text" id="customer-name" class="input-field" placeholder="Customer Name">
            <select id="transaction-type" class="input-field">
                <option value="deposit">Deposit</option>
                <option value="withdrawal">Withdrawal</option>
                <option value="balance-check">Balance Check</option>
            </select>
            <button id="add-to-queue" class="btn" onclick="addCustomerToQueue()">Add to Queue</button>
            <button id="add-priority" class="btn" onclick="addPriorityCustomer()">Add Priority Customer</button>
        </div>


        <!-- Queue Section -->
        <div id="queue-section">
            <h3>Queue</h3>
            <ul id="queue-list">
                <!-- Queue will be populated here -->
            </ul>
        </div>


        <!-- Process Queue Section -->
        <div>
```

```
        <button id="process-next" class="btn" onclick="processNextCustomer()">Process Next Customer</button>
    </div>

    <!-- Processed Transactions Section -->
    <div id="processed-section">
        <h3>Processed Transactions</h3>
        <ul id="processed-list">
            <!-- Processed transactions will be displayed here -->
        </ul>
    </div>
  </div>

  <script src="script.js"></script>
</body>
</html>
```

CSS:

```
body {
  font-family: Arial, sans-serif;
  background-color: #f4f7f6;
  margin: 0;
  padding: 0;
  display: flex;
  justify-content: center;
  align-items: center;
  height: 100vh;
}

.container {
  background-color: white;
  padding: 40px;
  border-radius: 10px;
  box-shadow: 0 10px 20px rgba(0, 0, 0, 0.1);
  width: 500px;
  text-align: center;
}

h1 {
  color: #333;
  margin-bottom: 20px;
}

.form-section {
  margin-bottom: 30px;
}

.input-field {
  padding: 12px;
  font-size: 16px;
  border: 1px solid #ccc;
```

```
    border-radius: 5px;
    margin-right: 10px;
    width: 200px;
}

select {
    padding: 12px;
    font-size: 16px;
    border: 1px solid #ccc;
    border-radius: 5px;
    width: 200px;
}

.btn {
    padding: 12px 20px;
    font-size: 16px;
    background-color: #5cb85c;
    color: white;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    margin-top: 10px;
}

.btn:hover {
    background-color: #4cae4c;
}

#queue-section, #processed-section {
    margin-top: 20px;
}

ul {
    list-style-type: none;
    padding: 0;
}

li {
    padding: 10px;
    border-bottom: 1px solid #ddd;
}

li.deposit {
    background-color: #dff0d8;
}

li.withdrawal {
    background-color: #fcf8e3;
}

li.balance-check {
    background-color: #f2dede;
}
```

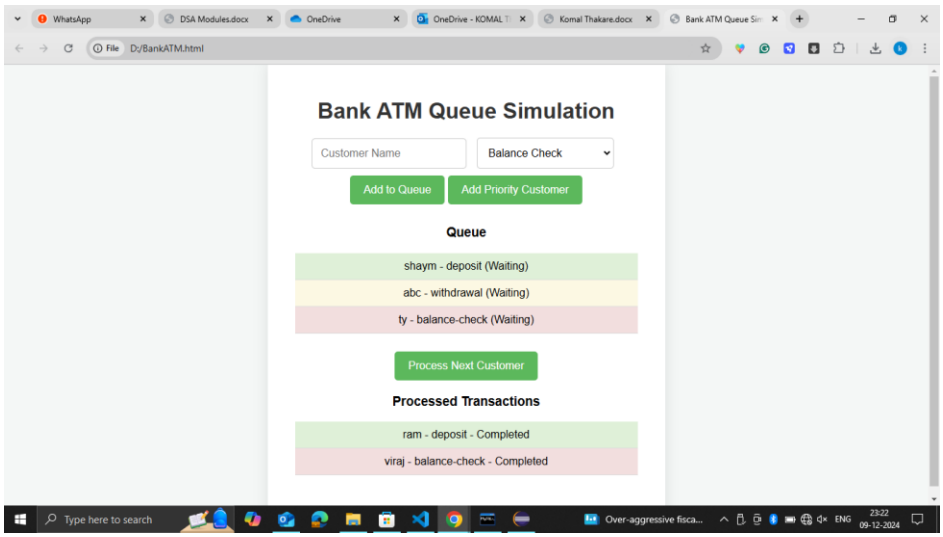
```
button:disabled {
    background-color: #ccc;
    cursor: not-allowed;
}
```

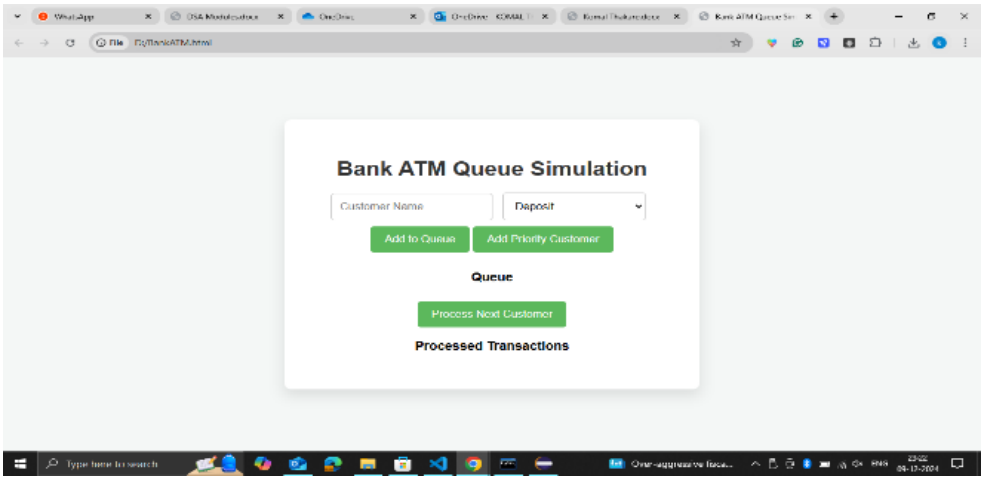
JAVASCRPIT:

```
// Customer queue (using a deque implemented with an array)
let queue = [];
let processedTransactions = [];

// Function to add a customer to the queue (normal)
function addCustomerToQueue() {
    let name = document.getElementById("customer-name").value.trim();
    let transactionType = document.getElementById("transaction-type").value;

    if (name === "") {
        alert("Please enter a customer name.");
        return;
    }

    let customer = { name, transactionType, status: "Waiting" };
    queue.push(customer);
    displayQueue();
    document.getElementById("customer-name").value = ""; // Clear input field
}

// Function to add a priority customer to the front of the queue
function addPriorityCustomer() {
    let name = document.getElementById("customer-name").value.trim();
    let transactionType = document.getElementById("transaction-type").value;

    if (name === "") {
        alert("Please enter a customer name.");
        return;
    }

    let customer = { name, transactionType, status: "Waiting" };
    queue.unshift(customer);  // Adds the customer to the front of the queue
    displayQueue();
    document.getElementById("customer-name").value = ""; // Clear input field
}
```

```
// Function to process the next customer in the queue
function processNextCustomer() {
    if (queue.length === 0) {
        alert("No customers in the queue.");
        return;
    }

    // Pop the first customer in the queue
    let customer = queue.shift();
    customer.status = "Processing...";

    // Simulate processing time based on transaction type
    let processingTime = 0;
    if (customer.transactionType === "deposit") {
        processingTime = 3000; // 3 seconds for deposit
    } else if (customer.transactionType === "withdrawal") {
        processingTime = 5000; // 5 seconds for withdrawal
    } else if (customer.transactionType === "balance-check") {
        processingTime = 2000; // 2 seconds for balance check
    }

    // Simulate the processing time
    setTimeout(() => {
        customer.status = "Completed";
        processedTransactions.push(customer);
        displayProcessedTransactions();
        displayQueue();
    }, processingTime);

    displayQueue();
}

// Function to display the queue of customers
function displayQueue() {
    const queueList = document.getElementById("queue-list");
    queueList.innerHTML = "";

    queue.forEach((customer) => {
        const li = document.createElement("li");
        li.classList.add(customer.transactionType);
        li.textContent = `${customer.name} - ${customer.transactionType} (${customer.status})`;
        queueList.appendChild(li);
    });
}
```

```
// Function to display processed transactions

function displayProcessedTransactions() {

    const processedList =
document.getElementById("processed-
list");

    processedList.innerHTML =
"";processedTransactions.forEach((transacti
on) => {

        const li =
document.createElement("li");


li.classList.add(transaction.transactionType)
;

        li.textContent = `${transaction.name} -
${transaction.transactionType} -
${transaction.status}`;

        processedList.appendChild(li);

    });}
```




Bank ATM Queue Simulation
Customer Name
Deposit
Add to Queue
Add Priority Customer
Queue
Process Next Customer
Processed Transactions

MODULE 4

1: Organizational Hierarchy Management System: Implement an organization's hierarchy using a tree structure where each node represents an employee. Simulate promotions, new hires, and removals dynamically, ensuring the tree stays balanced.

HTML :

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Enhanced Organizational Hierarchy</title>
  <link rel="stylesheet" href="styles.css">
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0-beta3/css/all.min.css">
</head>
<body>
  <div class="container">
    <h1>Enhanced Organizational Hierarchy</h1>
    <div class="controls">
      <input type="text" id="employeeName" placeholder="Employee Name">
      <input type="text" id="supervisorName" placeholder="Supervisor Name">
      <input type="text" id="employeeDetails" placeholder="Employee Details (Optional)">
      <div class="buttons">
        <button id="addEmployee"><i class="fas fa-user-plus"></i> Add Employee</button>
        <button id="removeEmployee"><i class="fas fa-user-minus"></i> Remove Employee</button>
        <button id="promoteEmployee"><i class="fas fa-arrow-up"></i> Promote Employee</button>
        <button id="viewDetails"><i class="fas fa-info-circle"></i> View Details</button>
      </div>
    </div>
    <div id="hierarchyTree" class="tree-container"></div>
  </div>
  <script src="script.js"></script>
</body>
</html>
```

CSS:

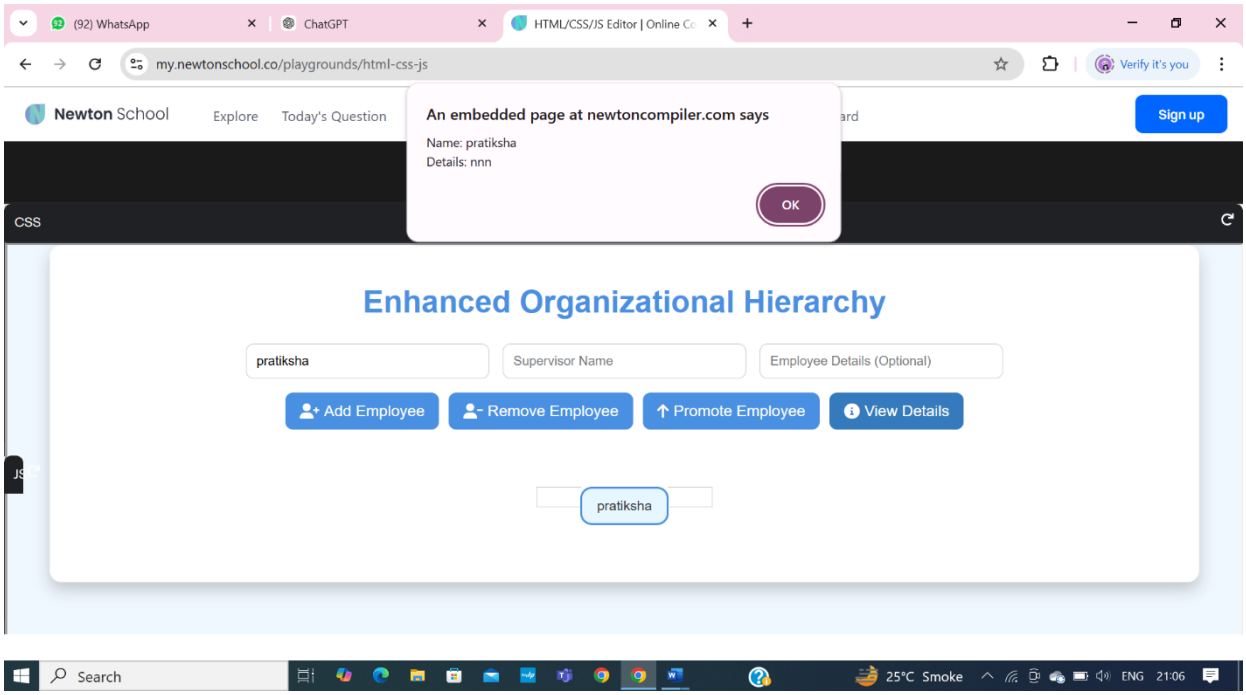
```
/* styles.css */
body {
  font-family: 'Roboto', sans-serif;
  margin: 0;
  padding: 0;
  background-color: #f0f8ff;
}

.container {
  width: 90%;
  margin: 0 auto;
  padding: 20px;
  text-align: center;
  background: white;
  border-radius: 12px;
  box-shadow: 0 8px 16px rgba(0, 0, 0, 0.2);
}

h1 {
  color: #4a90e2;
  margin-bottom: 20px;
}

.controls {
  position: absolute;
  top: 0;
  border: 1px solid #ccc;
  width: 50%;
  height: 20px;
}


.tree-node:before {
  left: -50%;
  border-right: none;
}

.tree-node:after {
  right: -50%;
  border-left: none;
}

.tree-node .employee {
  background: #e6f7ff;
  border: 2px solid #4a90e2;
  color: #333;
  padding: 10px 15px;
  border-radius: 12px;
  font-size: 14px;
  position: relative;
```

```
    margin-bottom: 20px;
    position: relative;
    margin: 20px auto;
    display: inline-block;
    text-align: center;
}

.tree-node:before,
.tree-node:after {
    content: '';
    transition: transform 0.3s ease, background-color
0.3s ease;
}

.tree-node .employee:hover {
    transform: scale(1.1);
    background-color: #cce7ff;
    cursor: pointer;
}
```

JAVASCRIPT:

```
class EmployeeNode {
    constructor(name, details = "No details provided")
{
        this.name = name;
        this.details = details;
        this.subordinates = [];
    }
}

class OrgHierarchy {
    constructor() {
        this.root = null;
    }

    addEmployee(name, supervisorName = null,
details = "") {
        const newEmployee = new
EmployeeNode(name, details);
        if (!this.root) {
            this.root = newEmployee;
            return true;
        }

        const supervisor =
this.findEmployee(supervisorName, this.root);
        if (supervisor) {
            supervisor.subordinates.push(newEmployee);
            return true;
        }
        return false;
    }

    removeEmployee(name) {
        if (!this.root) return false;

        if (this.root.name === name) {
            this.root = null;
            return true;
        }

        return this._removeEmployee(this.root, name);
    }
data-name="${node.name}">${node.name}</div>`;
        container.appendChild(element);

        const subContainer =
document.createElement("div");
        subContainer.style.display = "flex";
        subContainer.style.justifyContent = "center";

        node.subordinates.forEach(sub => {
            this.displayHierarchy(sub, subContainer);
        });
    element.appendChild(subContainer);
    }
}

// Instantiate the hierarchy
const orgHierarchy = new OrgHierarchy();
const treeContainer =
document.getElementById("hierarchyTree");

// UI Actions
function updateTree() {
    treeContainer.innerHTML = "";
    orgHierarchy.displayHierarchy(orgHierarchy.root,
treeContainer);
}

document.getElementById("addEmployee").addEventLi
stener("click", () => {
    const name =
document.getElementById("employeeName").value.tri
m();
    const supervisor =
document.getElementById("supervisorName").value.tri
m();
    const details =
document.getElementById("employeeDetails").value.tri
m();
    if (name) {
```

```
  _removeEmployee(current, name) {
    for (let i = 0; i < current.subordinates.length;
i++) {
      if (current.subordinates[i].name === name) {
        current.subordinates.splice(i, 1);
        return true;
      }
      if
(this._removeEmployee(current.subordinates[i],
name)) {
        return true;
      }
    }
    return false;
  }

  promoteEmployee(name) {
    const employee = this.findEmployee(name,
this.root);
    if (employee && this.root !== employee) {
      this.removeEmployee(name);
      this.root.subordinates.push(employee);
      return true;
    }
    return false;
  }

  findEmployee(name, node) {
    if (!node) return null;
    if (node.name === name) return node;
for (const sub of node.subordinates) {
      const found = this.findEmployee(name, sub);
      if (found) return found;
    }
    return null;
  }

  displayHierarchy(node, container) {
    if (!node) return;

const element = document.createElement("div");
    element.className = "tree-node";
    element.innerHTML = `<div class="employee"
    orgHierarchy.addEmployee(name, supervisor ||
null, details || "No details provided");
    updateTree();
  }
});

document.getElementById("removeEmployee").addEve
ntListener("click", () => {
  const name =
document.getElementById("employeeName").value.tri
m();
  if (name) {
    orgHierarchy.removeEmployee(name);
    updateTree();
  }
});

document.getElementById("promoteEmployee").addEv
entListener("click", () => {
  const name =
document.getElementById("employeeName").value.tri
m();
  if (name) {
    orgHierarchy.promoteEmployee(name);
    updateTree();
  }
});

document.getElementById("viewDetails").addEventList
ener("click", () => {
  const name =
document.getElementById("employeeName").value.tri
m();
  const employee = orgHierarchy.findEmployee(name,
orgHierarchy.root);
  if (employee) {
    alert(`Name: ${employee.name}\nDetails:
${employee.details}`);
  } else {
    alert("Employee not found!");
  }
});

// Initial Render
updateTree();
```



2: E-Commerce Recommendation System (Binary Search 'Tree): Build an e-commerce recommendation system where products are stored in a binary search tree (BST) based on customer ratings. Implement operations to find products within specific rating range and suggest similar products.

HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>E-Commerce Recommendation System</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <div class="container">
    <h1>E-Commerce Recommendation System</h1>
    <div class="controls">
      <input type="text" id="productName" placeholder="Product Name">
      <input type="number" id="productRating" placeholder="Product Rating (1-5)" min="1" max="5">
      <button id="addProduct">Add Product</button>
      <hr>
      <input type="number" id="ratingMin" placeholder="Min Rating (1-5)" min="1" max="5">
      <input type="number" id="ratingMax" placeholder="Max Rating (1-5)" min="1" max="5">
      <button id="findProducts">Find Products</button>
      <hr>
      <input type="number" id="similarRating" placeholder="Find Similar to Rating" min="1" max="5">
      <button id="suggestProducts">Suggest Similar Products</button>
    </div>
    <div id="output" class="output"></div>
  </div>
  <script src="script.js"></script>
</body>
</html>
```

CSS:

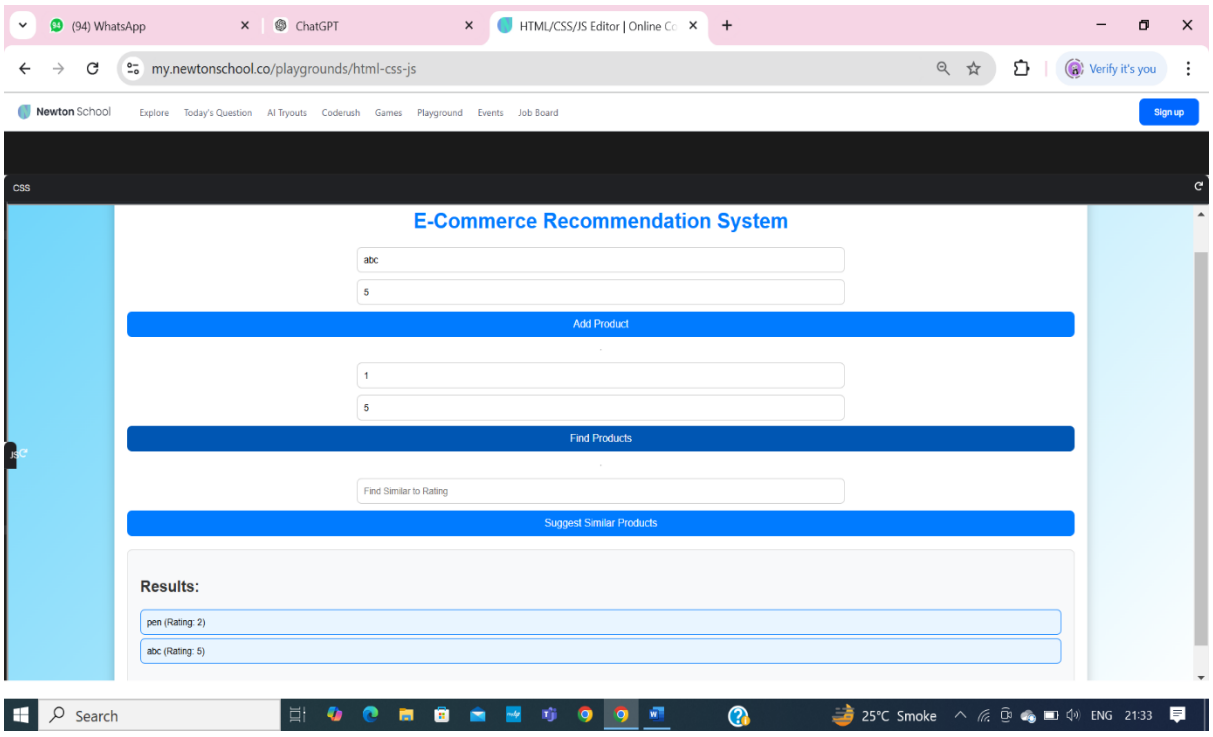
```
body {
    font-family: 'Arial', sans-serif;
    margin: 0;
    padding: 0;
    background: linear-gradient(135deg, #6dd5fa, #ffffff);
}

.container {
    width: 80%;
    margin: 20px auto;
    padding: 20px;
    background: #ffffff;
    border-radius: 12px;
    box-shadow: 0 8px 16px rgba(0, 0, 0, 0.2);
    text-align: center;
}

h1 {
    color: #007BFF;
    font-size: 2rem;
    margin-bottom: 20px;
}

.controls {
    margin-bottom: 20px;
    display: flex;
    flex-direction: column;
    gap: 10px;
}

.controls input {
    padding: 10px;
    width: 50%;
    margin: 0 auto;
    border: 1px solid #ccc;
    border-radius: 8px;
    outline: none;
    transition: box-shadow 0.3s ease-in-out;
    font-size: 14px;
}

.controls input:focus {
    box-shadow: 0 0 8px rgba(0, 123, 255, 0.5);
}

.controls button {
    padding: 10px 20px;
    background: #007BFF;
    color: #fff;
    border: none;
    border-radius: 8px;
    cursor: pointer;
    font-size: 16px;
    transition: background-color 0.3s ease, transform
0.2s ease;
}

.controls button:hover {
    background-color: #0056b3;
    transform: translateY(-2px);
}

.output {
    margin-top: 20px;
    padding: 20px;
    background: #f8f9fa;
    border: 1px solid #ddd;
    border-radius: 8px;
    text-align: left;
    max-height: 400px;
    overflow-y: auto;
}

.output h2 {
    font-size: 1.5rem;
    color: #333;
}

.output ul {
    list-style: none;
    padding: 0;
}

.output li {
    background: #e9f5ff;
    border: 1px solid #007BFF;
    padding: 10px;
    margin: 5px 0;
    border-radius: 8px;
    font-size: 14px;
}
suggestSimilar(rating, node = this.root, result = []) {
        if (node === null) return result;

        if (Math.abs(node.rating - rating) <= 1) {
            result.push(`${node.name} (Rating:
${node.rating})`);
        }

        this.suggestSimilar(rating, node.left, result);
        this.suggestSimilar(rating, node.right, result);

        return result;
    }
}
```

JAVASCRIPT:

```
class Product {
  constructor(name, rating) {
    this.name = name;
    this.rating = rating;
    this.left = null;
    this.right = null;
  }
}

class BST {
  constructor() {
    this.root = null;
  }

  addProduct(name, rating) {
    const newNode = new Product(name, rating);

    if (this.root === null) {
      this.root = newNode;
      return true;
    }

    let current = this.root;
    while (true) {
      if (rating < current.rating) {
        if (current.left === null) {
          current.left = newNode;
          return true;
        }
        current = current.left;
      } else {
        if (current.right === null) {
          current.right = newNode;
          return true;
        }
        current = current.right;
      }
    }
  }

  findProductsInRange(min, max, node = this.root, result = []) {
    if (node === null) return result;

    if (node.rating >= min && node.rating <= max) {
      result.push(`${node.name} (Rating: ${node.rating})`);
    }

    if (min < node.rating) this.findProductsInRange(min, max,
node.left, result);
    if (max > node.rating) this.findProductsInRange(min, max,
node.right, result);

function displayError(message) {
  const output = document.getElementById("output");
  output.innerHTML = `<h2 style="color:
red;">Error:</h2><p>${message}</p>`;
}
// Add event listeners
document.getElementById("addProduct").addEventListen
er("click", () => {
  const name =
document.getElementById("productName").value.trim();
  const rating =
parseInt(document.getElementById("productRating").valu
e);

  if (name && rating >= 1 && rating <= 5) {
    bst.addProduct(name, rating);
    updateOutput([`Product "${name}" added
successfully!`]);
  } else {
    displayError("Please provide a valid product name
and rating (1-5).");
  }
});

document.getElementById("findProducts").addEventListe
ner("click", () => {
  const min =
parseInt(document.getElementById("ratingMin").value);
  const max =
parseInt(document.getElementById("ratingMax").value);

  if (min >= 1 && max <= 5 && min <= max) {
    const products = bst.findProductsInRange(min, max);
    if (products.length) {
      updateOutput(products);
    } else {
      updateOutput(["No products found in the specified
range."]);
    }
  } else {
    displayError("Please provide a valid rating range (1-
5).");
  }
});

document.getElementById("suggestProducts").addEventLi
stener("click", () => {
  const rating =
parseInt(document.getElementById("similarRating").value
);
```

```
        return result;
    }
// Instantiate the BST
const bst = new BST();

// Helper functions
function updateOutput(message) {
    const output = document.getElementById("output");
    output.innerHTML =
`<h2>Results:</h2><ul>${message.map(item =>
`<li>${item}</li>`).join("")}</ul>`;
}
```

```
    if (rating >= 1 && rating <= 5) {
        const suggestions = bst.suggestSimilar(rating);
        if (suggestions.length) {
            updateOutput(suggestions);
        } else {
            updateOutput(["No similar products found."]);
        }
    } else {
        displayError("Please provide a valid rating (1-5).");
    }
});
```



3: Social Network Friend Recommendation (Graph): Use a graph to represent connections between users in a social network. Implement a BES algorithm to suggest friend recommendations based on |mutual connections.

HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport"
content="width=device-width, initial-scale=1.0">
  <title>Social Network Friend
Recommendation</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <div class="container">
    <h1>Social Network Friend
Recommendation</h1>
    <div class="controls">
      <input type="text" id="userName"
placeholder="Enter User Name">
      <input type="text" id="friendName" placeholder="Enter Friend
Name">
      <button id="addConnection">Add Connection</button>
      <hr>
      <input type="text" id="recommendUser" placeholder="Enter User
to Recommend Friends For">
      <button id="recommendFriends">Recommend Friends</button>
    </div>
    <div id="output" class="output"></div>
  </div>
  <script src="script.js"></script>
</body>
</html>
```

CSS:

```
body {
  font-family: 'Arial', sans-serif;
  margin: 0;
  padding: 0;
  background: linear-gradient(135deg, #ff9a9e,
#fad0c4);
}

.container {
  width: 80%;
  margin: 20px auto;
  padding: 20px;
  background: #ffffff;
  border-radius: 12px;
  box-shadow: 0 8px 16px rgba(0, 0, 0, 0.2);
  text-align: center;
}

h1 {
  color: #ff6f61;
  font-size: 2rem;
  margin-bottom: 20px;
}

.controls {
  margin-bottom: 20px;
  display: flex;
  flex-direction: column;
  gap: 10px;
}

.controls input:focus {
  box-shadow: 0 0 8px rgba(255, 111, 97, 0.5);
}

.controls button {
  padding: 10px 20px;
  background: #ff6f61;
  color: #fff;
  border: none;
  border-radius: 8px;
  cursor: pointer;
  font-size: 16px;
  transition: background-color 0.3s ease, transform 0.2s ease;
}

.controls button:hover {
  background-color: #e55a4d;
  transform: translateY(-2px);
}

.output {
  margin-top: 20px;
  padding: 20px;
  background: #f8f9fa;
  border: 1px solid #ddd;
  border-radius: 8px;
  text-align: left;
  max-height: 400px;
  overflow-y: auto;
}

.output h2 {
  font-size: 1.5rem;
```

```
}

.controls input {
    padding: 10px;
    width: 50%;
    margin: 0 auto;
    border: 1px solid #ccc;
    border-radius: 8px;
    outline: none;
    transition: box-shadow 0.3s ease-in-out;
    font-size: 14px;
}
    color: #333;
}

.output ul {
    list-style: none;
    padding: 0;
}

.output li {
    background: #ffe1df;
    border: 1px solid #ff6f61;
    padding: 10px;
    margin: 5px 0;
    border-radius: 8px;
    font-size: 14px;
}
```

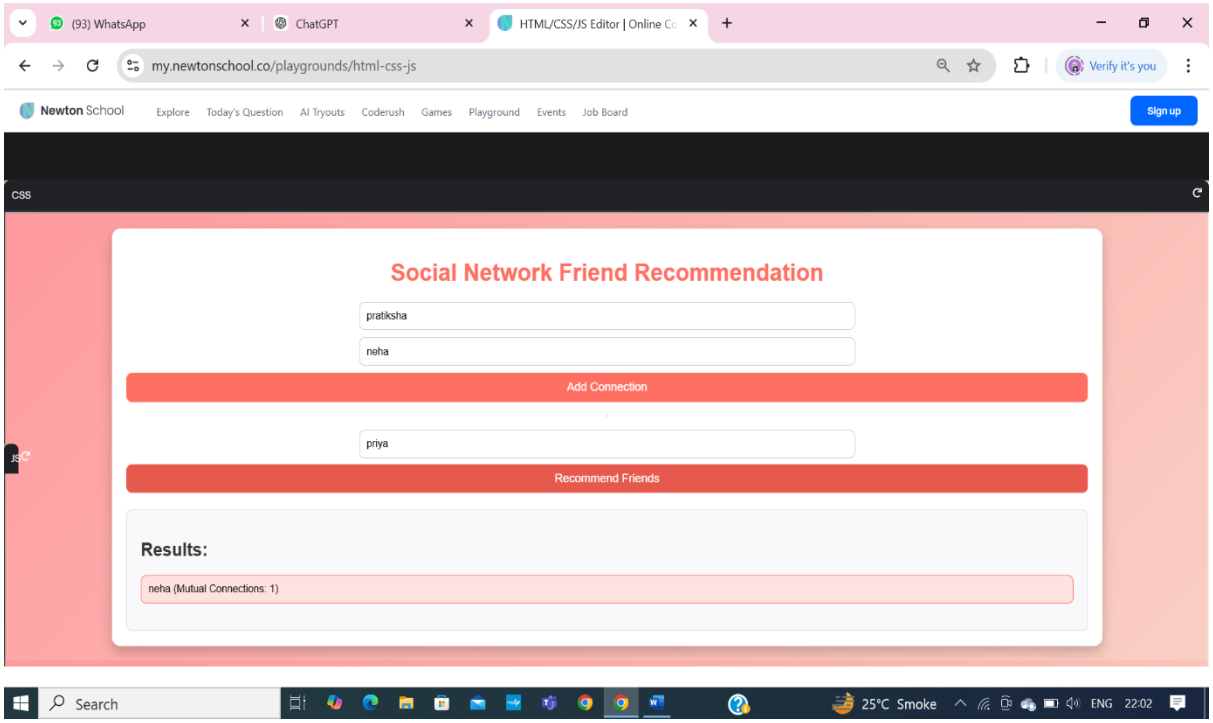
JAVASCRIPT:

```
class SocialNetwork {
    constructor() {
        this.graph = {};
    }

    addConnection(user, friend) {
        if (!this.graph[user]) this.graph[user] = new
Set();
        if (!this.graph[friend]) this.graph[friend] =
new Set();

        this.graph[user].add(friend);
        this.graph[friend].add(user);
    }

    recommendFriends(user) {
        if (!this.graph[user]) return [`User "${user}"
does not exist in the network.`];

        const visited = new Set();
        const recommendations = new Map();

        const queue = [user];
        visited.add(user);

        while (queue.length > 0) {
            const current = queue.shift();

            for (const friend of this.graph[current]) {
                if (!visited.has(friend)) {
                    queue.push(friend);
                    visited.add(friend);

                    for (const mutual of this.graph[friend])
{
```

```
// Instantiate the Social Network
const network = new SocialNetwork();

// Helper functions
function updateOutput(message) {
    const output = document.getElementById("output");
    output.innerHTML = `<h2>Results:</h2><ul>${message.map(item =>
`<li>${item}</li>`).join("")}</ul>`;
}

function displayError(message) {
    const output = document.getElementById("output");
    output.innerHTML = `<h2 style="color:
red;">Error:</h2><p>${message}</p>`;
}


// Add event listeners
document.getElementById("addConnection").addEventListener("click", ()
=> {
    const user = document.getElementById("userName").value.trim();
    const friend = document.getElementById("friendName").value.trim();

    if (user && friend && user !== friend) {
        network.addConnection(user, friend);
        updateOutput([`Connection added between "${user}" and
"${friend}"!`]);
    } else {
        displayError("Please provide valid user and friend names.");
    }
});

document.getElementById("recommendFriends").addEventListener("click",
() => {
    const user = document.getElementById("recommendUser").value.trim();
```

```
                if (mutual !== user &&
!this.graph[user].has(mutual)) {
                    recommendations.set(mutual,
(recommendations.get(mutual) || 0) + 1);
                }
            }
        }
    }

    const sortedRecommendations =
[...recommendations.entries()].sort((a, b) => b[1] -
a[1]);
    return sortedRecommendations.map(([name,
count]) => `${name} (Mutual Connections:
${count})`);
  }
}
```

```
if (user) {
  const recommendations = network.recommendFriends(user);
  if (recommendations.length) {
    updateOutput(recommendations);
  } else {
    updateOutput(["No recommendations found."]);
  }
 } else {
  displayError("Please provide a valid user name.");
 }
});
```



4: Shortest Path in a City (Graph): Given a city represented as a graph with road networks (nodes for Intersections , edges for roads), use Dijkstra's algorithm to find the shortest path between any two Intersections .

HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>City Shortest Path Finder</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <div class="container">
    <h1>City Shortest Path Finder</h1>
    <div class="controls">
      <input type="text" id="intersectionA" placeholder="Start Intersection">
      <input type="text" id="intersectionB" placeholder="End Intersection">
      <input type="number" id="distance" placeholder="Distance Between" min="1">
      <button id="addRoad">Add Road</button>
      <hr>
      <input type="text" id="startPoint" placeholder="Start Intersection">
      <input type="text" id="endPoint" placeholder="End Intersection">
      <button id="findPath">Find Shortest Path</button>
    </div>
    <div id="output" class="output"></div>
  </div>
  <script src="script.js"></script>
</body>
</html>
```

CSS:

```
body {
  font-family: 'Arial', sans-serif;
  margin: 0;
  padding: 0;
  background: linear-gradient(135deg, #76b852, #8dc26f);
}

.container {
  width: 80%;
  margin: 20px auto;
  padding: 20px;
  background: #ffffff;
  border-radius: 12px;
  box-shadow: 0 8px 16px rgba(0, 0, 0, 0.2);
  text-align: center;
}

h1 {
  color: #4caf50;
  font-size: 2rem;
  margin-bottom: 20px;
}

.controls {
  margin-bottom: 20px;
  display: flex;
  flex-direction: column;
  border: none;
  border-radius: 8px;
  cursor: pointer;
  font-size: 16px;
  transition: background-color 0.3s ease, transform 0.2s ease;
}

.controls button:hover {
  background-color: #3e8e41;
  transform: translateY(-2px);
}

.output {
  margin-top: 20px;
  padding: 20px;
  background: #f8f9fa;
  border: 1px solid #ddd;
  border-radius: 8px;
  text-align: left;
  max-height: 400px;
  overflow-y: auto;
}

.output h2 {
  font-size: 1.5rem;
  color: #333;
}

.output ul {
  list-style: none;
  padding: 0;
}
```

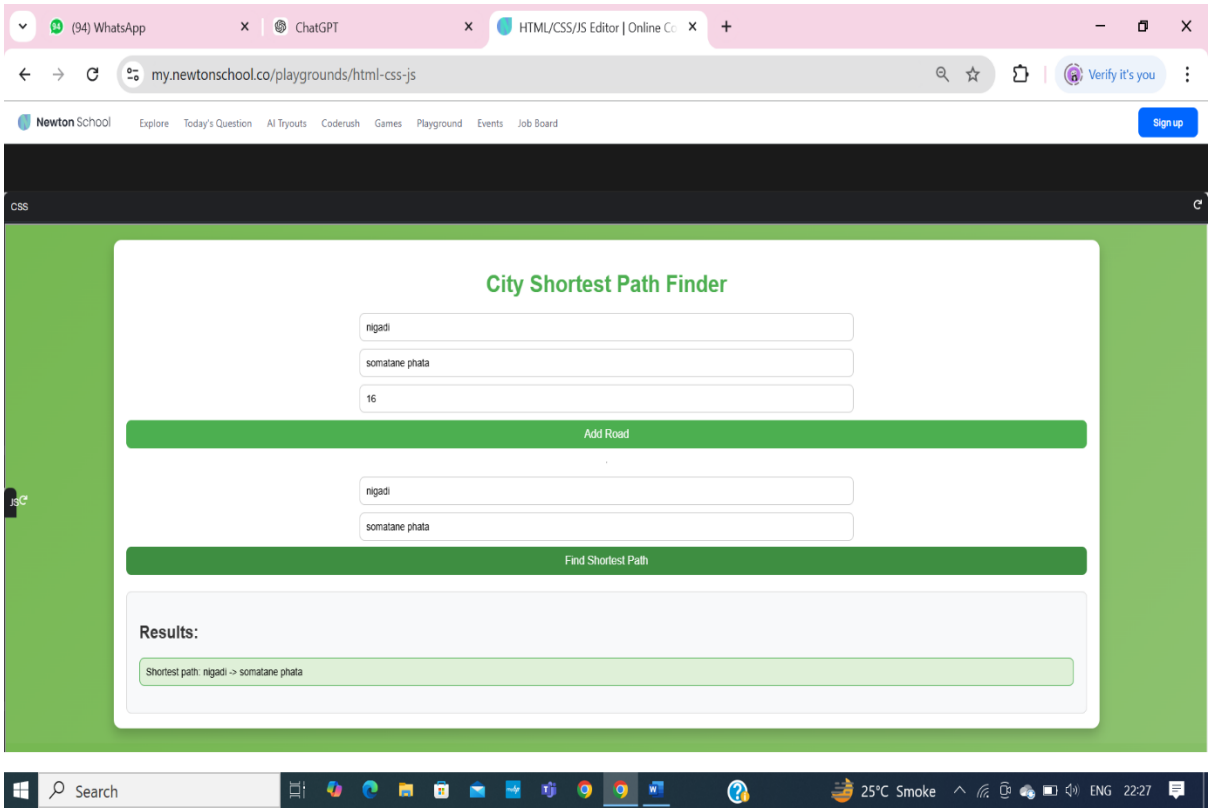
```
    gap: 10px;
}

.controls input {
    padding: 10px;
    width: 50%;
    margin: 0 auto;
    border: 1px solid #ccc;
    border-radius: 8px;
    outline: none;
    transition: box-shadow 0.3s ease-in-out;
    font-size: 14px;
}

.controls input:focus {
    box-shadow: 0 0 8px rgba(76, 175, 80,
0.5);
}

.controls button {
    padding: 10px 20px;
    background: #4caf50;
    color: #fff;
```

JAVASCRIPT:

```
class Graph {
    constructor() {
        this.adjacencyList = {};
    }

    addRoad(intersectionA, intersectionB,
distance) {
        if (!this.adjacencyList[intersectionA])
this.adjacencyList[intersectionA] = [];
        if (!this.adjacencyList[intersectionB])
this.adjacencyList[intersectionB] = [];

        this.adjacencyList[intersectionA].push({
node: intersectionB, distance });
        this.adjacencyList[intersectionB].push({
node: intersectionA, distance });
    }

    findShortestPath(start, end) {
        const distances = {};
        const priorityQueue = new
PriorityQueue();
        const previous = {};
        const path = [];
        let smallest;

        // Initialize distances and priority
queue
        return [];
    }
}

class PriorityQueue {
    constructor() {
        this.values = [];
    }

    enqueue(val, priority) {
        this.values.push({ val, priority });
        this.sort();
    }

    dequeue() {
        return this.values.shift();
    }

    sort() {
        this.values.sort((a, b) => a.priority - b.priority);
    }
}

// Instantiate the graph

const cityGraph = new Graph();

// Helper functions
function updateOutput(message) {
    const output = document.getElementById("output");
```

```
.output li {
    background: #dff0d8;
    border: 1px solid #4caf50;
    padding: 10px;
    margin: 5px 0;
    border-radius: 8px;
    font-size: 14px;
}
```

```
    for (let intersection in
this.adjacencyList) {
      if (intersection === start) {
        distances[intersection] = 0;

priorityQueue.enqueue(intersection, 0);
      } else {
        distances[intersection] = Infinity;

priorityQueue.enqueue(intersection,
Infinity);
      }
      previous[intersection] = null;
    }

    // While there are nodes in the priority
queue
    while (priorityQueue.values.length) {
      smallest =
priorityQueue.dequeue().val;

      if (smallest === end) {
        // Build path and return
        while (previous[smallest]) {
          path.push(smallest);
          smallest = previous[smallest];
        }
        path.push(start);
        return path.reverse();
      }

      if (smallest || distances[smallest] !==
Infinity) {
        for (let neighbor of
this.adjacencyList[smallest]) {
          let candidate =
distances[smallest] + neighbor.distance;

          if (candidate <
distances[neighbor.node]) {
            distances[neighbor.node] =
candidate;
            previous[neighbor.node] =
smallest;

priorityQueue.enqueue(neighbor.node,
candidate);
          }
        }
      }
    }
  output.innerHTML = `<h2>Results:</h2><ul>${message.map(item =>
`<li>${item}</li>`).join("")}</ul>`;
}

function displayError(message) {
  const output = document.getElementById("output");
  output.innerHTML = `<h2 style="color:
red;">Error:</h2><p>${message}</p>`;
}

// Add event listeners
document.getElementById("addRoad").addEventListener("click", () => {
  const intersectionA =
document.getElementById("intersectionA").value.trim();
  const intersectionB =
document.getElementById("intersectionB").value.trim();
  const distance = parseInt(document.getElementById("distance").value);

  if (intersectionA && intersectionB && !isNaN(distance) && distance > 0) {
    cityGraph.addRoad(intersectionA, intersectionB, distance);
    updateOutput([`Road added between "${intersectionA}" and
"${intersectionB}" with distance ${distance}.`]);
  } else {
    displayError("Please provide valid intersections and a positive
distance.");
  }
});

document.getElementById("findPath").addEventListener("click", () => {
  const start = document.getElementById("startPoint").value.trim();
  const end = document.getElementById("endPoint").value.trim();

  if (start && end) {
    const shortestPath = cityGraph.findShortestPath(start, end);
    if (shortestPath.length > 1) {
      updateOutput([`Shortest path: ${shortestPath.join(" -> ")}`]);
    } else {
      updateOutput(["No path found between the given intersections."]);
    }
  } else {
    displayError("Please provide valid start and end intersections.");
  }
});
```



5: File System Management (Tree): Simulate a file system where directories and files are stored in a tree_ $tructure. Implement operations like creating new files, deleting files, and listing files in different traversal orders (pre-order, post-order, in-order).

HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>File System Management</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <div class="container">
    <h1>File System Management</h1>
    <div class="controls">
      <input type="text" id="fileName" placeholder="Enter File/Directory Name">
      <input type="text" id="parentName" placeholder="Parent Directory (Optional)">
      <button id="createFile">Create File/Directory</button>
      <input type="text" id="deleteName" placeholder="Enter Name to Delete">
      <button id="deleteFile">Delete File/Directory</button>
      <hr>
      <button id="listPreOrder">List Files (Pre-order)</button>
      <button id="listInOrder">List Files (In-order)</button>
      <button id="listPostOrder">List Files (Post-order)</button>
    </div>
    <div id="output" class="output"></div>
  </div>
  <script src="script.js"></script>
</body>
</html>
```

CSS:

```
.controls button {
  padding: 10px 20px;
```

```
body {
    font-family: 'Arial', sans-serif;
    margin: 0;
    padding: 0;
    background: linear-gradient(135deg, #8e44ad,
#3498db);
    color: #fff;
}

.container {
    width: 80%;
    margin: 20px auto;
    padding: 20px;
    background: #2c3e50;
    border-radius: 12px;
    box-shadow: 0 8px 16px rgba(0, 0, 0, 0.3);
    text-align: center;
}

h1 {
    color: #ecf0f1;
    font-size: 2rem;
    margin-bottom: 20px;
}

.controls {
    margin-bottom: 20px;
    display: flex;
    flex-direction: column;
    gap: 10px;
}

.controls input {
    padding: 10px;
    width: 50%;
    margin: 0 auto;
    border: 1px solid #ccc;
    border-radius: 8px;
    outline: none;
    font-size: 14px;
    background: #ecf0f1;
    color: #2c3e50;
}
```

JAVASCRIPT:

```
// script.js

class TreeNode {
    constructor(name) {
        this.name = name;
        this.children = [];
    }
}

class FileSystem {
```

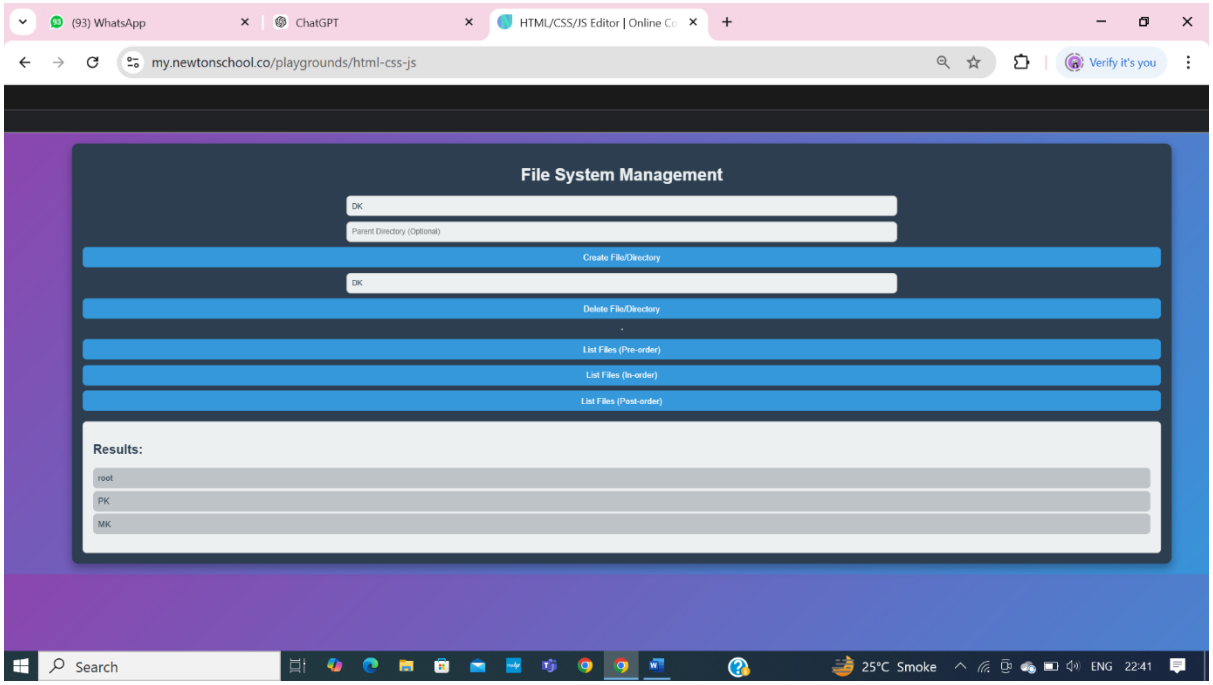
```
    background: #3498db;
    color: #fff;
    border: none;
    border-radius: 8px;
    cursor: pointer;
    font-size: 16px;
    transition: background-color 0.3s ease, transform
0.2s ease;
}

.controls button:hover {
    background-color: #2980b9;
    transform: translateY(-2px);
}

.output {
    margin-top: 20px;
    padding: 20px;
    background: #ecf0f1;
    border-radius: 8px;
    text-align: left;
    color: #2c3e50;
    max-height: 400px;
    overflow-y: auto;
}

.output h2 {
    font-size: 1.5rem;
}

.output ul {
    list-style: none;
    padding: 0;
}

.output li {
    background: #bdc3c7;
    margin: 5px 0;
    padding: 10px;
    border-radius: 8px;
}
```

```
        for (const child of node.children) {
            this.traversePostOrder(child, result);
        }
        result.push(node.name);
        return result;
    }
}

// Instantiate the File System
const fileSystem = new FileSystem();

// Helper function to update output
```

```
  constructor() {
    this.root = new TreeNode("root");
  }

  findNode(name, current = this.root) {
    if (current.name === name) return current;

    for (const child of current.children) {
      const result = this.findNode(name, child);
      if (result) return result;
    }
    return null;
  }

  createNode(name, parentName = "root") {
    const parentNode = this.findNode(parentName);
    if (!parentNode) return `Parent "${parentName}"
not found.`;

    parentNode.children.push(new
TreeNode(name));
    return `"${name}" created under
"${parentName}".`;
  }

  deleteNode(name, current = this.root) {
    const index = current.children.findIndex(child =>
child.name === name);
    if (index !== -1) {
      current.children.splice(index, 1);
      return `"${name}" deleted.`;
    }

    for (const child of current.children) {
      const result = this.deleteNode(name, child);
      if (result) return result;
    }
    return `"${name}" not found.`;
  }

  traversePreOrder(node = this.root, result = []) {
    result.push(node.name);
    for (const child of node.children) {
      this.traversePreOrder(child, result);
    }
    return result;
  }

  traverseInOrder(node = this.root, result = []) {
    if (node.children.length) {
      this.traverseInOrder(node.children[0], result);
    }
    result.push(node.name);
    for (let i = 1; i < node.children.length; i++) {
      this.traverseInOrder(node.children[i], result);
    }
    return result;
```

```
function updateOutput(message) {
  const output =
document.getElementById("output");
  output.innerHTML =
`<h2>Results:</h2><ul>${message.map(item =>
`<li>${item}</li>`).join("")}</ul>`;
}

// Add event listeners
document.getElementById("createFile").addEventLis
tener("click", () => {
  const fileName =
document.getElementById("fileName").value.trim();
  const parentName =
document.getElementById("parentName").value.tri
m() || "root";

  if (fileName) {
    const result = fileSystem.createNode(fileName,
parentName);
    updateOutput([result]);
  } else {
    updateOutput(["Please enter a valid
file/directory name."]);
  }
});

document.getElementById("deleteFile").addEventLis
tener("click", () => {
  const deleteName =
document.getElementById("deleteName").value.tri
m();

  if (deleteName) {
    const result =
fileSystem.deleteNode(deleteName);
    updateOutput([result]);
  } else {
    updateOutput(["Please enter a valid name to
delete."]);
  }
});

document.getElementById("listPreOrder").addEvent
Listener("click", () => {
  const result = fileSystem.traversePreOrder();
  updateOutput(result);
});

document.getElementById("listInOrder").addEventLi
stener("click", () => {
  const result = fileSystem.traverseInOrder();
  updateOutput(result);
});
```

```
    }

    traversePostOrder(node = this.root, result = []) {

document.getElementById("listPostOrder").addEvent
Listener("click", () => {
    const result = fileSystem.traversePostOrder();
    updateOutput(result);
});
```



6: AVL Tree for Stock Price Management: Use an AVL tree to maintain stock prices. Ensure that after jeach insertion, the tree remains balanced by performing rotations.

HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport"
content="width=device-width, initial-scale=1.0">
    <title>Stock Price Management (AVL
Tree)</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="container">
placeholder="Enter Stock Price" min="1">
            <button id="insertStock">Insert Stock Price</button>
            <button id="listInOrder">List Prices (In-order)</button>
            <button id="listPreOrder">List Prices (Pre-order)</button>
            <button id="listPostOrder">List Prices (Post-order)</button>
        </div>
        <div id="output" class="output"></div>
    </div>
    <script src="script.js"></script>
</body>
</html>


border: none;
```

```
    <h1>Stock Price Management (AVL
Tree)</h1>
    <div class="controls">
      <input type="number" id="stockPrice"
```

CSS:

```
body {
  font-family: 'Arial', sans-serif;
  margin: 0;
  padding: 0;
  background: linear-gradient(135deg, #6dd5fa,
#2980b9);
  color: #fff;
}

.container {
  width: 80%;
  margin: 20px auto;
  padding: 20px;
  background: #2c3e50;
  border-radius: 12px;
  box-shadow: 0 8px 16px rgba(0, 0, 0, 0.3);
  text-align: center;
}

h1 {
  font-size: 2rem;
  margin-bottom: 20px;
}

.controls {
  margin-bottom: 20px;
  display: flex;
  flex-direction: column;
  gap: 10px;
}

.controls input {
  padding: 10px;
  width: 50%;
  margin: 0 auto;
  border: 1px solid #ccc;
  border-radius: 8px;
  outline: none;
  font-size: 14px;
  background: #ecf0f1;
  color: #2c3e50;
}

.controls button {
  padding: 10px 20px;
  background: #3498db;
  color: #fff;
  border-radius: 8px;
  cursor: pointer;
  font-size: 16px;
  transition: background-color 0.3s ease, transform 0.2s ease;
}

.controls button:hover {
  background-color: #2980b9;
  transform: translateY(-2px);
}

.output {
  margin-top: 20px;
  padding: 20px;
  background: #ecf0f1;
  border-radius: 8px;
  color: #2c3e50;
  text-align: left;
  max-height: 400px;
  overflow-y: auto;
}

.output h2 {
  font-size: 1.5rem;
}

.output ul {
  list-style: none;
  padding: 0;
}

.output li {
  background: #bdc3c7;
  margin: 5px 0;
  padding: 10px;
  border-radius: 8px;
}
```

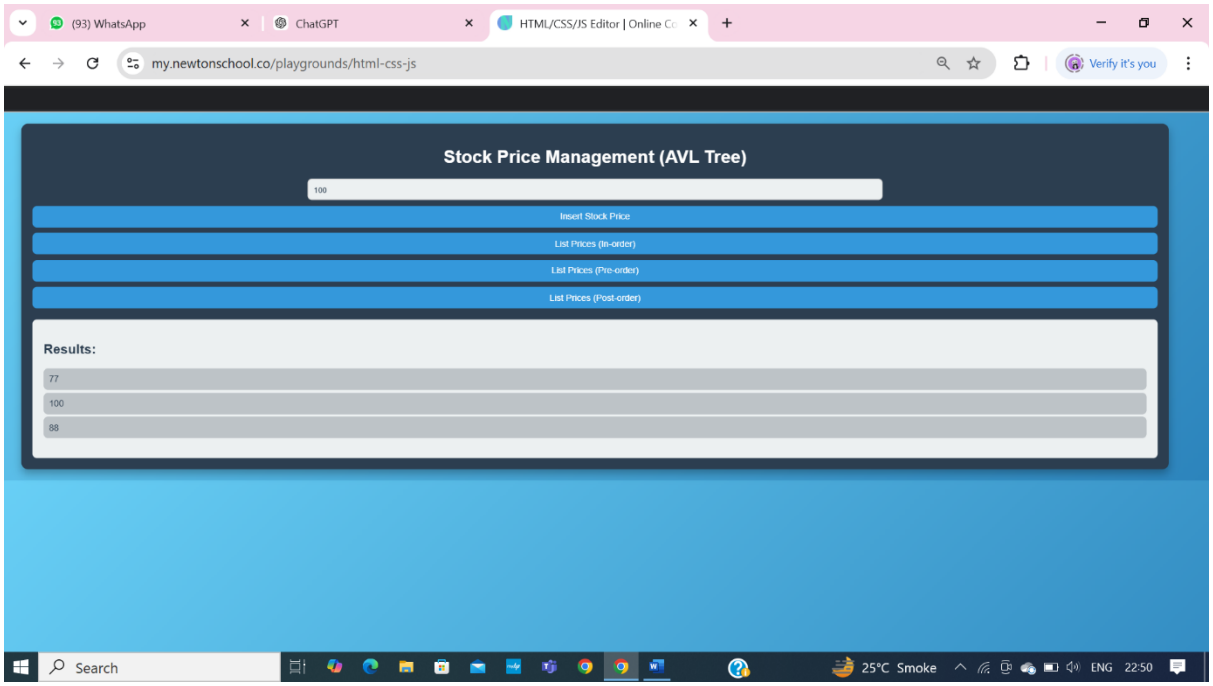
JAVASCRIPT:

```
    // Left-right case
    if (balanceFactor > 1 && value > node.left.value) {
```

```
class AVLNode {
  constructor(value) {
    this.value = value;
    this.left = null;
    this.right = null;
    this.height = 1;
  }
}

class AVLTree {
  constructor() {
    this.root = null;
  }

  getHeight(node) {
    return node ? node.height : 0;
  }

  getBalanceFactor(node) {
    return node ? this.getHeight(node.left) -
this.getHeight(node.right) : 0;
  }

  rotateRight(y) {
    const x = y.left;
    const T2 = x.right;

    // Perform rotation
    x.right = y;
    y.left = T2;

    // Update heights
    y.height = Math.max(this.getHeight(y.left),
this.getHeight(y.right)) + 1;
    x.height = Math.max(this.getHeight(x.left),
this.getHeight(x.right)) + 1;

    return x;
  }

  rotateLeft(x) {
    const y = x.right;
    const T2 = y.left;

    // Perform rotation
    y.left = x;
    x.right = T2;

    // Update heights
    x.height = Math.max(this.getHeight(x.left),
this.getHeight(x.right)) + 1;
    y.height = Math.max(this.getHeight(y.left),
this.getHeight(y.right)) + 1;

    return y;
  }

  insertNode(node, value) {
    if (!node) {
      return new AVLNode(value);
    }

    if (value < node.value) {
      node.left = this.insertNode(node.left, value);
    } else if (value > node.value) {
      node.right = this.insertNode(node.right, value);
    } else {
      return node; // Duplicate values are not allowed
    }

    node.height = 1 + Math.max(this.getHeight(node.left),
this.getHeight(node.right));

    const balanceFactor = this.getBalanceFactor(node);

    // Left-left case
    if (balanceFactor > 1 && value < node.left.value) {
      return this.rotateRight(node);
    }

    // Right-right case
    if (balanceFactor < -1 && value > node.right.value) {
      return this.rotateLeft(node);
    }

    // Left-right case
    if (balanceFactor > 1 && value > node.left.value) {
      node.left = this.rotateLeft(node.left);
      return this.rotateRight(node);
    }

    // Right-left case
    if (balanceFactor < -1 && value < node.right.value) {
      node.right = this.rotateRight(node.right);
      return this.rotateLeft(node);
    }

    return node;
  }

  insert(value) {
    this.root = this.insertNode(this.root, value);
  }

  inOrderTraversal(node = this.root, result = []) {
    if (node) {
      this.inOrderTraversal(node.left, result);
      result.push(node.value);
      this.inOrderTraversal(node.right, result);
    }
    return result;
  }

  preOrderTraversal(node = this.root, result = []) {
    if (node) {
      result.push(node.value);

this.preOrderTraversal(node.left, result);
      this.preOrderTraversal(node.right, result);
    }
    return result;
  }

  postOrderTraversal(node = this.root, result = []) {
    if (node) {
      this.postOrderTraversal(node.left, result);
      this.postOrderTraversal(node.right, result);
      result.push(node.value);
    }
    return result;
  }
}

// Instantiate AVL Tree
const avlTree = new AVLTree();

// Helper function to update output
function updateOutput(message) {
  const output = document.getElementById("output");
  output.innerHTML = `<h2>Results:</h2><ul>${message.map(item
=> `<li>${item}</li>`).join("")}</ul>`;
}

// Add event listeners
```

```
  }

  insertNode(node, value) {
    if (!node) return new AVLNode(value);

    if (value < node.value) {
      node.left = this.insertNode(node.left,
value);
    } else if (value > node.value) {
      node.right = this.insertNode(node.right,
value);
    } else {
      // Duplicate values are not allowed
      return node;
    }

    // Update height
    node.height =
Math.max(this.getHeight(node.left),
this.getHeight(node.right)) + 1;

    // Check balance
    const balanceFactor =
this.getBalanceFactor(node);

    // Left heavy
    if (balanceFactor > 1 && value <
node.left.value) {
      return this.rotateRight(node);
    }

    // Right heavy
    if (balanceFactor < -1 && value >
node.right.value) {
      return this.rotateLeft(node);
    }
```

```
document.getElementById("insertStock").addEventListener("click",
() => {
  const stockPrice =
parseInt(document.getElementById("stockPrice").value);

  if (!isNaN(stockPrice) && stockPrice > 0) {
    avlTree.insert(stockPrice);
    updateOutput([`Stock price ${stockPrice} added.`]);
  } else {
    updateOutput(["Please enter a valid stock price."]);
  }
});

document.getElementById("listInOrder").addEventListener("click", ()
=> {
  const result = avlTree.inOrderTraversal();
  updateOutput(result);
});

document.getElementById("listPreOrder").addEventListener("click",
() => {
  const result = avlTree.preOrderTraversal();
  updateOutput(result);
});

document.getElementById("listPostOrder").addEventListener("click",
() => {
  const result = avlTree.postOrderTraversal();
  updateOutput(result);
});
```


(93) WhatsApp
ChatGPT
HTML/CSS/JS Editor | Online Co
my.newtonschool.co/playgrounds/html-css-js
Verify it's you
Stock Price Management (AVL Tree)
100
Insert Stock Price
List Prices (In-order)
List Prices (Pre-order)
List Prices (Post-order)
Results:
77
100
88
Search
25°C Smoke
ENG
22:50

7: Graph Coloring Problem (Greedy): Solve the graph coloring problem using a greedy algorithm to |inimize the number of colors needed to color a graph such that no two adjacent nodes share the same color.

HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Graph Coloring Problem</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <div class="container">
    <h1>Graph Coloring Problem (Greedy Algorithm)</h1>
    <div class="controls">
      <input type="text" id="node1" placeholder="Node 1">
      <input type="text" id="node2" placeholder="Node 2">
      <button id="addEdge">Add Edge</button>
      <button id="colorGraph">Color Graph</button>
      <hr>
      <div id="graphOutput"></div>
    </div>
    <div id="result" class="output"></div>
  </div>
  <script src="script.js"></script>
</body>
</html>
```

CSS:

```
/* styles.css */
body {
  font-family: 'Arial', sans-serif;
  margin: 0;
  padding: 0;
  background: linear-gradient(135deg, #ff7e5f, #feb47b);
  color: #fff;
}

.container {
  width: 80%;
  margin: 20px auto;
  padding: 20px;
  background: #2c3e50;
  border-radius: 12px;
  box-shadow: 0 8px 16px rgba(0, 0, 0, 0.3);
  text-align: center;
}

h1 {
  font-size: 2rem;
  margin-bottom: 20px;
}

.controls {
  margin-bottom: 20px;
  display: flex;
  flex-direction: column;
  gap: 10px;
}

.controls input {
.controls button {
  padding: 10px 20px;
  background: #3498db;
  color: #fff;
  border: none;
  border-radius: 8px;
  cursor: pointer;
  font-size: 16px;
  transition: background-color 0.3s ease, transform 0.2s ease;
}

.controls button:hover {
  background-color: #2980b9;
  transform: translateY(-2px);
}

#graphOutput {
  margin: 20px 0;
  padding: 10px;
  background: #ecf0f1;
  border-radius: 8px;
  color: #2c3e50;
  max-height: 300px;
  overflow-y: auto;
}

.output {
  margin-top: 20px;
  padding: 20px;
  background: #ecf0f1;
  border-radius: 8px;
  color: #2c3e50;
  text-align: left;
```

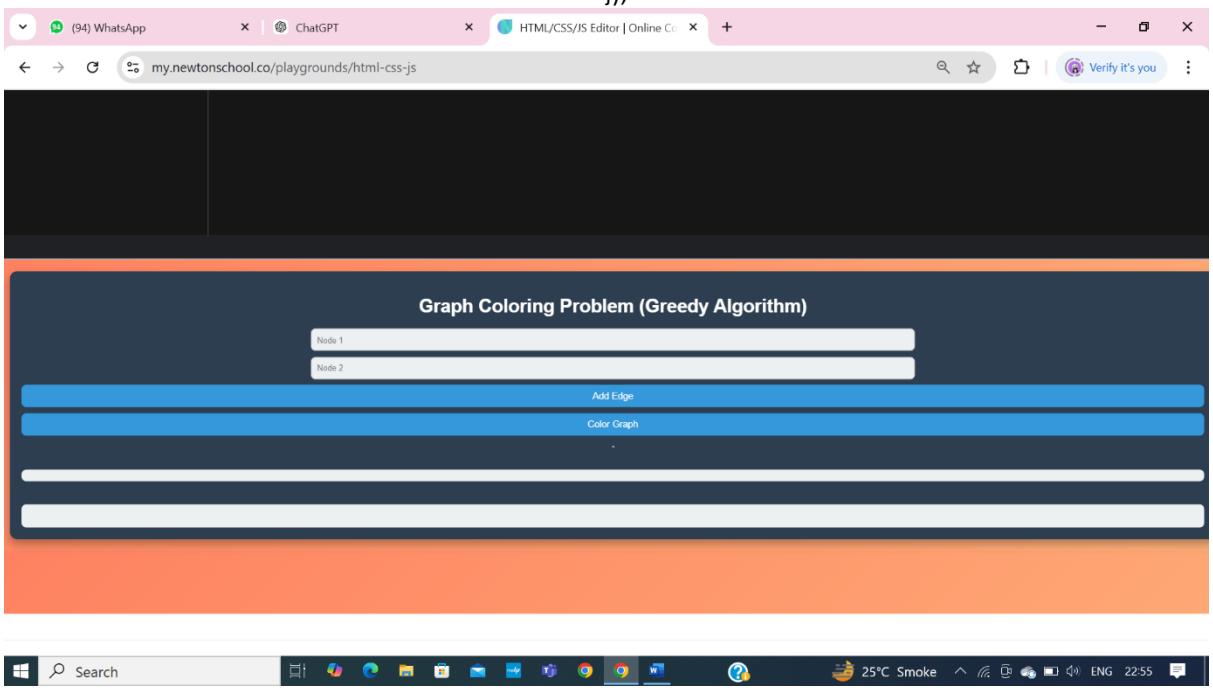
```
    padding: 10px;
    width: 50%;
    margin: 0 auto;
    border: 1px solid #ccc;
    border-radius: 8px;
    outline: none;
    font-size: 14px;
    background: #ecf0f1;
    color: #2c3e50;
}
```

```
    max-height: 400px;
    overflow-y: auto;
}

.output ul {
    list-style: none;
    padding: 0;
}

.output li {
    background: #bdc3c7;
    margin: 5px 0;
    padding: 10px;
    border-radius: 8px;
}
```

JAVASCRIPT:

```
class Graph {
    constructor() {
        this.adjacencyList = {};
    }

    addEdge(node1, node2) {
        if (!this.adjacencyList[node1]) {
            this.adjacencyList[node1] = [];
        }
        if (!this.adjacencyList[node2]) {
            this.adjacencyList[node2] = [];
        }
        this.adjacencyList[node1].push(node2);
        this.adjacencyList[node2].push(node1);
    }

    greedyColoring() {
        const colors = {};
        const result = {};
        const nodes = Object.keys(this.adjacencyList);

        for (const node of nodes) {
            const usedColors = new Set(
                this.adjacencyList[node].map(neighbor =>
colors[neighbor])
            );

            let color = 1;
            while (usedColors.has(color)) {
                color++;
            }

            colors[node] = color;
        }
        return result;
    }

    visualize() {
        const output = document.getElementById("graphOutput");
        output.innerHTML = "<h3>Graph Structure:</h3>";
        for (const node in this.adjacencyList) {
            output.innerHTML += `<p>${node} ->
[${this.adjacencyList[node].join(", ")}]</p>`;
        }
    }
}

const graph = new Graph();

document.getElementById("addEdge").addEventListener("click", ()
=> {
    const node1 = document.getElementById("node1").value.trim();
    const node2 = document.getElementById("node2").value.trim();

    if (node1 && node2) {
        graph.addEdge(node1, node2);
        graph.visualize();
        document.getElementById("node1").value = "";
        document.getElementById("node2").value = "";
    } else {
        alert("Please enter valid node names.");
    }
});

document.getElementById("colorGraph").addEventListener("click",
() => {
    const coloring = graph.greedyColoring();
    const output = document.getElementById("result");
    output.innerHTML = "<h2>Graph Coloring:</h2><ul>";
    for (const node in coloring) {
```

```
result[node] = `Node ${node} -> Color ${color}`;
        output.innerHTML += `<li>${coloring[node]}</li>`;
    }
    output.innerHTML += "</ul>";
});
```



8.Minimum Spanning Tree for a Power Grid: Implement Kruskal's algorithm to find the minimum spanning tree (MST) for a power grid system connecting cities. Each city is a node, and each connection between cities has a cost.

HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Minimum Spanning Tree (Kruskal's Algorithm)</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <div class="container">
    <h1>Minimum Spanning Tree for Power Grid</h1>
    <div class="controls">
      <input type="text" id="city1" placeholder="City 1">
      <input type="text" id="city2" placeholder="City 2">
      <input type="number" id="cost" placeholder="Connection Cost" min="1">
      <button id="addConnection">Add Connection</button>
      <button id="calculateMST">Calculate MST</button>
    </div>
    <div id="graphOutput"></div>
    <div id="result" class="output"></div>
  </div>
  <script src="script.js"></script>
</body>
</html>
```

CSS:

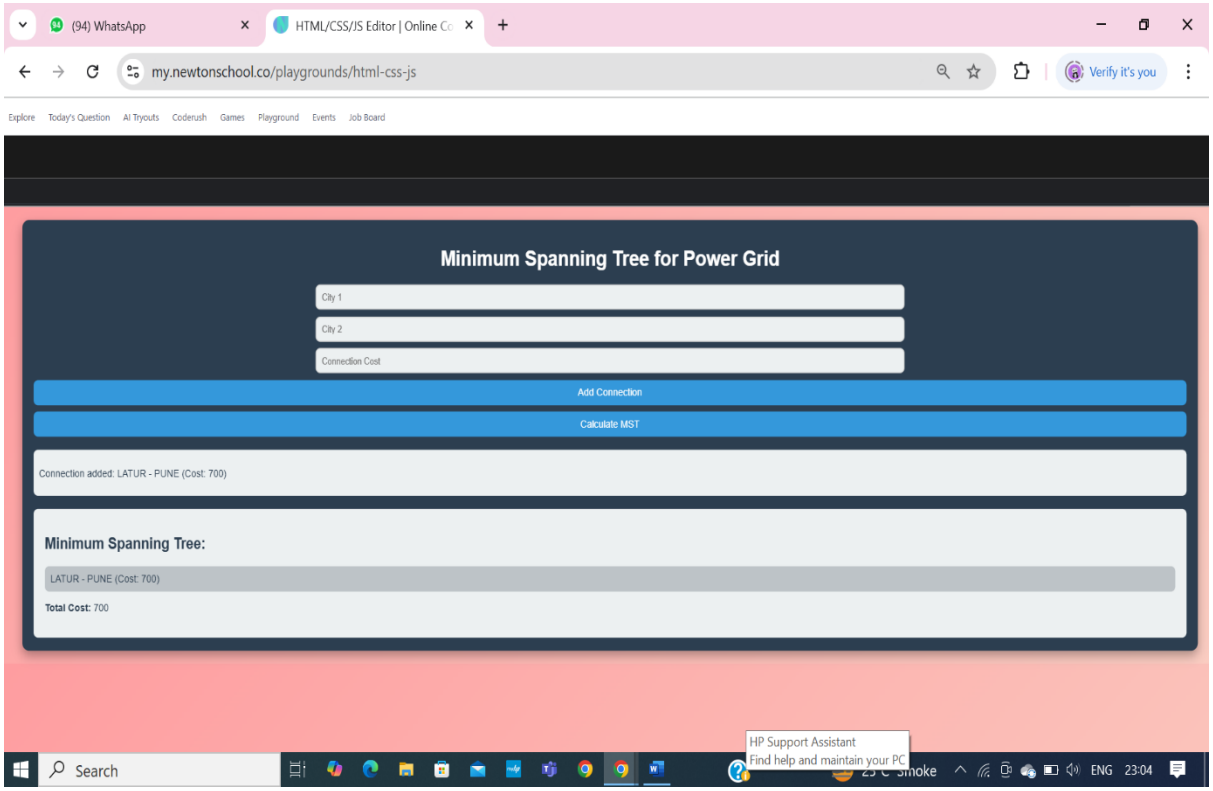
```
controls button:hover {
```

```
/* styles.css */
body {
    font-family: 'Arial', sans-serif;
    margin: 0;
    padding: 0;
    background: linear-gradient(135deg, #ff9a9e,
#fad0c4);
    color: #2c3e50;
}

.container {
    width: 80%;
    margin: 20px auto;
    padding: 20px;
    background: #2c3e50;
    border-radius: 12px;
    box-shadow: 0 8px 16px rgba(0, 0, 0, 0.3);
    text-align: center;
    color: #fff;
}

h1 {
    font-size: 2rem;
    margin-bottom: 20px;
}

.controls {
    margin-bottom: 20px;
    display: flex;
    flex-direction: column;
    gap: 10px;
}

.controls input {
    padding: 10px;
    width: 50%;
    margin: 0 auto;
    border: 1px solid #ccc;
    border-radius: 8px;
    outline: none;
    font-size: 14px;
    background: #ecf0f1;
    color: #2c3e50;
}

.controls button {
    padding: 10px 20px;
    background: #3498db;
    color: #fff;
    border: none;
    border-radius: 8px;
    cursor: pointer;
    font-size: 16px;
    transition: background-color 0.3s ease, transform 0.2s
ease;
}
    background-color: #2980b9;
    transform: translateY(-2px);
}

#graphOutput {
    margin: 20px 0;
    padding: 10px;
    background: #ecf0f1;
    border-radius: 8px;
    color: #2c3e50;
    max-height: 300px;
    overflow-y: auto;
    text-align: left;
}

.output {
    margin-top: 20px;
    padding: 20px;
    background: #ecf0f1;
    border-radius: 8px;
    color: #2c3e50;
    text-align: left;
    max-height: 400px;
    overflow-y: auto;
}

.output ul {
    list-style: none;
    padding: 0;
}

.output li {
    background: #bdc3c7;
    margin: 5px 0;
    padding: 10px;
    border-radius: 8px;
}
```

.JAVASCRIPT:

```
// script.js

class Edge {
  constructor(city1, city2, cost) {
    this.city1 = city1;
    this.city2 = city2;
    this.cost = cost;
  }
}

class Graph {
  constructor() {
    this.edges = [];
    this.cities = new Set();
  }

  addEdge(city1, city2, cost) {
    this.edges.push(new Edge(city1, city2, cost));
    this.cities.add(city1);
    this.cities.add(city2);
  }

  findParent(parent, city) {
    if (parent[city] !== city) {
      parent[city] = this.findParent(parent,
parent[city]);
    }
    return parent[city];
  }

  union(parent, rank, city1, city2) {
    const root1 = this.findParent(parent, city1);
    const root2 = this.findParent(parent, city2);

    if (rank[root1] < rank[root2]) {
      parent[root1] = root2;
    } else if (rank[root1] > rank[root2]) {
      parent[root2] = root1;
    } else {
      parent[root2] = root1;
      rank[root1]++;
    }
  }

  kruskalMST() {
    this.edges.sort((a, b) => a.cost - b.cost);

    const parent = {};
    const rank = {};
    const result = [];

    this.cities.forEach(city => {
      parent[city] = city;
      rank[city] = 0;
    });

    for (const edge of this.edges) {
      if (this.findParent(parent, edge.city1) !==
this.findParent(parent, edge.city2)) {
        result.push(edge);
        this.union(parent, rank, edge.city1, edge.city2);
      }
    }

    return result;
  }
}

const graph = new Graph();

document.getElementById("addConnection").addEventListener("click",
() => {
  const city1 = document.getElementById("city1").value.trim();
  const city2 = document.getElementById("city2").value.trim();
  const cost = parseInt(document.getElementById("cost").value);

  if (city1 && city2 && !isNaN(cost) && cost > 0) {
    graph.addEdge(city1, city2, cost);
    const output = document.getElementById("graphOutput");
    output.innerHTML += `<p>Connection added: ${city1} - ${city2}
(Cost: ${cost})</p>`;
    document.getElementById("city1").value = "";
    document.getElementById("city2").value = "";
    document.getElementById("cost").value = "";
  } else {
    alert("Please enter valid cities and cost.");
  }
});

document.getElementById("calculateMST").addEventListener("click",
() => {
  const mst = graph.kruskalMST();
  const output = document.getElementById("result");

  if (mst.length === 0) {
    output.innerHTML = "<h2>No connections available to calculate
MST.</h2>";
    return;
  }

  let totalCost = 0;
  const resultHTML = mst.map(edge => {
    totalCost += edge.cost;
    return `<li>${edge.city1} - ${edge.city2} (Cost: ${edge.cost})</li>`;
  }).join("");

  output.innerHTML = `<h2>Minimum Spanning
Tree:</h2><ul>${resultHTML}</ul><p><strong>Total Cost:</strong>
${totalCost}</p>`;
});
```

```
for (const edge of this.edges) {
  const root1 = this.findParent(parent, edge.city1);
  const root2 = this.findParent(parent, edge.city2);

  if (root1 !== root2) {
    result.push(edge);
```



9: Red-Black Tree for Dynamic Leaderboard: Implement a red-black tree to manage a dynamic gaming leaderboard, As players gain points, their rank in the tree adjusts in real time.

HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Red-Black Tree Leaderboard</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <div class="container">
    <h1>Red-Black Tree Leaderboard</h1>
    <div class="controls">
      <input type="text" id="playerName" placeholder="Player Name">
      <input type="number" id="playerScore" placeholder="Score" min="0">
      <button id="addPlayer">Add Player</button>
    </div>
    <div id="leaderboard" class="output"></div>
  </div>
  <script src="script.js"></script>
</body>
</html>
```

CSS:

```
/* styles.css */
body {
  font-family: 'Arial', sans-serif;
  margin: 0;
  padding: 0;
  background: linear-gradient(135deg, #ff6a00, #ee0979);
  color: #fff;
}

.container {
  width: 80%;
  margin: 20px auto;
  padding: 20px;
  background: #2c3e50;
  border-radius: 12px;
  box-shadow: 0 8px 16px rgba(0, 0, 0, 0.3);
  text-align: center;
}

h1 {
  font-size: 2rem;
  margin-bottom: 20px;
}

.controls {
  margin-bottom: 20px;
  display: flex;
  flex-direction: column;
  gap: 10px;
}

.controls input {
  padding: 10px;
  width: 50%;
  margin: 0 auto;
  border: 1px solid #ccc;
  border-radius: 8px;
  outline: none;
  font-size: 14px;
  background: #ecf0f1;
  color: #2c3e50;
}

.controls button {
  padding: 10px 20px;
  background: #3498db;
  color: #fff;
  border: none;
  border-radius: 8px;
  cursor: pointer;
  font-size: 16px;

transition: background-color 0.3s ease, transform 0.2s ease;
}

.controls button:hover {
  background-color: #2980b9;
  transform: translateY(-2px);
}

.output {
  margin-top: 20px;
  padding: 20px;
  background: #ecf0f1;
  border-radius: 8px;
  color: #2c3e50;
  text-align: left;
  max-height: 400px;
  overflow-y: auto;
}

.output ul {
  list-style: none;
  padding: 0;
}

.output li {
  background: #bdc3c7;
  margin: 5px 0;
  padding: 10px;
  border-radius: 8px;
}
```

JAVASCRIPT:

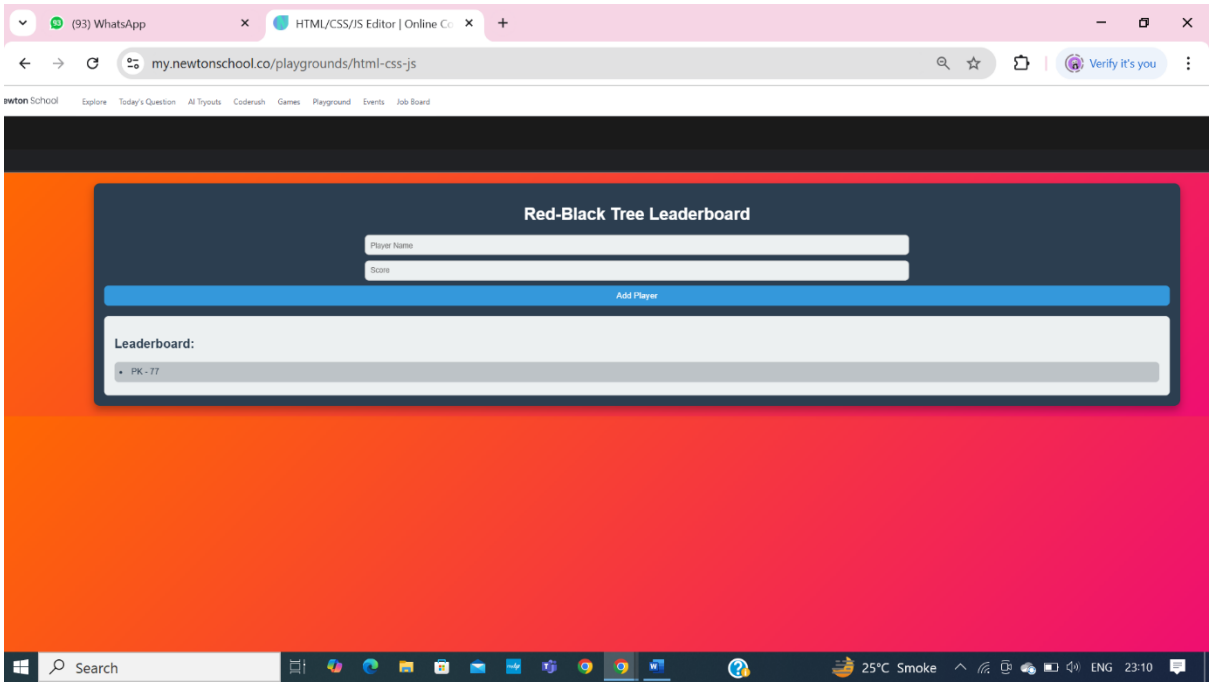
```
class RedBlackTreeNode {
  constructor(data, color = 'RED') {
    this.data = data;
    this.color = color;
    this.parent = null;
    this.left = null;
    this.right = null;
  }
```

```
        this.rotateRight(node);
      }
      node.parent.color = 'BLACK';
      node.parent.parent.color = 'RED';
      this.rotateLeft(node.parent.parent);
    }
  }
}
```

```
}

class RedBlackTree {
  constructor() {
    this.NIL = new RedBlackTreeNode(null);
    this.NIL.color = 'BLACK';
    this.root = this.NIL;
  }

  insert(data) {
    let newNode = new RedBlackTreeNode(data);
    newNode.left = this.NIL;
    newNode.right = this.NIL;

    if (this.root === this.NIL) {
      this.root = newNode;
      this.root.color = 'BLACK';
    } else {
      let parent = null;
      let current = this.root;

      while (current !== this.NIL) {
        parent = current;
        if (data.score < current.data.score) {
          current = current.left;
        } else {
          current = current.right;
        }
      }

      newNode.parent = parent;

      if (data.score < parent.data.score) {
        parent.left = newNode;
      } else {
        parent.right = newNode;
      }

      this.fixInsert(newNode);
    }
  }

  fixInsert(node) {
    while (node !== this.root && node.parent.color ===
'RED') {
      if (node.parent === node.parent.parent.left) {
        let uncle = node.parent.parent.right;
        if (uncle.color === 'RED') {
          node.parent.color = 'BLACK';
          uncle.color = 'BLACK';
          node.parent.parent.color = 'RED';
          node = node.parent.parent;
        } else {
          if (node === node.parent.right) {
            node = node.parent;
            this.rotateLeft(node);
          }
          this.root.color = 'BLACK';
  }

  rotateLeft(x) {
    let y = x.right;
    x.right = y.left;
    if (y.left !== this.NIL) {
      y.left.parent = x;
    }
    y.parent = x.parent;
    if (x.parent === null) {
      this.root = y;
    } else if (x === x.parent.left) {
      x.parent.left = y;
    } else {
      x.parent.right = y;
    }
    y.left = x;
    x.parent = y;
  }

  rotateRight(x) {
    let y = x.left;
    x.left = y.right;
    if (y.right !== this.NIL) {
      y.right.parent = x;
    }
    y.parent = x.parent;
    if (x.parent === null) {
      this.root = y;
    } else if (x === x.parent.right) {
      x.parent.right = y;
    } else {
      x.parent.left = y;
    }
    y.right = x;
    x.parent = y;
  }

  inOrderTraversal(node, result) {
    if (node !== this.NIL) {
      this.inOrderTraversal(node.left, result);
      result.push(node.data);
      this.inOrderTraversal(node.right, result);
    }
  }
}

const leaderboard = new RedBlackTree();

document.getElementById("addPlayer").addEventListener("click",
() => {
  const playerName =
document.getElementById("playerName").value.trim();
  const playerScore =
parseInt(document.getElementById("playerScore").value);
```

```
node.parent.color = 'BLACK';
                node.parent.parent.color = 'RED';
                this.rotateRight(node.parent.parent);
            }
        } else {
            let uncle = node.parent.parent.left;
            if (uncle.color === 'RED') {
                node.parent.color = 'BLACK';
                uncle.color = 'BLACK';
                node.parent.parent.color = 'RED';
                node = node.parent.parent;
            } else {
                if (node === node.parent.left) {
                    node = node.parent;
```

```
    if (playerName && !isNaN(playerScore)) {
        leaderboard.insert({ name: playerName, score: playerScore
});
        displayLeaderboard();
        document.getElementById("playerName").value = "";
        document.getElementById("playerScore").value = "";
    } else {
        alert("Please enter a valid name and score.");
    }
});

function displayLeaderboard() {
    const result = [];
    leaderboard.inOrderTraversal(leaderboard.root, result);

    const output = document.getElementById("leaderboard");
    output.innerHTML = "<h2>Leaderboard:</h2><ul>";
    result.reverse().forEach(player => {
        output.innerHTML += `<li>${player.name} -
${player.score}</li>`;
    });
    output.innerHTML += "</ul>";
}
```



10: Cyele Detection in Graph: Implement a graph traversal algorithm (DES) to detect cycles in a directed and undirected graph, simulating dependencies between software modules.

HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport"
content="width=device-width, initial-
scale=1.0">
  <title>Cycle Detection in Graph</title>
  <link rel="stylesheet" href="styles.css">
</head>
<body>
  <div class="container">
    <h1>Cycle Detection in Graph</h1>
    <div class="controls">
      <input type="text" id="module1"
placeholder="Module 1">
      <input type="text" id="module2" placeholder="Module
2">
      <button id="addEdge">Add Dependency</button>
      <button id="checkCycle">Check for Cycle</button>
    </div>
    <div id="graphOutput" class="output"></div>
    <div id="result" class="output"></div>
  </div>
  <script src="script.js"></script>
</body>
</html>
```

CSS:

```
/* styles.css */
body {
  font-family: 'Arial', sans-serif;
  margin: 0;
  padding: 0;
  background: linear-gradient(135deg,
#fbc2eb, #a18cd1);
  color: #2c3e50;
}

.container {
  width: 80%;
  margin: 20px auto;
  padding: 20px;
  background: #2c3e50;
  border-radius: 12px;
  box-shadow: 0 8px 16px rgba(0, 0, 0,
0.3);
  text-align: center;
  color: #fff;
}

h1 {
  font-size: 2rem;
  margin-bottom: 20px;
}

.controls {
  margin-bottom: 20px;
  display: flex;
  flex-direction: column;
  gap: 10px;
}

controls button {
  padding: 10px 20px;
  background: #3498db;
  color: #fff;
  border: none;
  border-radius: 8px;
  cursor: pointer;
  font-size: 16px;
  transition: background-color 0.3s ease, transform 0.2s ease;
}

.controls button:hover {
  background-color: #2980b9;
  transform: translateY(-2px);
}

.output {
  margin-top: 20px;
  padding: 20px;
  background: #ecf0f1;
  border-radius: 8px;
  color: #2c3e50;
  text-align: left;
  max-height: 400px;
  overflow-y: auto;
}

.output ul {
  list-style: none;
  padding: 0;
}

.output li {
  background: #bdc3c7;
  margin: 5px 0;
  padding: 10px;
```

```
.controls input {
  padding: 10px;
  width: 45%;
  margin: 0 auto;
  border: 1px solid #ccc;
  border-radius: 8px;
  outline: none;
  font-size: 14px;
  background: #ecf0f1;
  color: #2c3e50;
}
```

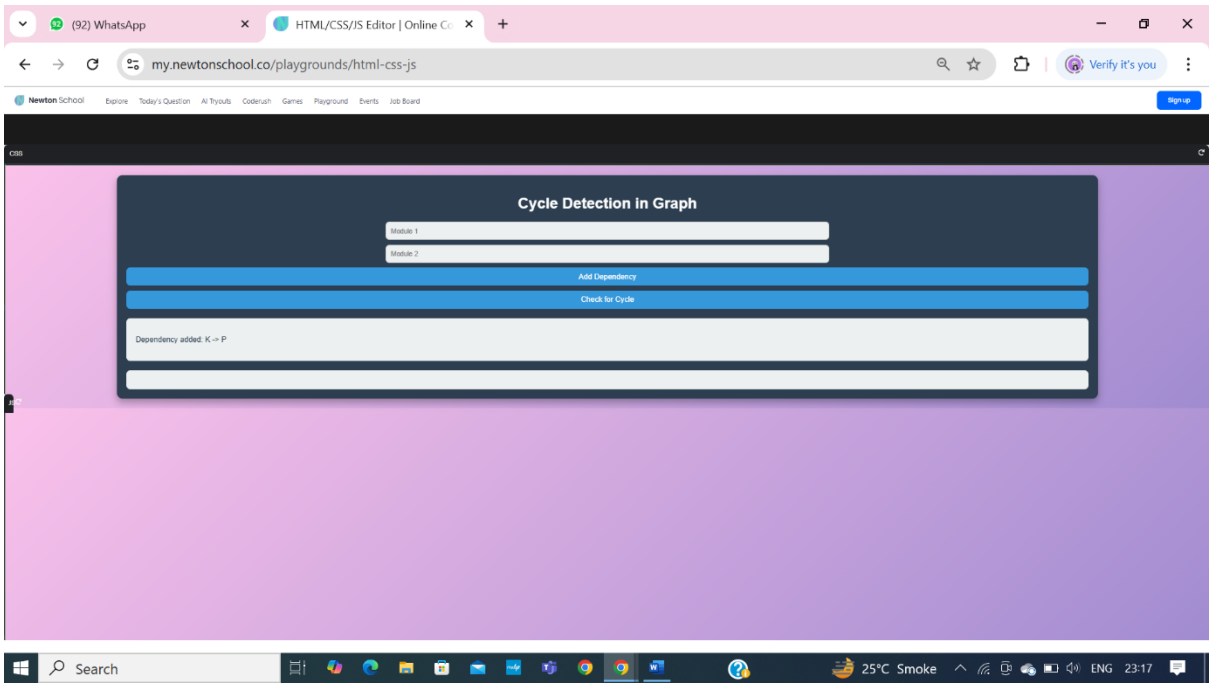
JAVASCRIPT:

```
class Graph {
  constructor() {
    this.adjacencyList = {};
  }

  addEdge(module1, module2) {
    if (!this.adjacencyList[module1]) {
      this.adjacencyList[module1] = [];
    }

this.adjacencyList[module1].push(module
2);
  }

  detectCycle() {
    const visited = new Set();
    const recursionStack = new Set();

    for (const module in
this.adjacencyList) {
      if (!visited.has(module)) {
        if (this._dfs(module, visited,
recursionStack)) {
          return true;
        }
      }
    }
    return false;
  }

  _dfs(module, visited, recursionStack) {
    visited.add(module);
    recursionStack.add(module);

    for (const neighbor of
this.adjacencyList[module]) {
      if (!visited.has(neighbor)) {
        if (this._dfs(neighbor, visited,
recursionStack)) {
          return true;
        }
```

```
    border-radius: 8px;
}
```

```
const graph = new Graph();

document.getElementById("addEdge").addEventListener("click
", () => {
  const module1 =
document.getElementById("module1").value.trim();
  const module2 =
document.getElementById("module2").value.trim();

  if (module1 && module2) {
    graph.addEdge(module1, module2);
    const output = document.getElementById("graphOutput");
    output.innerHTML += `<p>Dependency added: ${module1}
-> ${module2}</p>`;
    document.getElementById("module1").value = "";
    document.getElementById("module2").value = "";
  } else {
    alert("Please enter valid module names.");
  }
});

document.getElementById("checkCycle").addEventListener("cli
ck", () => {
  const hasCycle = graph.detectCycle();
  const result = document.getElementById("result");
  if (hasCycle) {
    result.innerHTML = "<h2>Cycle detected in the
graph!</h2>";
  } else {
    result.innerHTML = "<h2>No cycle detected in the
graph.</h2>";
  }
});
```

```
        } else if
(recursionStack.has(neighbor)) {
            return true;
        }
    }

    recursionStack.delete(module);
    return false;
  }
}
```



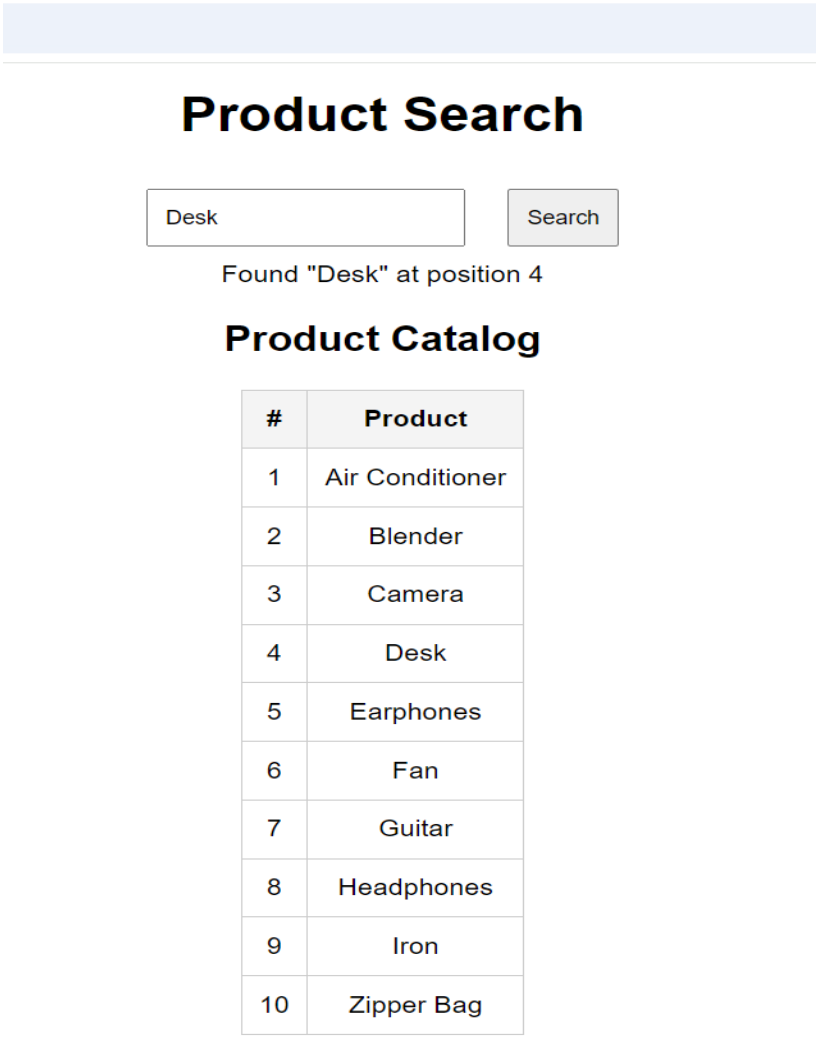
MODULE5

1: E-commerce Product Search with Binary Search: Implement a binary search algorithm to search for products in a sorted product catalog. Compare its performance against linear search.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Product Search</title>
  <style>
    body { font-family: Arial, sans-serif; text-align: center; padding: 20px; }
    input, button { padding: 10px; margin: 10px; }
    table { margin: 20px auto; border-collapse: collapse; }
    th, td { border: 1px solid #ccc; padding: 10px; }
    th { background-color: #f4f4f4; }
  </style>
</head>
<body>
  <h1>Product Search</h1>
  <input type="text" id="search" placeholder="Search product">
  <button onclick="searchProduct()">Search</button>
  <div id="result"></div>
  <h2>Product Catalog</h2>
  <table>
    <tr><th>#</th><th>Product</th></tr>
    <tbody id="catalog"></tbody>
  </table>
  <script>
    const products = [
```

```
        "Air Conditioner", "Blender", "Camera", "Desk", "Earphones",
        "Fan", "Guitar", "Headphones", "Iron", "Zipper Bag"
    ];
const catalog = document.getElementById("catalog");
    products.forEach((product, i) => {
        catalog.innerHTML += `<tr><td>${i + 1}</td><td>${product}</td></tr>`;
    });
function searchProduct() {
        const search = document.getElementById("search").value.trim();
        const result = document.getElementById("result");
        const binarySearch = (arr, target) => {
  let left = 0, right = arr.length - 1;
   while (left <= right) {
            const mid = Math.floor((left + right) / 2);
    if (arr[mid] === target) return mid;
    if (arr[mid] < target) left = mid + 1;
            else right = mid - 1;
          }
          return -1;
        };
const index = binarySearch(products, search);
 result.innerText = index !== -1
          ? `Found "${search}" at position ${index + 1}`
   : `"${search}" not found in catalog.`;
    } </script></body></html>
```

output:


Product Search
Desk
Search
Found "Desk" at position 4
Product Catalog
Product
1 Air Conditioner
2 Blender
3 Camera
4 Desk
5 Earphones
6 Fan
7 Guitar
8 Headphones
9 Iron
10 Zipper Bag

2: Contact List Sorting (Merge Sort): Sort a large list of phone contacts using merge sort and compare the time complexity with quick sort when applied to smaller lists.

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Contact List Sorting</title>
   <style>
      body {
         font-family: Arial, sans-serif;
         background-color: #f4f4f9;
         margin: 0;
         padding: 0;
         display: flex;
         justify-content: center;
         align-items: center;
         height: 100vh; }
      .container {
         text-align: center;
         width: 60%;         }
      h1 {
     margin-bottom: 20px; }
.contact-list {
         margin-top: 20px;
         list-style-type: none;
         padding: 0;
```

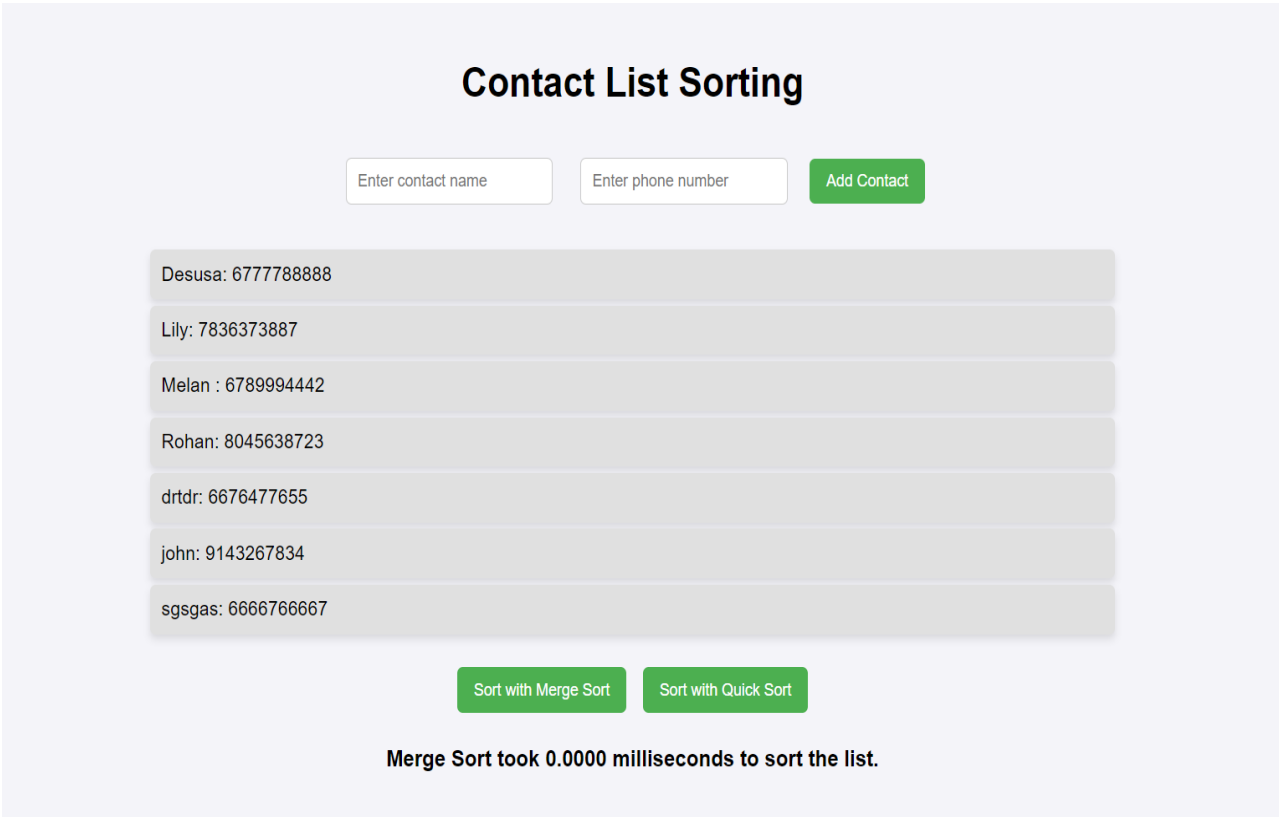
```
    display: inline-block;
    text-align: left;
    width: 100%;        }
    .contact-list li {
    background-color: #e0e0e0;
    padding: 10px;
    margin: 5px 0;
    border-radius: 5px;
    box-shadow: 0 4px 6px rgba(0, 0, 0,           0.1); }
.input-field {
    padding: 10px;
    margin: 10px;
    border-radius: 5px;
    border: 1px solid #ccc; }
.form-container {
    margin-top: 30px; }
button {
    padding: 10px 15px;
    border: none;
    border-radius: 5px;
    background-color: #4CAF50;
    color: white;
    cursor: pointer;
    margin: 5px;
    transition: background-color 0.3s ease;  }
button:hover {
    background-color: #45a049;         }

.output {
```

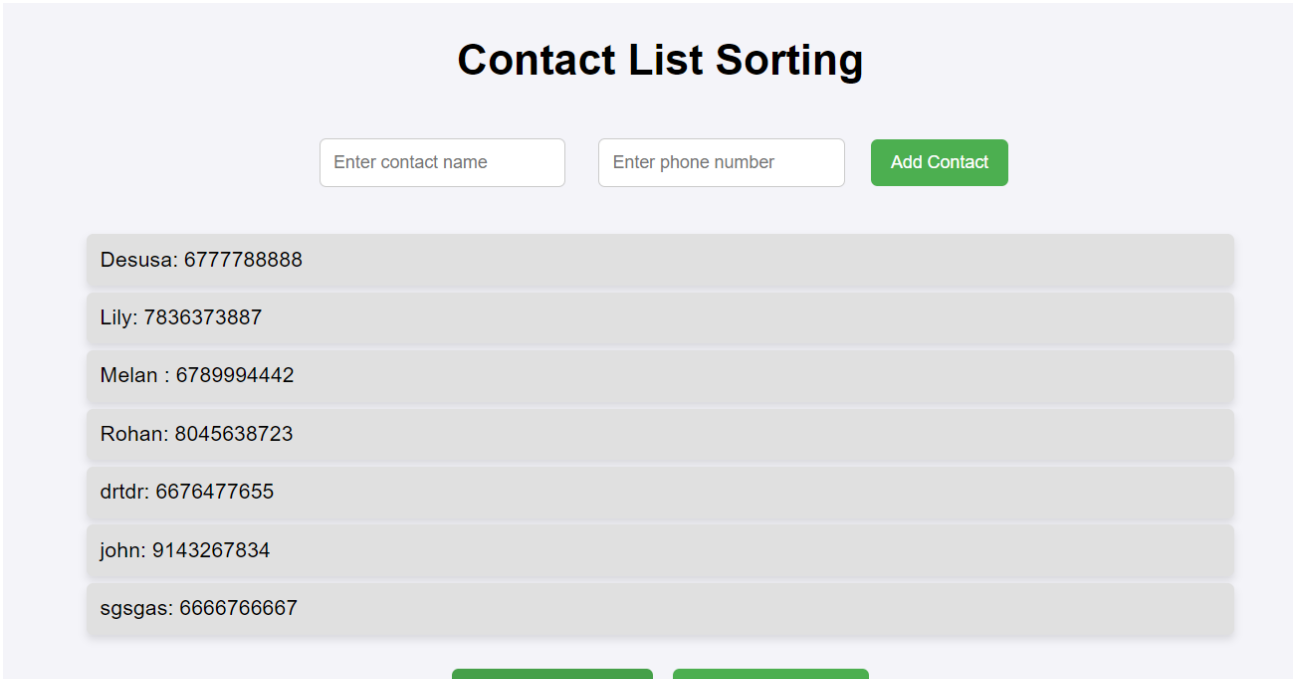
```
        margin-top: 20px;
        font-size: 18px;
        font-weight: bold;
    }</style></head><body>
  <div class="container">
    <h1>Contact List Sorting</h1>
    <div class="form-container">
<input type="text" class="input-field" id="contactName" placeholder="Enter contact name">
<input type="text" class="input-field" id="contactPhone" placeholder="Enter phone number">
        <button onclick="addContact()">Add Contact</button>
    </div>
    <ul class="contact-list" id="contactList"></ul>
    <div>
        <button onclick="sortWithMergeSort()">Sort with Merge Sort</button>
        <button onclick="sortWithQuickSort()">Sort with Quick Sort</button>
    </div>
    <div class="output" id="output"></div>
  </div> <script>
    let contacts = [];
    function addContact() {
        const name = document.getElementById('contactName').value;
        const phone = document.getElementById('contactPhone').value;
        if (name && phone) {
            contacts.push({ name, phone });
            document.getElementById('contactName').value = '';
document.getElementById('contactPhone').value = '';
  displayContacts();  } }
    function displayContacts() {
```

```
        const list = document.getElementById('contactList');
        list.innerHTML = '';
        contacts.forEach(contact => {
            const li = document.createElement('li');
            li.textContent = `${contact.name}: ${contact.phone}`;
            list.appendChild(li);
        }); }
    function mergeSort(arr) {
        if (arr.length <= 1) return arr;
 const middle = Math.floor(arr.length / 2);
        const left = arr.slice(0, middle);
        const right = arr.slice(middle);
 return merge(mergeSort(left), mergeSort(right));
    }
    function merge(left, right) {
        let result = [];
        let leftIndex = 0;
        let rightIndex = 0;
while (leftIndex < left.length && rightIndex < right.length) {
            if (left[leftIndex].name < right[rightIndex].name) {
                result.push(left[leftIndex]);
                leftIndex++;
            } else {
                result.push(right[rightIndex]);
                rightIndex++;
            } }
    return result.concat(left.slice(leftIndex), right.slice(rightIndex));
    }
    function quickSort(arr) {
```

```
        if (arr.length <= 1) return arr;
  const pivot = arr[0];
        const left = [];
        const right = [];
   for (let i = 1; i < arr.length; i++) {
            if (arr[i].name < pivot.name) {
               left.push(arr[i]);
            } else {
               right.push(arr[i]);
            } }
return [...quickSort(left), pivot, ...quickSort(right)];
      }
 function sortWithMergeSort() {
          const start = performance.now();
          const sortedContacts = mergeSort(contacts);
          const end = performance.now();
          contacts = sortedContacts;
          displayContacts();
          const timeTaken = (end - start).toFixed(4);
document.getElementById('output').textContent = `Merge Sort took ${timeTaken}
milliseconds to sort the list.`;
      }
      function sortWithQuickSort() {
          const start = performance.now();
          const sortedContacts = quickSort(contacts);
          const end = performance.now();
          contacts = sortedContacts;
          displayContacts();
          const timeTaken = (end - start).toFixed(4);
```

```
        document.getElementById('output').textContent = `Quick Sort took ${timeTaken} milliseconds to sort the list.`;
    }
</script></body></html>
```





3: Event Ranking System (Heap Sort): Implement heap sort to rank participants in a large-scale competition based on their scores. Test your solution with large datasets.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Event Ranking System - Heap Sort</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      background-color: #f4f4f9;
      margin: 0;
      padding: 0;
      display: flex;
      justify-content: center;
      align-items: center;
      height: 100vh;}
    .container {
      text-align: center;
      width: 60%;  }
    h1 {
      margin-bottom: 20px;  }
 .participant-list {
      margin-top: 20px;
      list-style-type: none;
      padding: 0;
      display: inline-block;
      text-align: left;
```

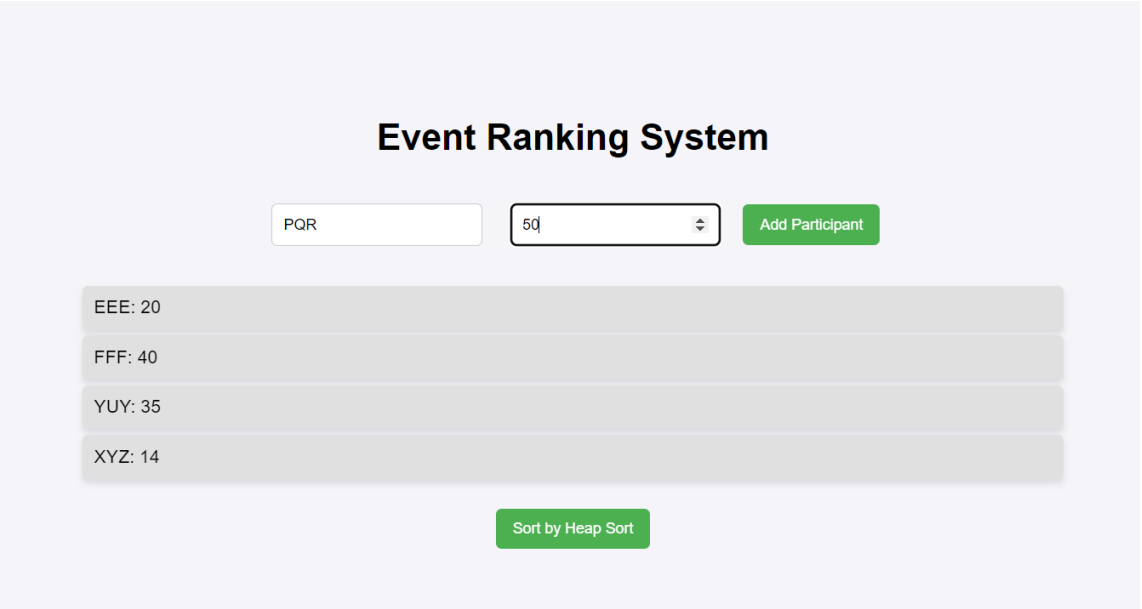
```
        width: 100%;       }
        .participant-list li {
        background-color: #e0e0e0;
        padding: 10px;
        margin: 5px 0;
        border-radius: 5px;
box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);  }
      .input-field {
        padding: 10px;
        margin: 10px;
        border-radius: 5px;
        border: 1px solid #ccc;}
      .form-container {
        margin-top: 30px;  }
      button {
        padding: 10px 15px;
        border: none;
        border-radius: 5px;
        background-color: #4CAF50;
        color: white;
        cursor: pointer;
        margin: 5px;
        transition: background-color 0.3s ease; }
      button:hover {
        background-color: #45a049;       }

      .output {
        margin-top: 20px;
        font-size: 18px;
```

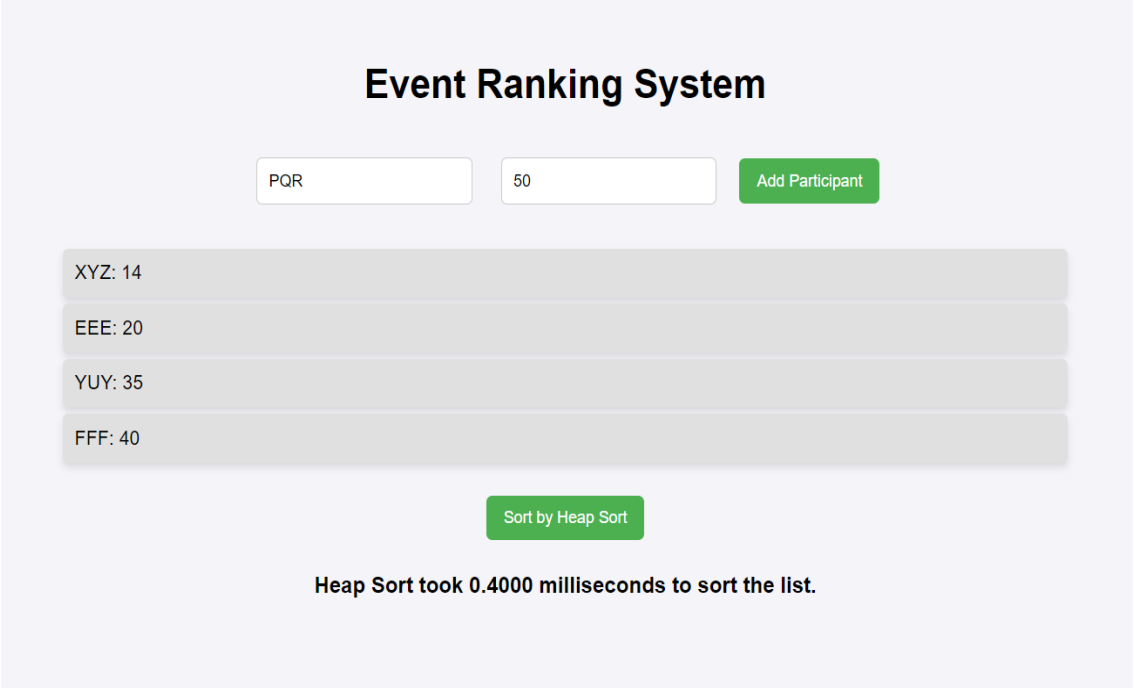
```
        font-weight: bold; }
    </style></head><body>
    <div class="container">
        <h1>Event Ranking System</h1>
        <div class="form-container">
            <input type="text" class="input-field" id="participantName" placeholder="Enter
participant name">
            <input type="number" class="input-field" id="participantScore" placeholder="Enter
participant score">
            <button onclick="addParticipant()">Add Participant</button>
        </div>
        <ul class="participant-list" id="participantList"></ul>
        <div>
            <button onclick="sortWithHeapSort()">Sort by Heap Sort</button>
        </div>
        <div class="output" id="output"></div>
    </div>
    <script>
        let participants = [];
        function addParticipant() {
            const name = document.getElementById('participantName').value;
            const score = parseInt(document.getElementById('participantScore').value);

            if (name && !isNaN(score)) {
                participants.push({ name, score });
                document.getElementById('participantName').value = '';
document.getElementById('participantScore').value = '';
    displayParticipants();
            } }
        function displayParticipants() {
```

```
        const list = document.getElementById('participantList');
        list.innerHTML = '';
        participants.forEach(participant => {
 const li = document.createElement('li'); li.textContent = `${participant.name}: ${participant.score}`;
   list.appendChild(li);
        });  }
      function heapSort(arr) {
        const n = arr.length;
 for (let i = Math.floor(n / 2) - 1; i >= 0; i--) {
   heapify(arr, n, i);
        }
        for (let i = n - 1; i > 0; i--) {
          // Move current root to end
          [arr[0], arr[i]] = [arr[i], arr[0]];
          heapify(arr, i, 0);
        }  }
      function heapify(arr, n, i) {
        let largest = i;
        const left = 2 * i + 1;
      const right = 2 * i + 2;
if (left < n && arr[left].score > arr[largest].score) {
          largest = left;
        } if (right < n && arr[right].score > arr[largest].score) {
          largest = right;
      }
        if (largest !== i) {
  [arr[i], arr[largest]] = [arr[largest], arr[i]];
        heapify(arr, n, largest); } }
```

```
        function sortWithHeapSort() {
            const start = performance.now();
            heapSort(participants);
            const end = performance.now();
            displayParticipants();
 const timeTaken = (end - start).toFixed(4);
document.getElementById('output').textContent = `Heap Sort took ${timeTaken} milliseconds to sort the list.`; }
  </script></body></html>
```

output:


Event Ranking System
PQR
50
Add Participant
EEE: 20
FFF: 40
YUY: 35
XYZ: 14
Sort by Heap Sort


Event Ranking System
PQR
50
Add Participant
XYZ: 14
EEE: 20
YUY: 35
FFF: 40
Sort by Heap Sort
Heap Sort took 0.4000 milliseconds to sort the list.

4: Efficient Storage using Hash Tables: Design a hash table to store and retrieve employee records based on employee IDs. Implement different hash functions and collision handling techniques (chaining, open addressing)

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>Event Ranking System - Heap Sort</title>
   <style>
      body {
   font-family: 'Arial', sans-serif;
   background-color: #f4f4f4;
   margin: 0;
   padding: 20px;
   display: flex;
   justify-content: center;
   align-items: center;
   height: 100vh;
}
.container {
   background-color: #fff;
   border-radius: 8px;
   box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
   padding: 30px;
   width: 100%;
   max-width: 600px;}
h2 {
   text-align: center;
   color: #333;
```

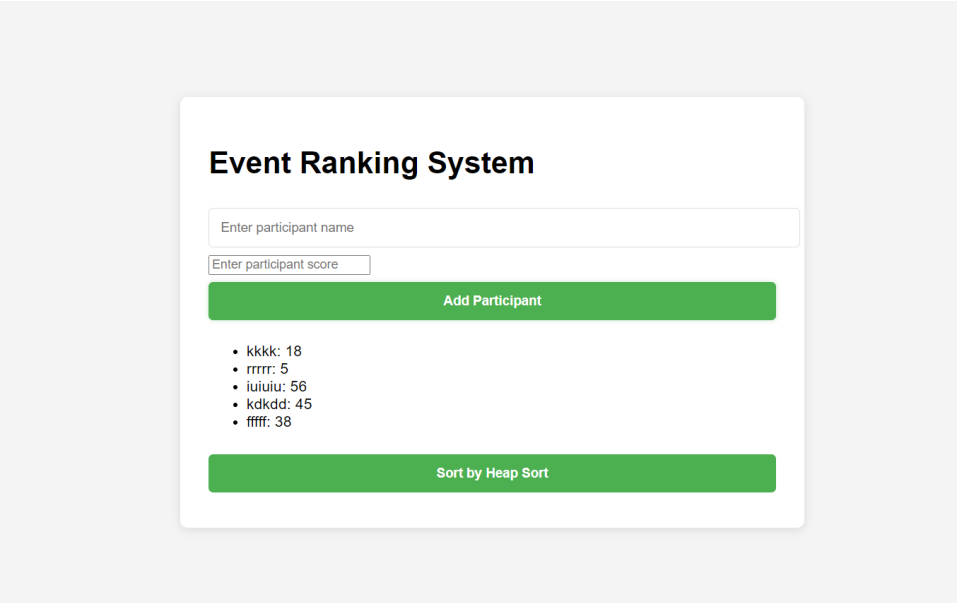
```
    margin-bottom: 20px;}
h3 {
    color: #555;
    margin-bottom: 10px;}
input[type="text"], button {
    width: 100%;
    padding: 12px;
    margin: 8px 0;
    border-radius: 5px;
    border: 1px solid #ddd;
    font-size: 14px;}
input[type="text"]:focus, button:focus {
    outline: none;
    border-color: #4CAF50;
    box-shadow: 0 0 5px rgba(76, 175, 80, 0.3);
}
button {
    background-color: #4CAF50;
    color: white;
    border: none;
    font-weight: bold;
    cursor: pointer;
    transition: background-color 0.3s ease;}

button:hover {
 background-color: #45a049;}
#recordList, #searchResult {
    margin-top: 20px;
    padding: 15px;
```

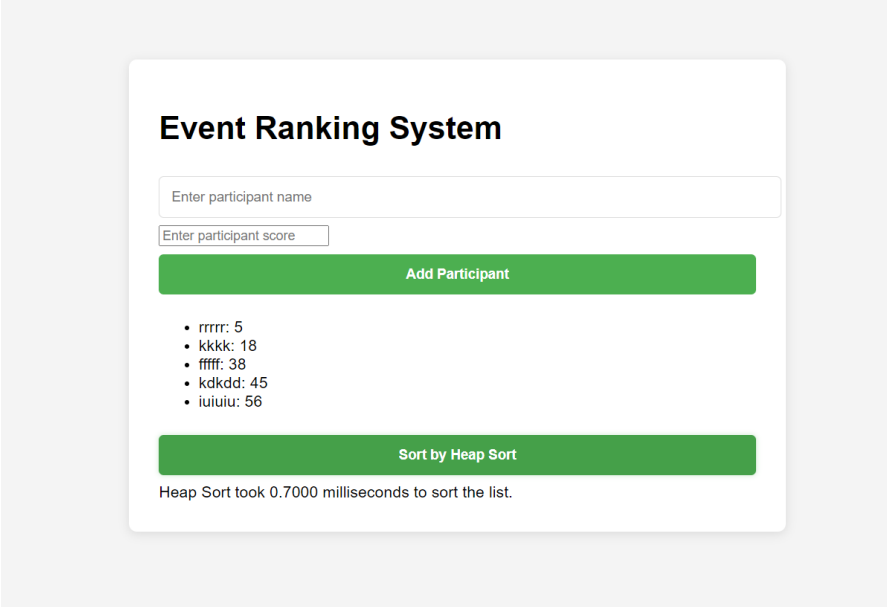
```
   border-radius: 5px;
   background-color: #f9f9f9;
   border: 1px solid #ddd;
}
#recordList div, #searchResult div {
   background-color: #fff;
   padding: 10px;
   margin: 5px 0;
   border-radius: 5px;
   box-shadow: 0 2px 5px rgba(0, 0, 0, 0.05);}
#searchResult p {
   color: #d9534f;
   font-size: 16px;
#recordList p {
   font-size: 14px;
   color: #333;}
#searchResult {
   display: none; }
h3, p {
   margin: 5px 0;
} </style></head><body>
   <div class="container">
      <h1>Event Ranking System</h1>

      <div class="form-container">
         <input type="text" class="input-field" id="participantName" placeholder="Enter participant name">
         <input type="number" class="input-field" id="participantScore" placeholder="Enter participant score">
         <button onclick="addParticipant()">Add Participant</button>
```

```
        </div>
        <ul class="participant-list" id="participantList"></ul>
        <div>
            <button onclick="sortWithHeapSort()">Sort by Heap Sort</button>
        </div>
        <div class="output" id="output"></div>
    </div>
    <script>
        let participants = [];
        function addParticipant() {
            const name = document.getElementById('participantName').value;
            const score = parseInt(document.getElementById('participantScore').value);
            if (name && !isNaN(score)) {
    participants.push({ name, score });
document.getElementById('participantName').value = '';
                document.getElementById('participantScore').value = '';
         displayParticipants();}}
        function displayParticipants() {
            const list = document.getElementById('participantList');
            list.innerHTML = '';
            participants.forEach(participant => {
                const li = document.createElement('li');
                li.textContent = `${participant.name}: ${participant.score}`;
                list.appendChild(li);
            }); }
        function heapSort(arr) {
            const n = arr.length;
 for (let i = Math.floor(n / 2) - 1; i >= 0; i--) {
                heapify(arr, n, i);  }
```

```
        for (let i = n - 1; i > 0; i--) {
            [arr[0], arr[i]] = [arr[i], arr[0]];
            heapify(arr, i, 0); }}
    function heapify(arr, n, i) {
        let largest = i;
        const left = 2 * i + 1;
        const right = 2 * i + 2;
if (left < n && arr[left].score > arr[largest].score) {
            largest = left;
        }
 if (right < n && arr[right].score > arr[largest].score) {
            largest = right; }
        if (largest !== i) {
 [arr[i], arr[largest]] = [arr[largest], arr[i]];
            heapify(arr, n, largest); }}
    function sortWithHeapSort() {
        const start = performance.now();
        heapSort(participants);
        const end = performance.now();
        displayParticipants();
        const timeTaken = (end - start).toFixed(4);
        document.getElementById('output').textContent = `Heap Sort took ${timeTaken}
milliseconds to sort the list.`;
  }</script></body></html>
```

Output:

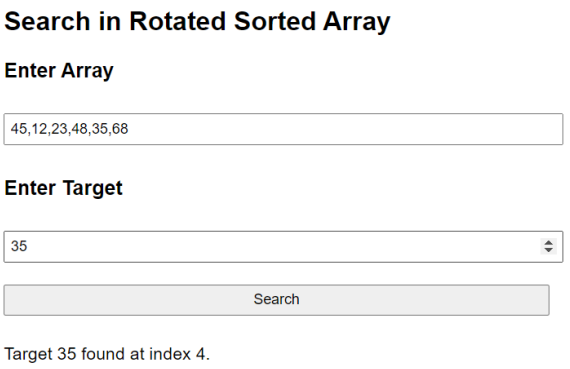

Event Ranking System
Enter participant name
Enter participant score
Add Participant
kkkk: 18
rrrrr: 5
iuiuiu: 56
kdkdd: 45
fffff: 38
Sort by Heap Sort


Event Ranking System
Enter participant name
Enter participant score
Add Participant
rrrrr: 5
kkkk: 18
fffff: 38
kdkdd: 45
iuiuiu: 56
Sort by Heap Sort
Heap Sort took 0.7000 milliseconds to sort the list.

5: Searching in a Rotated Sorted Array: Solve the problem of searching for a specific element in a rotated sorted array using a modified binary search algorithm.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <title>Rotated Sorted Array Search</title>
  <style>
    body {
      font-family: Arial;
      max-width: 500px;
      margin: 20px auto;
      padding: 20px;
    }
    input, button {
      width: 100%;
      margin: 10px 0;
      padding: 5px;
    }
  </style></head><body>
  <h2>Search in Rotated Sorted Array</h2>
  <h3>Enter Array</h3>
  <input type="text" id="arrayInput" placeholder="Enter numbers separated by commas">
  <h3>Enter Target</h3>
  <input type="number" id="targetInput" placeholder="Enter target number">
  <button onclick="searchRotatedArray()">Search</button>
  <div id="result"></div>
  <script>
    function searchRotatedArray() {
      const arrayInput = document.getElementById('arrayInput').value;
```

```
    const target = parseInt(document.getElementById('targetInput').value, 10);
    const resultDiv = document.getElementById('result');

    if (!arrayInput) {
      resultDiv.innerHTML = '<p>Please enter a valid array.</p>';
      return;
    }

    const arr = arrayInput.split(',').map(Number);
    const index = rotatedBinarySearch(arr, target);

    if (index !== -1) {
      resultDiv.innerHTML = `<p>Target ${target} found at index ${index}.</p>`;
    } else {
      resultDiv.innerHTML = `<p>Target ${target} not found in the array.</p>`;
    }
  }
  function rotatedBinarySearch(arr, target) {
    let left = 0;
    let right = arr.length - 1;

    while (left <= right) {
      const mid = Math.floor((left + right) / 2);
      if (arr[mid] === target) {
        return mid;  }
      if (arr[left] <= arr[mid]) {
if (target >= arr[left] && target < arr[mid]) {
          right = mid - 1;
        } else {
```



```
            left = mid + 1;
                }
              }
              else {
 if (target > arr[mid] && target <= arr[right]) {
                  left = mid + 1;
                } else {
                  right = mid - 1;
                } }}
          return -1;
      }
   </script></body></html>
```

Output:

6.Sorting a Music Library (Quick Sort): Implement quick sort to arrange songs in a music library by different parameters (duration, artist, genre). Optimize the algorithm for large datasets.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Music Library Sorter</title>
  <style>
    body {
      font-family: 'Arial', sans-serif;
      background-color: #f4f4f4;
      margin: 0;
      padding: 20px;
      display: flex;
      justify-content: center;
      align-items: flex-start;
      height: 100vh;
    }
    .container {
      background-color: #fff;
      border-radius: 8px;
      box-shadow: 0 4px 12px rgba(0, 0, 0, 0.1);
      padding: 40px;
      width: 100%;
      max-width: 800px;
    }
    h2 {
      text-align: center;
```

```
    color: #333;
    margin-bottom: 20px;
    font-size: 28px;
}
h3 {
    color: #444;
    margin-bottom: 10px;
    font-size: 18px;
}
input[type="text"], input[type="number"], select, button {
    width: 100%;
    padding: 12px;
    margin: 10px 0;
    border-radius: 5px;
    border: 1px solid #ddd;
    font-size: 16px;
}
input[type="text"]:focus, input[type="number"]:focus, select:focus, button:focus {
    outline: none;
    border-color: #4CAF50;
    box-shadow: 0 0 5px rgba(76, 175, 80, 0.3);
}
button {
    background-color: #4CAF50;
    color: white;
    border: none;
    font-weight: bold;
    cursor: pointer;
    transition: background-color 0.3s ease;
```

```
}
button:hover {
    background-color: #45a049;
}
#sortedLibrary {
    margin-top: 30px;
    padding: 20px;
    background-color: #fff;
    border-radius: 5px;
    border: 1px solid #ddd;
    box-shadow: 0 2px 10px rgba(0, 0, 0, 0.05);
}
#libraryList {
    list-style-type: none;
    padding: 0;
}
#libraryList li {
    padding: 12px;
    border-bottom: 1px solid #ddd;
    font-size: 16px;
    background-color: #fafafa;
    margin-bottom: 5px;
}

#libraryList li:last-child {
    border-bottom: none;
}
input::placeholder {
    color: #aaa;
```

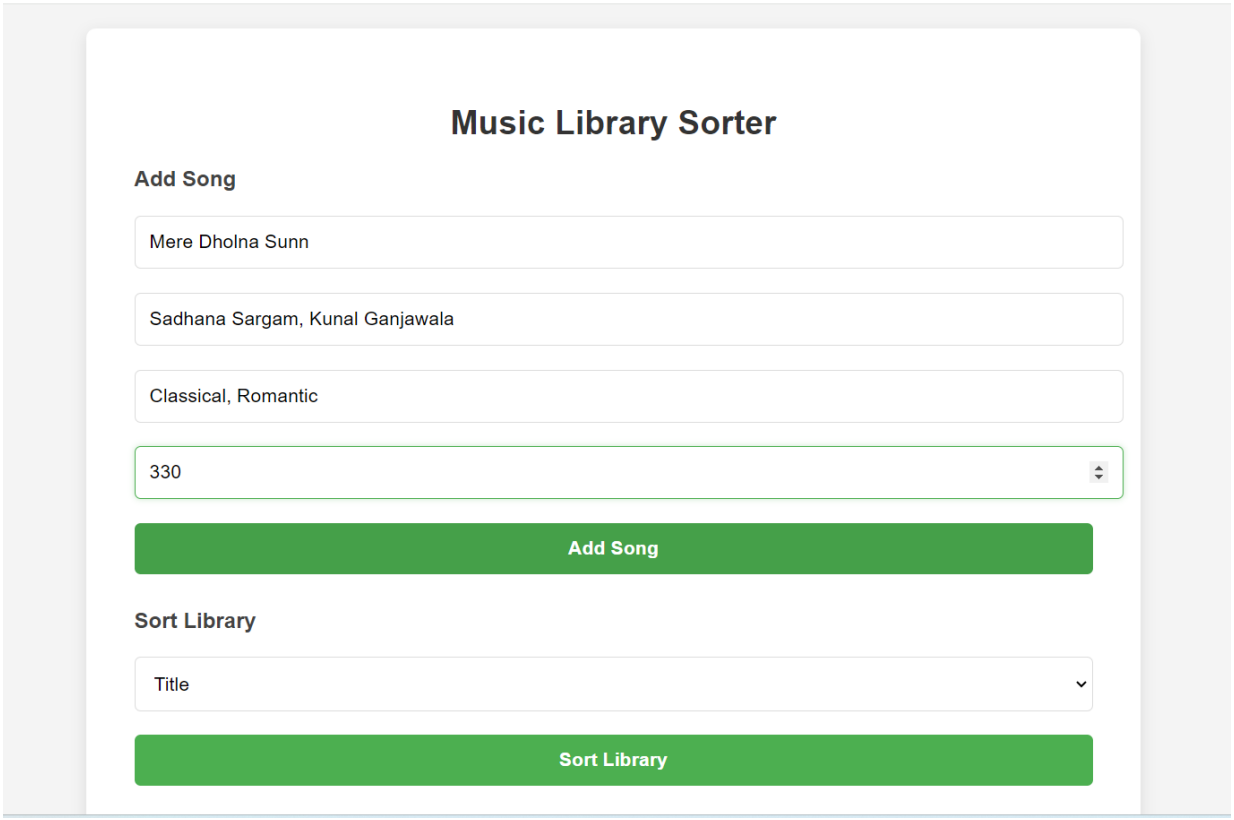
```
            font-style: italic;
        }
        @media (max-width: 768px) {
            .container {
                padding: 20px;
            }
            h2 {
                font-size: 24px;
            }
            h3 {
                font-size: 16px;
            }
            input[type="text"], input[type="number"], select, button {
                font-size: 14px;
            }
        }
    </style>
</head>
<body>
    <div class="container">
        <h2>Music Library Sorter</h2>

        <h3>Add Song</h3>
        <input type="text" id="songTitle" placeholder="Song Title">
        <input type="text" id="songArtist" placeholder="Artist">
        <input type="text" id="songGenre" placeholder="Genre">
        <input type="number" id="songDuration" placeholder="Duration (in seconds)">
        <button onclick="addSong()">Add Song</button>
        <h3>Sort Library</h3>
```

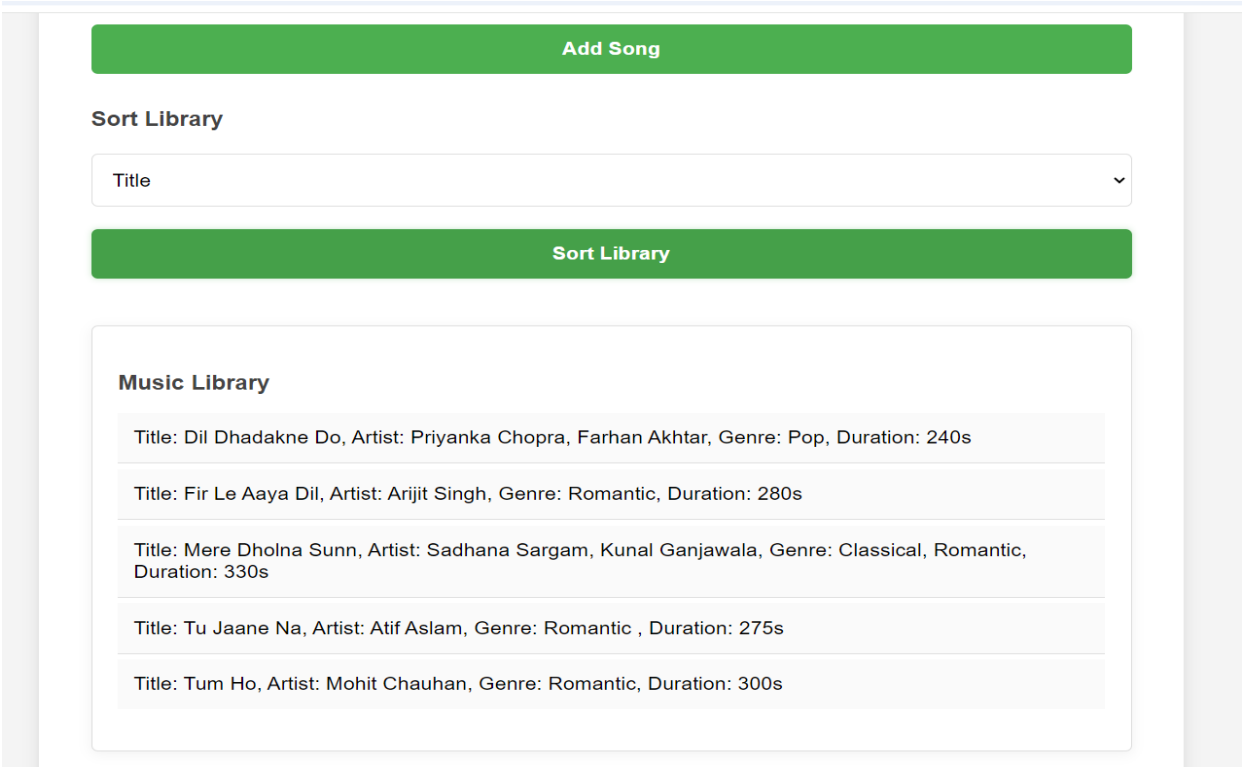
```
    <select id="sortParameter">
      <option value="title">Title</option>
      <option value="artist">Artist</option>
      <option value="genre">Genre</option>
      <option value="duration">Duration</option>
    </select>
    <button onclick="sortLibrary()">Sort Library</button>
    <div id="sortedLibrary">
      <h3>Music Library</h3>
      <ul id="libraryList"></ul>
    </div>
  </div>
  <script>
    const musicLibrary = [];
    function addSong() {
      const title = document.getElementById('songTitle').value;
      const artist = document.getElementById('songArtist').value;
      const genre = document.getElementById('songGenre').value;
      const duration = parseInt(document.getElementById('songDuration').value, 10);

      if (!title || !artist || !genre || isNaN(duration)) {
        alert('Please fill all fields correctly.');
        return;
      }
musicLibrary.push({ title, artist, genre, duration });
      displayLibrary();
      document.getElementById('songTitle').value = '';
      document.getElementById('songArtist').value = '';
      document.getElementById('songGenre').value = '';
```

```
        document.getElementById('songDuration').value = '';
    }

    function displayLibrary() {
        const libraryList = document.getElementById('libraryList');
        libraryList.innerHTML = '';

        musicLibrary.forEach(song => {
            const listItem = document.createElement('li');
            listItem.textContent = `Title: ${song.title}, Artist: ${song.artist}, Genre: ${song.genre}, Duration: ${song.duration}s`;
            libraryList.appendChild(listItem);
        });
    }
    function quickSort(array, key) {
        if (array.length <= 1) {
            return array;
        }
        const pivotIndex = Math.floor(Math.random() * array.length);
        const pivot = array[pivotIndex];
        const less = [];
        const greater = [];

        for (let i = 0; i < array.length; i++) {
            if (i === pivotIndex) continue;
            if (array[i][key] < pivot[key]) {
                less.push(array[i]);
            } else {
                greater.push(array[i]);
```

```
            }
        }
        return [...quickSort(less, key), pivot, ...quickSort(greater, key)];
    }
    function sortLibrary() {
        const sortParameter = document.getElementById('sortParameter').value;
        const sortedLibrary = quickSort(musicLibrary, sortParameter);
        musicLibrary.splice(0, musicLibrary.length, ...sortedLibrary); // Update original library
        displayLibrary();
    }
  </script>
</body>
</html>
```

Output:


Music Library Sorter
Add Song
Mere Dholna Sunn
Sadhana Sargam, Kunal Ganjawala
Classical, Romantic
330
Add Song
Sort Library
Title
Sort Library


Add Song
Sort Library
Title
Sort Library
Music Library
Title: Dil Dhadakne Do, Artist: Priyanka Chopra, Farhan Akhtar, Genre: Pop, Duration: 240s
Title: Fir Le Aaya Dil, Artist: Arijit Singh, Genre: Romantic, Duration: 280s
Title: Mere Dholna Sunn, Artist: Sadhana Sargam, Kunal Ganjawala, Genre: Classical, Romantic, Duration: 330s
Title: Tu Jaane Na, Artist: Atif Aslam, Genre: Romantic , Duration: 275s
Title: Tum Ho, Artist: Mohit Chauhan, Genre: Romantic, Duration: 300s

7: Caching using LRU Cache: Implement an LRU (Least Recently Used) Cache system using a combination of hash maps and doubly linked lists to store frequently accessed data efficiently.

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>LRU Cache</title>
   <style>
      body {
         font-family: Arial, sans-serif;
         padding: 20px;
      }
      .cache-output {
         margin-top: 20px;
         padding: 10px;
         background-color: #f4f4f4;
         border: 1px solid #ddd;
      }
      button {
         padding: 8px;
         margin-top: 10px;
         cursor: pointer;
      }
   </style></head><body>
   <h2>LRU Cache Example</h2>
   <label for="cacheSize">Enter Cache Size: </label>
   <input type="number" id="cacheSize" placeholder="Cache Size" />
   <button onclick="initializeLRU()">Initialize Cache</button>
```

```
<div>
    <h3>Perform Cache Operations</h3>
    <input type="text" id="key" placeholder="Enter Key" />
    <input type="text" id="value" placeholder="Enter Value" />
    <button onclick="put()">Put Data in Cache</button>
    <button onclick="get()">Get Data from Cache</button>
</div>
<div class="cache-output">
    <h4>Cache Output:</h4>
    <pre id="cacheOutput"></pre>
</div> <script>
    class Node {
        constructor(key, value) {
            this.key = key;
            this.value = value;
            this.prev = null;
            this.next = null;
        }
    }
    class LRUCache {
        constructor(capacity) {
            this.capacity = capacity;
            this.cache = new Map();
            this.head = new Node(null, null);
            this.tail = new Node(null, null);
            this.head.next = this.tail;
            this.tail.prev = this.head;
        }
        moveToFront(node) {
```

```
        this.remove(node);
        this.add(node);
    }
    add(node) {
        node.next = this.head.next;
        node.prev = this.head;
        this.head.next.prev = node;
        this.head.next = node;
    }
    remove(node) {
        const prev = node.prev;
        const next = node.next;
        prev.next = next;
        next.prev = prev;
    }
    get(key) {
        if (!this.cache.has(key)) return -1;
        const node = this.cache.get(key);
        this.moveToFront(node);
        return node.value;
    }
    put(key, value) {
        if (this.cache.has(key)) {
            const node = this.cache.get(key);
            node.value = value;
            this.moveToFront(node);
        } else {
            if (this.cache.size >= this.capacity) {
const lruNode = this.tail.prev;
```

```
            this.cache.delete(lruNode.key);
            this.remove(lruNode);
        }
        const newNode = new Node(key, value);
        this.cache.set(key, newNode);
        this.add(newNode);
      }
    }
}
let lruCache;
function initializeLRU() {
    const cacheSize = document.getElementById('cacheSize').value;
    if (cacheSize && !isNaN(cacheSize) && cacheSize > 0) {
        lruCache = new LRUCache(Number(cacheSize));
        displayCache();
    } else {
        alert('Please enter a valid cache size.');
    }
}
function put() {
    const key = document.getElementById('key').value;
    const value = document.getElementById('value').value;
    if (lruCache && key && value) {
        lruCache.put(key, value);
        displayCache();
    } else {
        alert('Please initialize cache and enter valid data.');
    }
}
```

```
        function get() {
            const key = document.getElementById('key').value;
            if (lruCache && key) {
                const result = lruCache.get(key);
                const output = result !== -1 ? `Value: ${result}` : 'Not found in cache';
                document.getElementById('cacheOutput').textContent = `Get Result: ${output}`;
            } else {
                alert('Please initialize cache and enter a valid key.');
            }
        }
        function displayCache() {
            if (lruCache) {
                const output = [];
                let current = lruCache.head.next;
                while (current !== lruCache.tail) {
                    output.push(`Key: ${current.key}, Value: ${current.value}`);
                    current = current.next;   }
document.getElementById('cacheOutput').textContent = `Cache: \n${output.join('\n')}`;   } }
    </script></body></html>
```

Output:

LRU Cache Example

Enter Cache Size: 436 Initialize Cache

Perform Cache Operations

DERT 56 Put Data in Cache Get Data from Cache

Cache Output:

```
Get Result: Value: 56
```

LRU Cache Example

Enter Cache Size: 436 Initialize Cache

Perform Cache Operations

DERT 56 Put Data in Cache Get Data from Cache

Cache Output:

```
Cache:
Key: DERT, Value: 56
Key: RTTR, Value: 35
Key: FFFF, Value: 45
Key: TTTT, Value: 2
```

8.Dictionary Implementation with Hashing: Create a dictionary where words are stored using a hash table. Implement efficient lookup, insertion, and deletion operations using custom hash functions.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Dictionary with Hashing</title>
  <style>
body {
  font-family: Arial, sans-serif;
  background-color: #f4f4f4;
  color: #333;
  margin: 0;
  padding: 20px;
}
h2 {
  text-align: center;
  color: #007BFF;
  font-size: 24px;
}
input, button {
  padding: 12px;
  font-size: 16px;
  border: 1px solid #ddd;
  border-radius: 5px;
  margin: 10px 0;
  width: 100%;
  box-sizing: border-box;
```

```
}
button:hover {
    background-color: #007BFF;
    color: white;
    cursor: pointer;
}
div {
    margin-top: 20px;
    max-width: 600px;
    margin: 0 auto;
}
#dictionaryList {
    margin-top: 30px;
    padding: 20px;
    background-color: white;
    border: 1px solid #ddd;
    border-radius: 5px;
    box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
}
li {
    padding: 10px;
    border-bottom: 1px solid #ddd;
    list-style-type: none;
}
li:last-child {
    border-bottom: none;
}
.alert {
    padding: 15px;
```

```
    margin: 20px 0;
    background-color: #f44336;
    color: white;
    border-radius: 5px;
}

.alert.success {
    background-color: #4CAF50;
}

.alert.warning {
    background-color: #ff9800;
}
@media (max-width: 768px) {
    body {
        padding: 10px;
    }
    div {
        max-width: 100%;
    }
}
    </style></head><body>
    <h2>Dictionary with Hashing</h2>

    <input type="text" id="wordInput" placeholder="Enter Word">
    <input type="text" id="meaningInput" placeholder="Enter Meaning">
    <button onclick="addWord()">Add Word</button>

    <input type="text" id="searchWord" placeholder="Enter Word to Search">
```

```
    <button onclick="searchWord()">Search</button>

    <input type="text" id="deleteWord" placeholder="Enter Word to Delete">
    <button onclick="deleteWord()">Delete</button>

    <ul id="dictionaryList"></ul>

    <script>
        class HashTable {
            constructor(size = 20) {
                this.table = Array.from({ length: size }, () => []);
            }

            hash(word) {
                return Array.from(word).reduce((acc, char) => acc + char.charCodeAt(0), 0) %
this.table.length;
            }

            insert(word, meaning) {
                const bucket = this.table[this.hash(word)];
                const existing = bucket.find(entry => entry.word === word);
                existing ? existing.meaning = meaning : bucket.push({ word, meaning });
                this.render();
            }
            search(word) {
                const bucket = this.table[this.hash(word)];
const entry = bucket.find(entry => entry.word === word);
                return entry ? entry.meaning : 'Not found';
            }
```

```
    delete(word) {
        const bucket = this.table[this.hash(word)];
        const index = bucket.findIndex(entry => entry.word === word);
        index !== -1 && bucket.splice(index, 1);
        this.render();
    }
    render() {
        const list = document.getElementById('dictionaryList');
        list.innerHTML = this.table.flatMap(bucket => bucket.map(entry =>
            `<li>Word: ${entry.word}, Meaning: ${entry.meaning}</li>`)).join('');
    }
}
const dictionary = new HashTable();
function addWord() {
    const word = document.getElementById('wordInput').value.trim();
    const meaning = document.getElementById('meaningInput').value.trim();
    if (word && meaning) {
        dictionary.insert(word, meaning);
        document.getElementById('wordInput').value = '';
        document.getElementById('meaningInput').value = '';
    } else {
        alert('Please enter both word and meaning!');
    }
}
function searchWord() {
    const word = document.getElementById('searchWord').value.trim();
    if (word) {
        alert(`Meaning of "${word}": ${dictionary.search(word)}`);
        document.getElementById('searchWord').value = '';
```

```
        } else {
            alert('Please enter a word to search!');
        }
    }
    function deleteWord() {
        const word = document.getElementById('deleteWord').value.trim();
        if (word) {
            dictionary.delete(word);
            alert(`"${word}" has been deleted from the dictionary.`);
            document.getElementById('deleteWord').value = '';
        } else {
            alert('Please enter a word to delete!');
        }
    }
  </script>
</body>
</html>
```

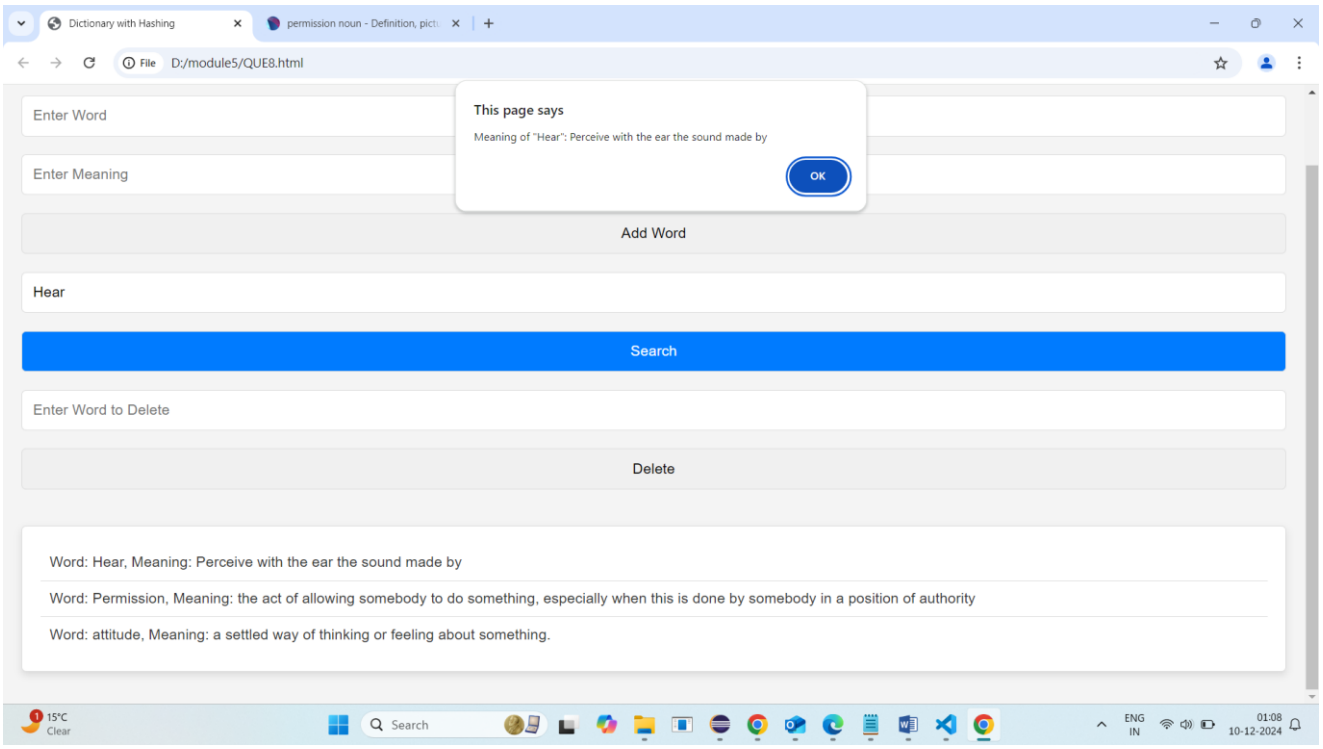
Dictionary with Hashing

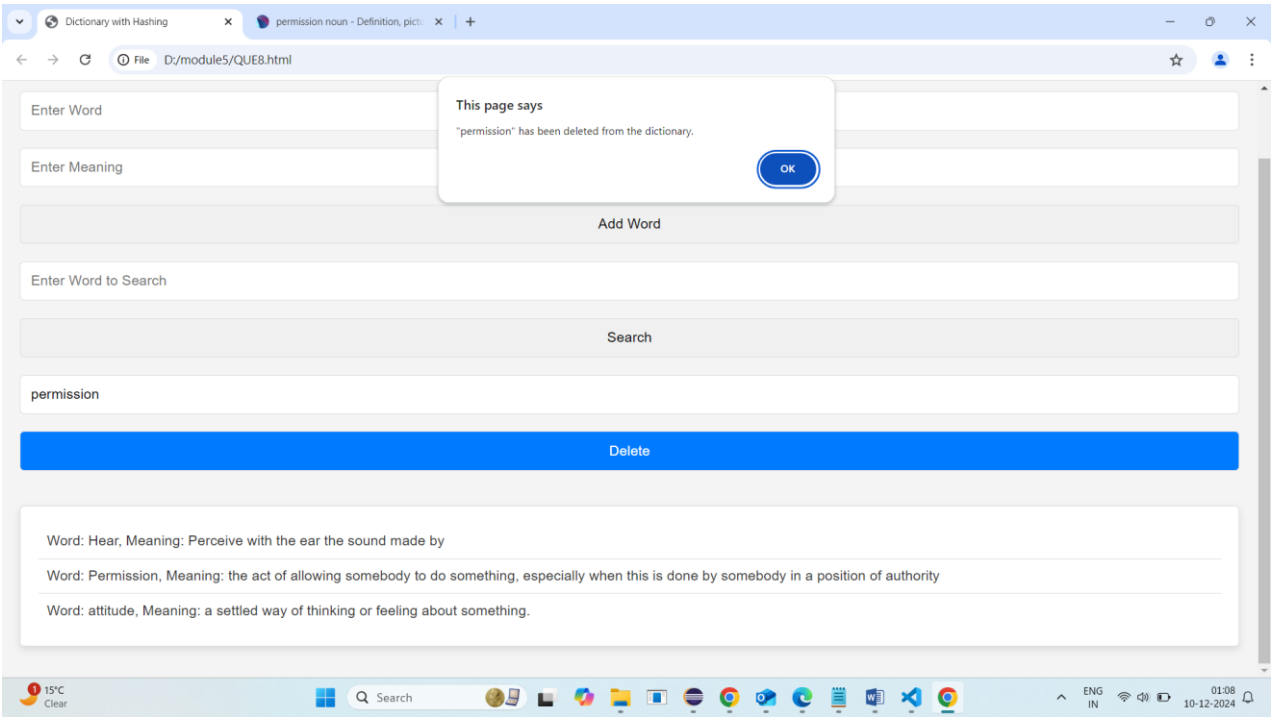
attitude

a settled way of thinking or feeling about something.

Add Word

Enter Word to Search





9: Inventory Search using Interpolation Search: Implement an interpolation search algorithm for finding items in an inventory management system where the data distribution is uniform. Compare its performance with binary and linear search algorithms.

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Inventory Search System</title>

    <style> body { font-family: Arial, sans-serif; max-width: 600px; margin: 20px auto; padding:

20px; } input, button { margin: 10px 0; padding: 8px; width: 100%; } table {
        width: 100%; border-collapse: collapse; margin-top: 20px; } th, td {
        border: 1px solid #ddd; padding: 8px; text-align: left; } th { background-
        color: #f4f4f4; }

    </style>
</head>
<body>

    <h2>Inventory Search System</h2>

    <input type="text" id="itemName" placeholder="Item Name">

    <input type="number" id="itemPrice" placeholder="Item Price">

    <button onclick="addItem()">Add Item</button>

    <input type="text" id="searchName" placeholder="Enter Name to Search">

    <button onclick="searchInventory()">Search</button>

    <table id="inventoryTable">

        <thead>

            <tr><th>Item Name</th><th>Item Price</th></tr>

        </thead>
        <tbody></tbody>
    </table>

    <div id="searchResult"></div>


    <script> let
        inventory = [];


        function addItem() {
```

```
    const itemName = document.getElementById('itemName').value.trim(); const itemPrice =
    parseFloat(document.getElementById('itemPrice').value.trim()); if (!itemName ||
    isNaN(itemPrice)) return alert('Please enter a valid name and price.');

    inventory.push({ name: itemName, price: itemPrice });
    inventory.sort((a, b) => a.name.localeCompare(b.name));
    displayInventory(); document.getElementById('itemName').value = '';
    document.getElementById('itemPrice').value = '';

}

function displayInventory() { document.querySelector('#inventoryTable
    tbody').innerHTML =

        inventory.map(item =>

`<tr><td>${item.name}</td><td>${item.price}</td></tr>`).join('');
}

function interpolationSearch(arr, key) {

    let low = 0, high = arr.length - 1; while (low <= high && key >= arr[low].name
    && key <= arr[high].name) {

        if (low === high) return arr[low].name === key ? low : -1; let pos = low +
        Math.floor(((key.charCodeAt(0) - arr[low].name.charCodeAt(0)) *

(high - low)) / (arr[high].name.charCodeAt(0) - arr[low].name.charCodeAt(0))); if
        (arr[pos].name === key) return pos; if (arr[pos].name < key) low = pos +
        1; else high = pos - 1;

    }

    return -1;

}

function binarySearch(arr, key) {

    let low = 0, high = arr.length - 1; while
    (low <= high) {

        const mid = Math.floor((low + high) / 2); if
        (arr[mid].name === key) return mid; if
        (arr[mid].name < key) low = mid + 1; else
        high = mid - 1;

    }

    return -1;

}
```

```
        function linearSearch(arr, key) {
            for (let i = 0; i < arr.length; i++) {
                if (arr[i].name === key) return i;
            }
            return -1;
        }

        function searchInventory() {
            const searchName = document.getElementById('searchName').value.trim(); if
            (!searchName) return alert('Please enter a valid name.');

            const start = performance.now(); const index =
            interpolationSearch(inventory, searchName); const duration =
            performance.now() - start;
            const startBinary = performance.now(); const binaryIndex =
            binarySearch(inventory, searchName); const durationBinary =
            performance.now() - startBinary;

            const startLinear = performance.now(); const linearIndex =
            linearSearch(inventory, searchName); const durationLinear =
            performance.now() - startLinear;

            document.getElementById('searchResult').innerHTML = index !== -
                1
                    ? `<p>Item Found: <strong>${inventory[index].name}</strong> with Price:
<strong>${inventory[index].price}</strong></p>`
                    : `<p>Item not found in the inventory.</p>`;

            document.getElementById('searchResult').innerHTML += `
                <p>Interpolation Search Time: ${duration.toFixed(4)} ms</p>
                <p>Binary Search Time: ${durationBinary.toFixed(4)} ms</p>
                <p>Linear Search Time: ${durationLinear.toFixed(4)} ms</p>`;
        }
    </script>
</body>
</html>
```

Inventory Search System

Mouse

5000

Add Item

Enter Name to Search

Search

Item Name	Item Price
Earphones	2000
Laptop	50000
Mobile	10000
Tablet	30000

Item Name

Item Price

Add Item

Laptop

Search

Item Name	Item Price
Earphones	2000
Laptop	50000
Mobile	10000
Mouse	5000
Tablet	30000

Item Found: **Laptop** with Price: **50000**

Interpolation Search Time: 0.8000 ms

Binary Search Time: 0.2000 ms

Linear Search Time: 0.0000 ms

10: Sorting Patient Data in a Hospital: Design an algorithm to sort patient data based on emergency levels using heap sort. Ensure that the sorting happens in real-time for critical situations in an emergency room.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Emergency Room Patient Sorting</title>
    <style> body { font-family: Arial, sans-serif; max-width: 600px; margin: 20px auto; padding:
20px; } input, button { margin: 10px 0; padding: 8px; width: 100%; } table {
        width: 100%; border-collapse: collapse; margin-top: 20px; } th, td {
        border: 1px solid #ddd; padding: 8px; text-align: left; } th { background-
        color: #f4f4f4; }
    </style>
</head>
<body>
    <h2>Emergency Room Patient Sorting</h2>
    <input type="text" id="patientName" placeholder="Patient Name">
    <input type="number" id="emergencyLevel" placeholder="Emergency Level (1-10)">
    <button onclick="addPatient()">Add Patient</button>
    <table id="patientTable">
        <thead>
            <tr><th>Patient Name</th><th>Emergency Level</th></tr>
        </thead>
        <tbody></tbody>
    </table>
    <script> let
        patients = [];

        function addPatient() {
            const name = document.getElementById('patientName').value.trim(); const level =
parseInt(document.getElementById('emergencyLevel').value.trim()); if (!name || isNaN(level))
return alert('Please enter a valid name and emergency level.');
```

```
        patients.push({ name, level }); heapSort(patients);
        displayPatients();
        document.getElementById('patientName').value = '';
        document.getElementById('emergencyLevel').value = ''; }

    function displayPatients() { document.querySelector('#patientTable
        tbody').innerHTML = patients.map(patient =>
`<tr><td>${patient.name}</td><td>${patient.level}</td></tr>`).join('');
    }

    function heapify(arr, n, i) {
        let largest = i; const
        left = 2 * i + 1; const
        right = 2 * i + 2;

        if (left < n && arr[left].level > arr[largest].level) largest = left; if (right
        < n && arr[right].level > arr[largest].level) largest = right;

        if (largest !== i) {
            [arr[i], arr[largest]] = [arr[largest], arr[i]]; heapify(arr,
            n, largest);
        }
    }

    function heapSort(arr) {
        const n = arr.length; for (let i = Math.floor(n / 2) - 1; i >= 0; i--)
        heapify(arr, n, i); for (let i = n - 1; i > 0; i--) {
            [arr[0], arr[i]] = [arr[i], arr[0]]; heapify(arr,
            i, 0);
        }
    }
  </script>
</body>
</html>
```

Emergency Room Patient Sorting

Vijay

2

Add Patient

Patient Name	Emergency Level
Priya	0
Raj	1
Rajesh	3

Emergency Room Patient Sorting

Patient Name

Emergency Level (1-10)

Add Patient

Patient Name	Emergency Level
Priya	0
Raj	1
Vijay	2
Rajesh	3

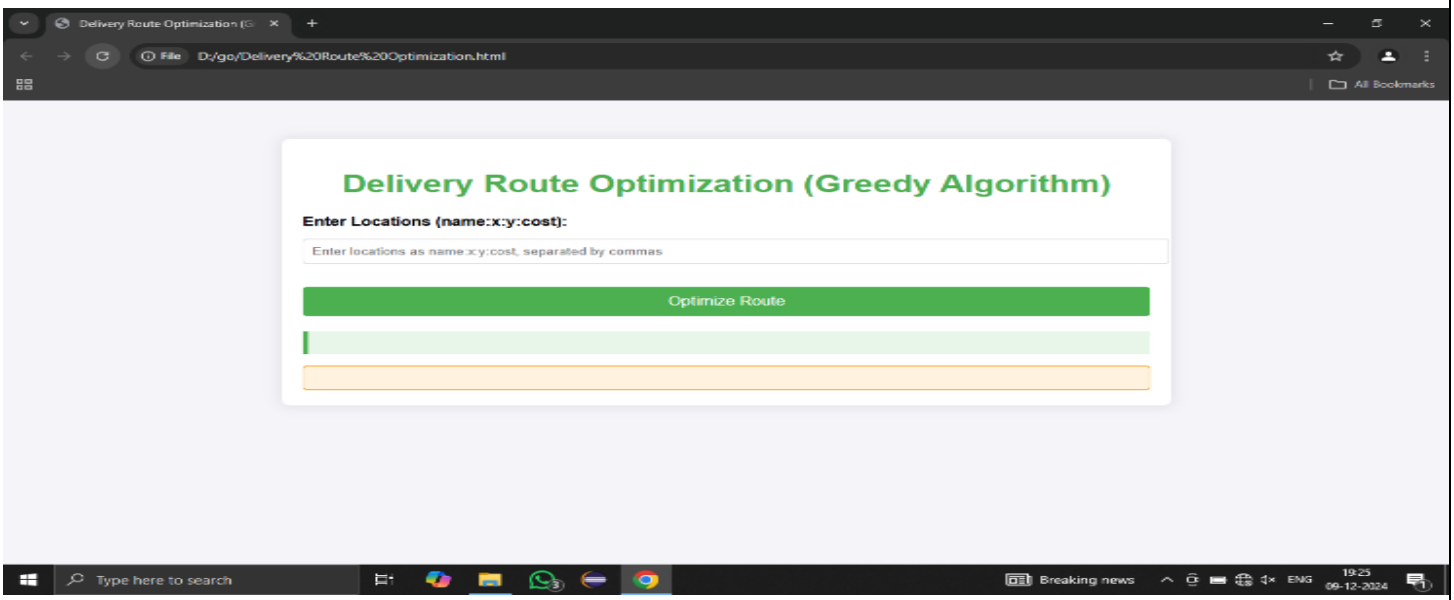
MODULE 6

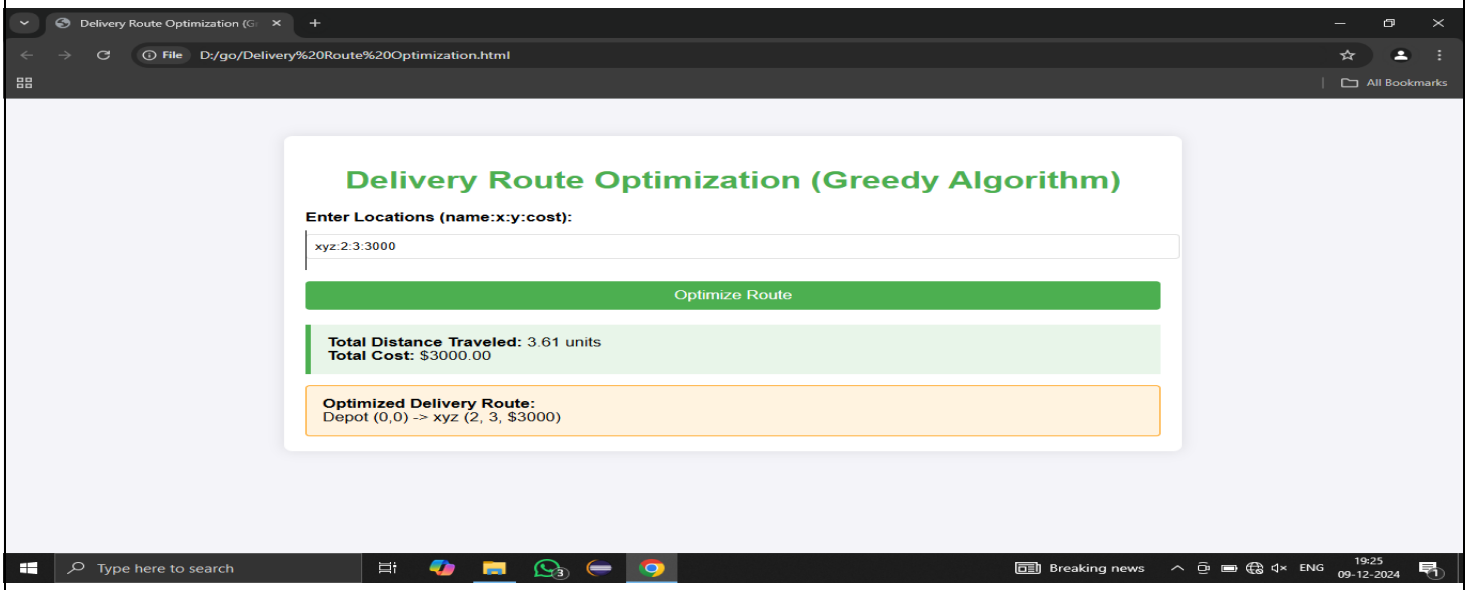
1: DELIVERY ROUTE OPTIMIZATION (GREEDY ALGORITHM): SOLVE THE DELIVERY ROUTE OPTIMIZATION PROBLEM FOR A DELIVERY SERVICE USING A GREEDY ALGORITHM. MINIMIZE THE TOTAL DISTANCE TRAVELED BY THE DELIVERY DRIVERTO DELIVER PACKAGES TO MULTIPLE DESTINATIONS.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport"
content="width=device-width, initial-scale=1.0">
  <title>Delivery Route Optimization (Greedy
Algorithm)</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      background-color: #f4f4f9;
      margin: 0;
      padding: 0;
    }
    .container {
      max-width: 800px;
      margin: 50px auto;
      background-color: white;
      padding: 20px;
      border-radius: 8px;
      box-shadow: 0 0 15px rgba(0, 0, 0, 0.1);
    }
    h1 {
      text-align: center;
      color: #4CAF50;
    }
    .form-group
      {
      margin: 15px 0;
    }
    label {
      font-weight: bold;
      display: block;
      margin-bottom: 8px;
    }
    input[type="text"], input[type="number"] {
      width: 100%;
      padding: 8px;
      margin: 5px 0 15px 0;
      border: 1px solid #ddd;
      border-radius: 4px;
    }
    button {
      width: 100%;
      padding: 10px;
      background-color: #4CAF50;
      color: white;
      border: none;
      border-radius: 4px;
      font-size: 16px;
    }
    button:hover {
      background-color: #45a049;
    }
    .result {
      margin-top: 20px
```

```
 padding: 15px;
            background-color: #e8f5e9;
            border-left: 5px solid #4CAF50;
        }
        .route {
            margin-top: 15px;
            padding: 15px;
            background-color: #fff3e0;
            border: 1px solid #ff9800;
            border-radius: 4px;
        }
    </style>
</head>
<body>

<div class="container">
    <h1>Delivery Route Optimization (Greedy Algorithm)</h1>

    <div class="form-group">
        <label for="locations">Enter Locations (name:x:y:cost):</label>
        <input type="text" id="locations" placeholder="Enter locations as name:x:y:cost, separated by commas">
    </div>
 <button onclick="optimizeRoute()">Optimize Route</button>
  <div class="result" id="result"></div>
    <div class="route" id="route"></div>
</div>

<script>
    function optimizeRoute() {
        // Get the input locations
        const locationsInput = document.getElementById('locations').value;

        // Parse the input string into an array of objects with name, x, y, and cost
        const locations = locationsInput.split(',').map(location => {
            const [name, x, y, cost] = location.trim().split(':');
            return { name, x: Number(x), y: Number(y), cost: Number(cost) };
        });

        // If there are no valid locations or input is invalid
        if (locations.length === 0 || locations.some(loc => isNaN(loc.x) || isNaN(loc.y) || isNaN(loc.cost))) {
document.getElementById('result').innerHTML = 'Please enter valid locations in the format name:x:y:cost.';
            return;
        }
  // Function to calculate distance between two points
        function calculateDistance(point1, point2) {
            return Math.sqrt(Math.pow(point2.x - point1.x, 2) + Math.pow(point2.y - point1.y, 2));
        }
// Greedy algorithm to find the shortest route
        let currentLocation = { x: 0, y: 0 };  // Start from the depot (0,0)
        let remainingLocations = [...locations];
        let totalDistance = 0;
        let totalCost = 0;
        let route = ['Depot (0,0)'];
 // Iterate to find the nearest unvisited location until all locations are visited
```

```
        while (remainingLocations.length > 0) {
            let nearestLocation = null;
            let shortestDistance = Infinity;
            let nearestIndex = -1;
        // Find the nearest unvisited location
            for (let i = 0; i < remainingLocations.length; i++) {
                const dist = calculateDistance(currentLocation, remainingLocations[i]);
                if (dist < shortestDistance) {
                    shortestDistance = dist;
                    nearestLocation = remainingLocations[i];
                    nearestIndex = i;
                }
            }
        // Update the route, total distance, and total cost
            totalDistance += shortestDistance;
            totalCost += nearestLocation.cost;
            route.push(`${nearestLocation.name} (${nearestLocation.x}, ${nearestLocation.y}, $${nearestLocation.cost})`);
currentLocation = nearestLocation;
            // Remove the visited location from the remaining list
            remainingLocations.splice(nearestIndex, 1);
        }
    // Display the result

document.getElementById('result').innerHTML = `
            <strong>Total Distance Traveled:</strong> ${totalDistance.toFixed(2)} units <br>
            <strong>Total Cost:</strong> $${totalCost.toFixed(2)}
        `;
        document.getElementById('route').innerHTML = `
            <strong>Optimized Delivery Route:</strong><br>
            ${route.join(' -> ')}
        `;
    }
</script>

</body>
</html>
```

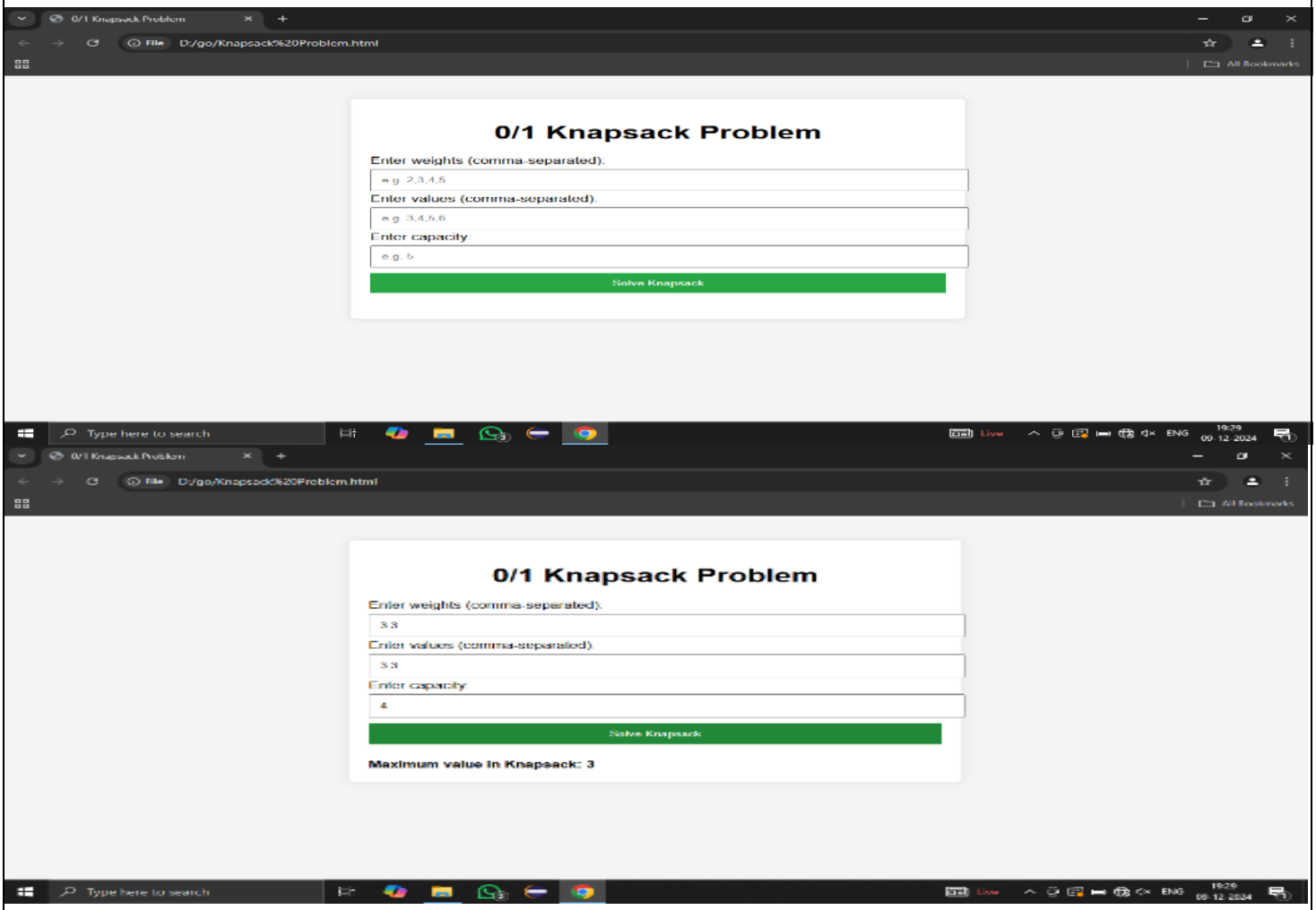




2: KNAPSACK PROBLEM (DYNAMIC PROGRAMMING): SOLVE THE 0/1 KNAPSACK PROBLEM USING DYNAMICPROGRAMMING, WHERE YOU ARE GIVEN A SET OF ITEMS, EACH WITH A WEIGHT AND VALUE, AND MUST DETERMINE THE MOST VALUABLE COMBINATION THAT CAN FIT WITHIN A WEIGHT LIMIT

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport"
content="width=device-width, initial-scale=1.0">
  <title>0/1 Knapsack Problem</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      margin: 20px;
      padding: 20px;
      background-color: #f4f4f4;
    }
    h1 {
      text-align: center;
    }
    .container {
      margin: auto;
      background: white;
      padding: 20px;
      border-radius: 5px;
      box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    }
    input, button {
      width: 100%;
      padding: 10px;
      margin: 5px 0;

}

 button {
      background-color: #28a745;
      color: white;
      border: none;
      cursor: pointer;
    }
```

```
        button:hover {
            background-color: #218838;
        }
        .result {
            margin-top: 20px;
            font-weight: bold;
        }
    </style>
</head>
<body>

<div class="container">
    <h1>0/1 Knapsack Problem</h1>
    <label for="weights">Enter weights (comma-separated):</label>
    <input type="text" id="weights" placeholder="e.g. 2,3,4,5">

    <label for="values">Enter values (comma-separated):</label>
    <input type="text" id="values" placeholder="e.g. 3,4,5,6">

    <label for="capacity">Enter capacity:</label>
    <input type="number" id="capacity" placeholder="e.g. 5">
    <button onclick="solveKnapsack()">Solve Knapsack</button>
    <div class="result" id="result"></div>
</div>

<script>
    function solveKnapsack() {
        const weightsInput = document.getElementById('weights').value;
        const valuesInput = document.getElementById('values').value;
        const capacity = parseInt(document.getElementById('capacity').value);
        const weights = weightsInput.split(',').map(Number);
        const values = valuesInput.split(',').map(Number);
        const n = weights.length;

        // Create a 2D array to store the maximum value at each n and capacity
        const dp = Array(n + 1).fill(0).map(() => Array(capacity + 1).fill(0))// Build the dp array
        for (let i = 1; i <= n; i++) {
            for (let w = 0; w <= capacity; w++) {
                if (weights[i - 1] <= w) {
                    dp[i][w] = Math.max(dp[i - 1][w], dp[i - 1][w - weights[i - 1]] + values[i - 1]);
                } else {
                    dp[i][w] = dp[i - 1][w];
                }
            }
        }
// The maximum value is in the bottom-right corner of the dp array
        const maxValue = dp[n][capacity];
        document.getElementById('result').innerText = `Maximum value in Knapsack: ${maxValue}`;
    }
</script>
</body>
</html>
```



3: DIVIDE AND CONQUER APPROACH FOR MATRIX MULTIPLICATION: IMPLEMENT A DIVIDE AND CONQUER ALGORITHM (STRASSEN'S ALGORITHM) FOR MATRIX MULTIPLICATION. COMPARE ITS PERFORMANCE WITH THE STANDARDMATRIX MULTIPLICATION ALGORITHM FOR LARGE MATRICES

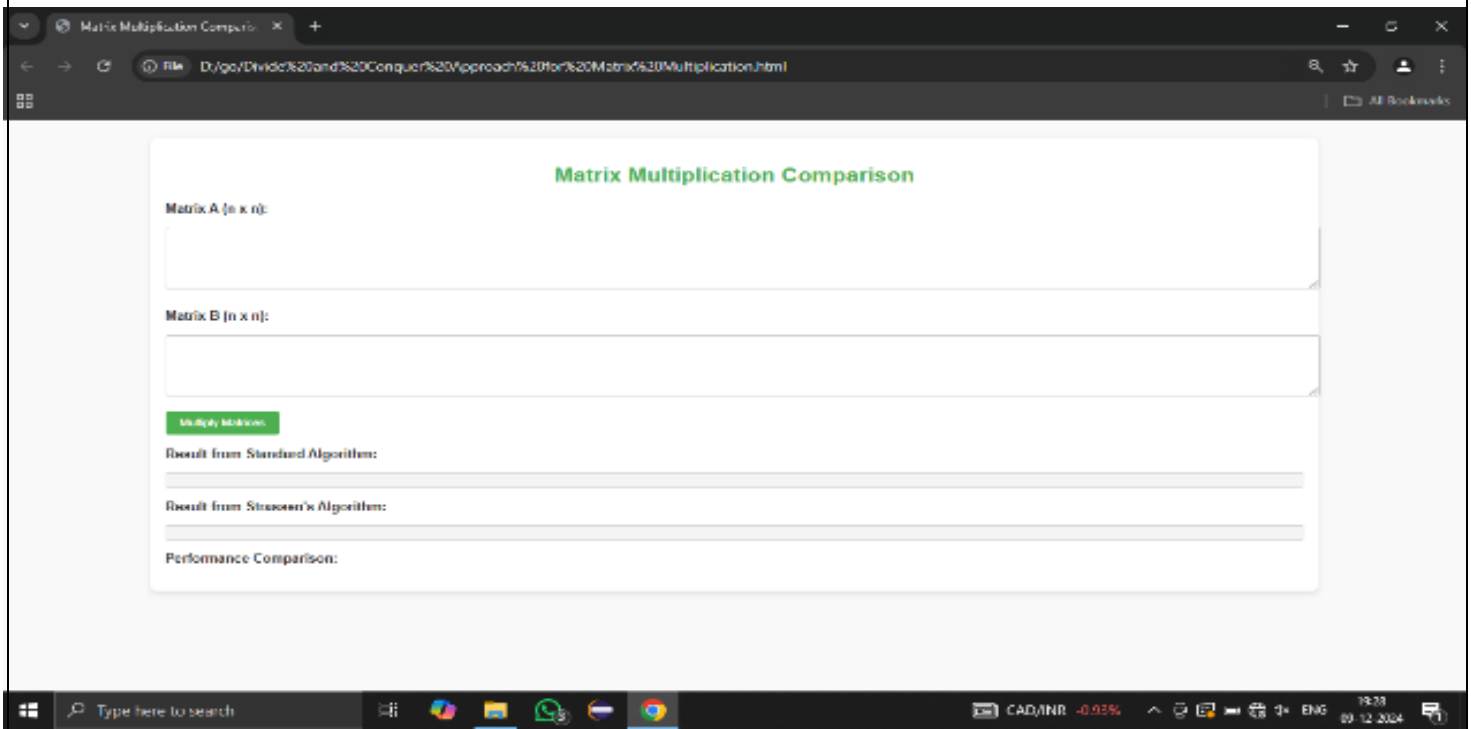
```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport"
content="width=device-width, initial-scale=1.0">
 <title>Matrix Multiplication Comparison</title>
 <style>
  body {
   font-family: Arial, sans-serif;
   padding: 20px;
   background-color: #f9f9f9;
  }
.container {
   width: 80%;
   margin: 0 auto;
   background-color: #fff;
   padding: 20px;
   border-radius: 8px;
   box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
```

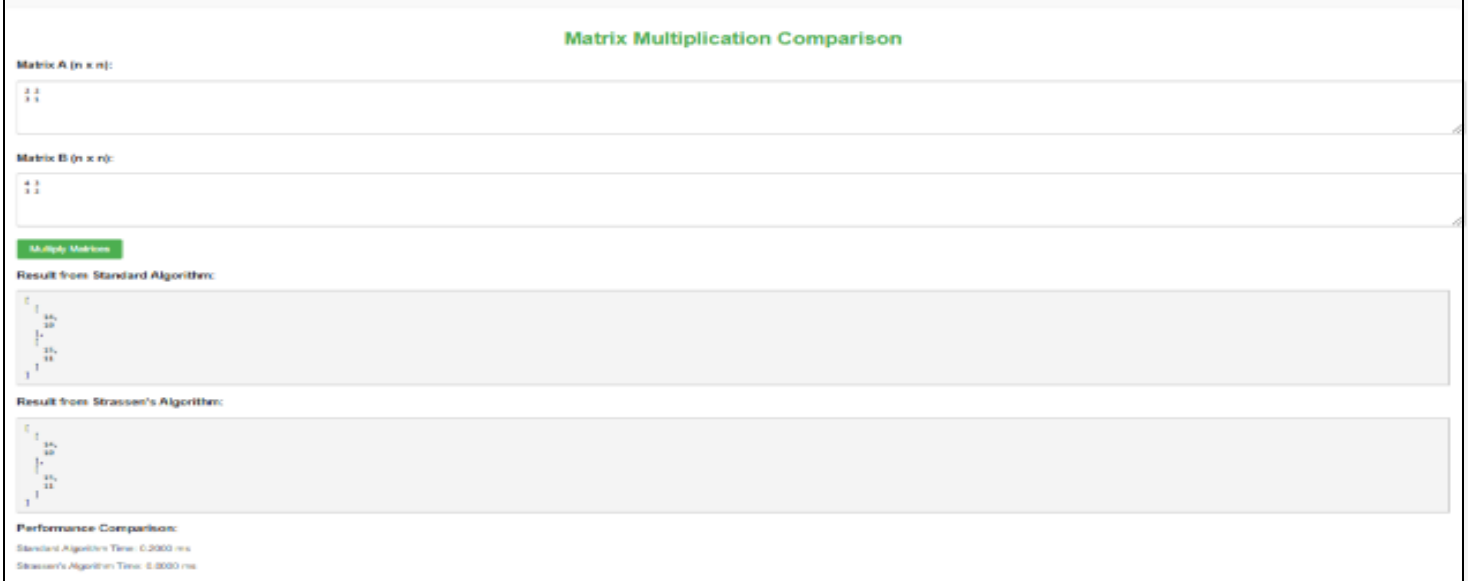
```
  }
 h1 {
    text-align: center;
    color: #4CAF50; /* Green color for title */
    font-size: 2em;
  }

  textarea {
    width: 100%;
    margin-bottom: 10px;
    padding: 10px;
    font-size: 1em;
    border-radius: 4px;
    border: 1px solid #ccc;
  }

  button {
    padding: 10px 20px;
    margin-top: 10px;
    background-color: #4CAF50; /* Green color for
button */
    color: white;
    border: none;
    border-radius: 4px;
    font-size: 1em;
    cursor: pointer;
    transition: background-color 0.3s;
  }

  button:hover {
    background-color: #45a049; /* Darker green
on hover */
  }

  pre {
    background-color: #f4f4f4;
    padding: 10px;
    border: 1px solid #ccc;
    font-family: monospace;
    overflow-x: auto;
  }

  h3 {
    color: #333;
    font-size: 1.2em;
  }

  p {
    font-size: 1em;
    color: #555;
  }
 </style>
</head>
<body>
 <div class="container">
  <h1>Matrix Multiplication Comparison</h1>

  <!-- Input for Matrix A and B -->
  <div>
   <h3>Matrix A (n x n):</h3>
   <textarea id="matrixA" rows="5"
cols="30"></textarea>
  </div>

  <div>
   <h3>Matrix B (n x n):</h3>
   <textarea id="matrixB" rows="5"
cols="30"></textarea>
```

```
  </div>

  <button onclick="performMultiplications()">Multiply Matrices</button>

  <h3>Result from Standard Algorithm:</h3>
  <pre id="standardResult"></pre>

  <h3>Result from Strassen's Algorithm:</h3>
  <pre id="strassenResult"></pre>

  <h3>Performance Comparison:</h3>
  <p id="standardTime"></p>
  <p id="strassenTime"></p>
 </div>

 <script>
  // Standard Matrix Multiplication
  function standardMatrixMultiply(A, B) {
    const n = A.length;
    const result = Array.from({ length: n }, () => Array(n).fill(0));

    for (let i = 0; i < n; i++) {
      for (let j = 0; j < n; j++) {
        for (let k = 0; k < n; k++) {
          result[i][j] += A[i][k] * B[k][j];
        }
      }
    }
    return result;
  }

  // Strassen's Algorithm
  function strassenMatrixMultiply(A, B) {
    const n = A.length;
    if (n === 1) {
      return [[A[0][0] * B[0][0]]];
    }

    const mid = Math.floor(n / 2);

    // Divide A into 4 submatrices
    const A11 = getSubMatrix(A, 0, mid, 0, mid);
    const A12 = getSubMatrix(A, 0, mid, mid, n);
    const A21 = getSubMatrix(A, mid, n, 0, mid);
    const A22 = getSubMatrix(A, mid, n, mid, n);

    // Divide B into 4 submatrices
    const B11 = getSubMatrix(B, 0, mid, 0, mid);
    const B12 = getSubMatrix(B, 0, mid, mid, n);
    const B21 = getSubMatrix(B, mid, n, 0, mid);
    const B22 = getSubMatrix(B, mid, n, mid, n);

    // Compute M1 to M7 using Strassen's formula
    const M1 = strassenMatrixMultiply(addMatrices(A11, A22), addMatrices(B11, B22));
    const M2 = strassenMatrixMultiply(addMatrices(A21, A22), B11);
    const M3 = strassenMatrixMultiply(A11, subtractMatrices(B12, B22));
    const M4 = strassenMatrixMultiply(A22, subtractMatrices(B21, B11));
    const M5 = strassenMatrixMultiply(addMatrices(A11, A12), B22);
```

```
    const M6 =
strassenMatrixMultiply(subtractMatrices(A21,
A11), addMatrices(B11, B12));

    const M7 =
strassenMatrixMultiply(subtractMatrices(A12,
A22), addMatrices(B21, B22));


    // Combine the results

    const C11 =
addMatrices(subtractMatrices(addMatrices(M1,
M4), M5), M7);

    const C12 = addMatrices(M3, M5);

    const C21 = addMatrices(M2, M4);

    const C22 =
addMatrices(subtractMatrices(addMatrices(M1,
M3), M2), M6);


    return combineSubMatrices(C11, C12, C21,
C22, n);
  }


  function getSubMatrix(matrix, rowStart,
rowEnd, colStart, colEnd) {
    const subMatrix = [];
    for (let i = rowStart; i < rowEnd; i++) {
      subMatrix.push(matrix[i].slice(colStart,
colEnd));
    }
    return subMatrix;
  }


  function addMatrices(A, B) {
    const n = A.length;
    const result = Array.from({ length: n }, (_, i)
=>
      A[i].map((val, j) => val + B[i][j])
    );
    return result;
  }


  function subtractMatrices(A, B) {
    const n = A.length;
    const result = Array.from({ length: n }, (_, i)
=>
      A[i].map((val, j) => val - B[i][j])
    );
    return result;
  }

 function combineSubMatrices(C11, C12, C21,
C22, n) {
    const result = Array.from({ length: n }, () =>
Array(n).fill(0));


    for (let i = 0; i < n / 2; i++) {
      for (let j = 0; j < n / 2; j++) {
        result[i][j] = C11[i][j];
        result[i][j + n / 2] = C12[i][j];
        result[i + n / 2][j] = C21[i][j];
        result[i + n / 2][j + n / 2] = C22[i][j];
      }
    }
    return result;
  }


  function parseMatrix(input) {
    const rows = input.trim().split("\n");
    return rows.map(row => row.split("
").map(Number));
  }


  function performMultiplications() {
```

```
        const matrixAInput =
document.getElementById("matrixA").value;

        const matrixBInput =
document.getElementById("matrixB").value;

        const A = parseMatrix(matrixAInput);

        const B = parseMatrix(matrixBInput);



        // Standard multiplication

        const startStandard = performance.now();


     const standardResult =
standardMatrixMultiply(A, B);

        const endStandard = performance.now();

        const standardTime = endStandard -
startStandard;

    // Strassen's multiplication

        const startStrassen = performance.now();

        const strassenResult =
strassenMatrixMultiply(A, B);

        const endStrassen = performance.now();

const strassenTime = endStrassen - startStrassen;


        // Display Results

document.getElementById("standardResult").tex
tContent = JSON.stringify(standardResult, null, 2);


document.getElementById("strassenResult").text
Content = JSON.stringify(strassenResult, null, 2);


document.getElementById("standardTime").text
Content = `Standard Algorithm Time:
${standardTime.toFixed(4)} ms`;


document.getElementById("strassenTime").textC
ontent = `Strassen's Algorithm Time:
${strassenTime.toFixed(4)} ms`;

    }

    </script>

</body>

</html>
```





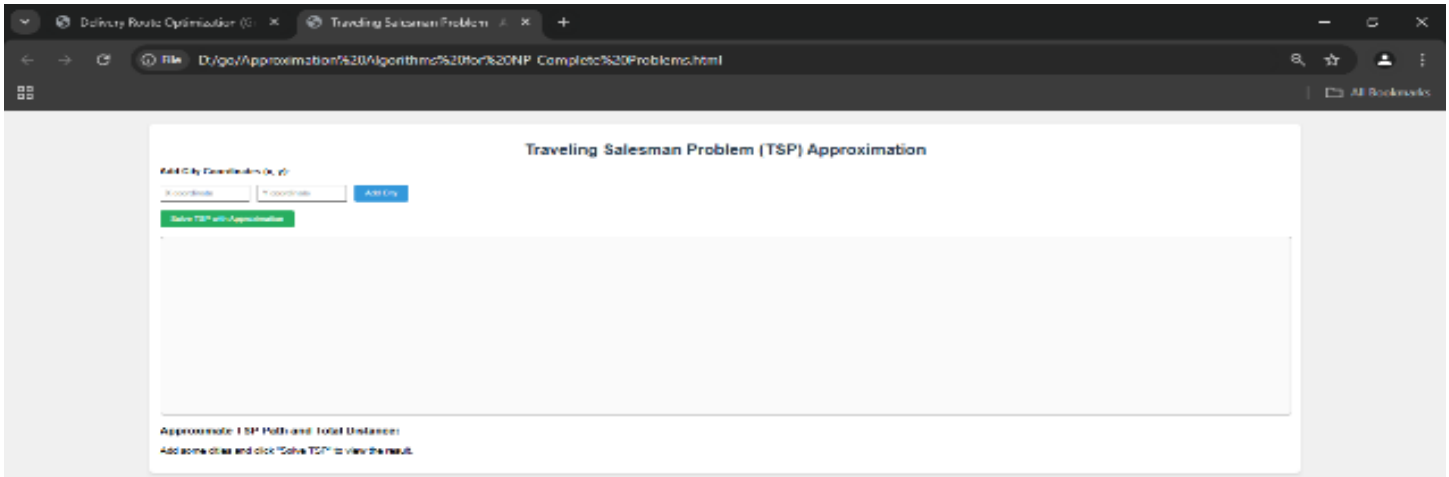
4: APPROXIMATION ALGORITHMS FOR NP-COMPLETE PROBLEMS: IMPLEMENT AN APPROXIMATION ALGORITHM FOR SOLVING THE TRAVELING SALESMAN PROBLEM. ANALYZE HOW CLOSE THE SOLUTION IS TO THE OPTIMAL PATH ANDDISCUSS THE COMPLEXITY OF THE ALGORITHM

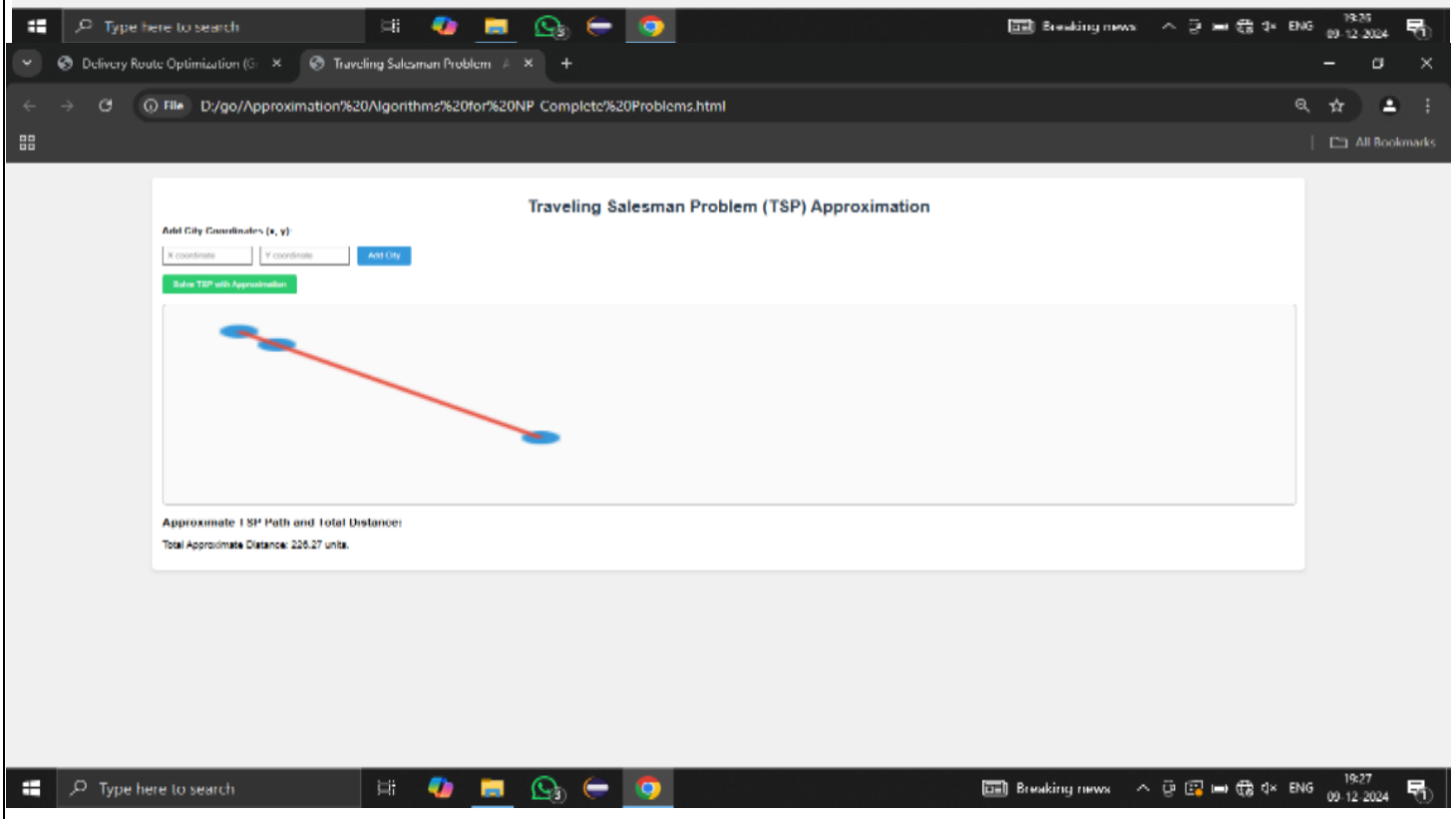
```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width, initial-scale=1.0">
 <title>Traveling Salesman Problem - Approximation</title>
 <style>
  body {
   font-family: Arial, sans-serif;
   background-color: #f0f0f0;
   padding: 20px;
  }
  h1 {
   text-align: center;
   color: #2c3e50;
  }
container {
   width: 80%;
   margin: 0 auto;
   padding: 20px;
   background-color: #fff;
   border-radius: 8px;
   box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
  }
 #canvas {
   margin: 20px 0;
border: 1px solid #ccc;
   display: block;
   background-color: #fafafa;
   width: 100%;
```

```
  height: 400px;
  border-radius: 8px;
}
button {
  padding: 10px 20px;
  background-color: #27ae60;
  color: white;
  font-size: 1em;
  border: none;
  border-radius: 4px;
  cursor: pointer;
}
button:hover {
  background-color: #2ecc71;
}
.result {
  margin-top: 20px;
  font-size: 1.2em;
}
.input-section {
  margin-bottom: 20px;
}
input[type="text"] {
  padding: 8px;
  width: 150px;
  margin-right: 10px;
  font-size: 1em;
}
.input-section button {
  background-color: #3498db;
}
.input-section button:hover {
  background-color: #2980b9;
  }
 </style>
</head>
<body>
 <div class="container">
  <h1>Traveling Salesman Problem (TSP) Approximation</h1>

  <!-- Input Section for Adding Cities -->
  <div class="input-section">
   <h3>Add City Coordinates (x, y):</h3>
   <input type="text" id="cityX" placeholder="X coordinate">
   <input type="text" id="cityY" placeholder="Y coordinate">
   <button onclick="addCity()">Add City</button>
  </div>

  <button onclick="solveTSP()">Solve TSP with Approximation</button>

  <canvas id="canvas"></canvas>

  <div class="result">
   <h3>Approximate TSP Path and Total Distance:</h3>
   <p id="tspResult">Add some cities and click "Solve TSP" to view the result.</p>
  </div>
 </div>

 <script>
  const canvas = document.getElementById("canvas");
  const ctx = canvas.getContext("2d");
```

```
let cities = [];
let tspPath = [];
let totalDistance = 0;

// Add city to the list
function addCity() {
  const cityX = parseFloat(document.getElementById("cityX").value);
  const cityY = parseFloat(document.getElementById("cityY").value);

  // Ensure the city coordinates are valid
  if (isNaN(cityX) || isNaN(cityY)) {
    alert("Please enter valid coordinates.");
    return;
  }

  // Add the city to the cities array
  cities.push({ x: cityX, y: cityY });

  // Clear the input fields
  document.getElementById("cityX").value = "";
  document.getElementById("cityY").value = "";

  drawCities(); // Re-draw the cities
}

// Calculate Euclidean distance between two cities
function calculateDistance(city1, city2) {
  return Math.sqrt(Math.pow(city2.x - city1.x, 2) + Math.pow(city2.y - city1.y, 2));
}

// Nearest Neighbor Algorithm to find an approximate solution to TSP
function solveTSP() {
  if (cities.length < 2) {
    alert("Please add at least 2 cities to solve the TSP.");
    return;
  }

  let unvisitedCities = [...cities];
  let currentCity = unvisitedCities.pop();
  tspPath = [currentCity];
  totalDistance = 0;

  while (unvisitedCities.length > 0) {
    let nearestCity = findNearestCity(currentCity, unvisitedCities);
    tspPath.push(nearestCity);
    totalDistance += calculateDistance(currentCity, nearestCity);
    currentCity = nearestCity;
    unvisitedCities = unvisitedCities.filter(city => city !== nearestCity);
  }

  // Complete the cycle by returning to the starting city
  totalDistance += calculateDistance(tspPath[tspPath.length - 1], tspPath[0]);
  tspPath.push(tspPath[0]); // Return to the starting point

  drawCitiesAndPath(); // Re-draw cities and the TSP path
```

```
document.getElementById("tspResult").textConte
nt = `Total Approximate Distance:
${totalDistance.toFixed(2)} units.`;
  }

  // Find the nearest city to the current city
  function findNearestCity(currentCity, cities) {
    let nearestCity = cities[0];
    let minDistance =
calculateDistance(currentCity, nearestCity);

    cities.forEach(city => {
      const distance =
calculateDistance(currentCity, city);
      if (distance < minDistance) {
        minDistance = distance;
        nearestCity = city;
      }
    });

    return nearestCity;
  }

  // Draw cities on the canvas
  function drawCities() {
    ctx.clearRect(0, 0, canvas.width,
canvas.height);

    // Draw all cities
    cities.forEach(city => {
      ctx.beginPath();
      ctx.arc(city.x, city.y, 5, 0, 2 * Math.PI);
      ctx.fillStyle = "#3498db";
      ctx.fill();
    });
  }

  // Draw TSP path on the canvas
  function drawCitiesAndPath() {
ctx.clearRect(0, 0, canvas.width, canvas.height);

    // Draw cities as dots
    cities.forEach(city => {
      ctx.beginPath();
      ctx.arc(city.x, city.y, 5, 0, 2 * Math.PI);
      ctx.fillStyle = "#3498db";
      ctx.fill();
    });

    // Draw TSP path
    ctx.beginPath();
    ctx.moveTo(tspPath[0].x, tspPath[0].y);
    tspPath.forEach(city => {
      ctx.lineTo(city.x, city.y);
    });
    ctx.strokeStyle = "#e74c3c";
    ctx.lineWidth = 2;
    ctx.stroke();
  }
 </script>
</body>
</html>
```


Traveling Salesman Problem (TSP) Approximation
Approximate TSP Path and Total Distance:


Traveling Salesman Problem (TSP) Approximation
Add City Coordinates (x, y):
Add City
Approximate TSP Path and Total Distance:
Total Approximate Distance: 226.27 units.