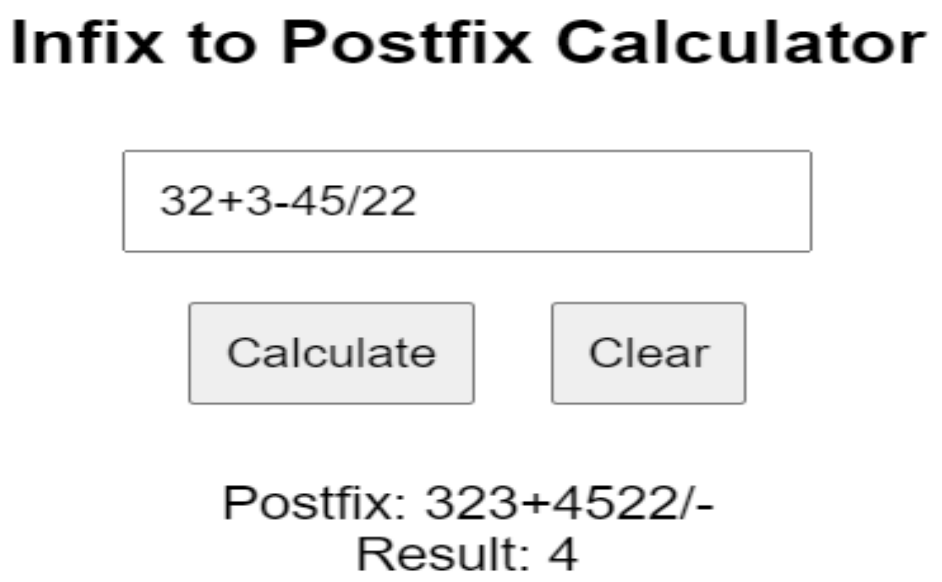
1: Expression Evaluation (Infix to Postfix Conversion): Implement a calculator that converts infix expressions to postfix notation using stacks. Evaluate the postfix expression to return the result. Handle complex expressions with parentheses and operator precedence efficiently.

```
<!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>
  <style>
    body {
      font-family: Arial, sans-serif;
      text-align: center;
      margin: 20px;
    }
    input, button {
      margin: 10px;
      padding: 10px;
      font-size: 16px;
    }
    #result {
      margin-top: 20px;
      font-size: 18px;
    }
  </style>
</head>
<body>
  <h2>Infix to Postfix Calculator</h2>
  <input type="text" id="expression" placeholder="Enter expression (e.g., 3+(5*2))">
  <div>
    <button onclick="calculate()">Calculate</button>
    <button onclick="clearInput()">Clear</button>
  </div>
  <div id="result"></div>
  <script>
    function precedence(op) {
      return { '+': 1, '-': 1, '*': 2, '/': 2 }[op] || 0;
    }
    function infixToPostfix(expression) {
      let stack = [], postfix = '';
      for (let char of expression) {
        if (!isNaN(char)) {
          postfix += char;
        } else if (char === '(') {
          stack.push(char);
        } else if (char === ')') {
          while (stack.length && stack[stack.length - 1] !== '(') {
            postfix += stack.pop(); }
          stack.pop();
        } else {
          while (stack.length && precedence(stack[stack.length - 1]) >= precedence(char)) {
            postfix += stack.pop();
          }
          stack.push(char);
        }
      }
```

```
    while (stack.length) postfix += stack.pop();
    return postfix;
  }
  function evaluatePostfix(postfix) {
    let stack = [];
    for (let char of postfix) {
      if (!isNaN(char)) {
        stack.push(Number(char));
      } else {
        let b = stack.pop(), a = stack.pop();
        stack.push(char === '+' ? a + b : char === '-' ? a - b : char === '*' ? a * b : a / b);
      }
    }
    return stack.pop();
  }
  function calculate() {
    const expression = document.getElementById('expression').value;
    try {
      const postfix = infixToPostfix(expression);
      const result = evaluatePostfix(postfix);
      document.getElementById('result').innerHTML = `Postfix: ${postfix}<br>Result: ${result}`;
    } catch (e) {
      document.getElementById('result').innerText = 'Invalid Expression!';
    }
  }
  function clearInput() {
    document.getElementById('expression').value = '';
    document.getElementById('result').innerText = '';
  }
 </script>
</body>
</html>
```

Output:

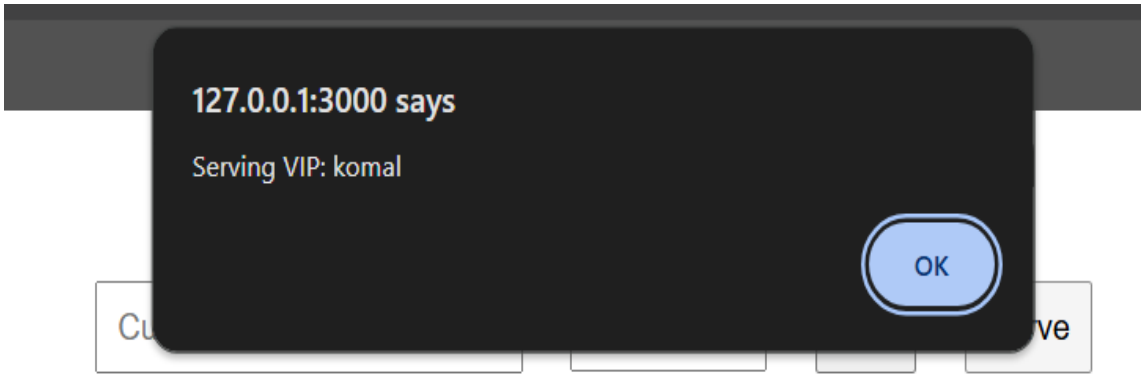

Infix to Postfix Calculator
32+3-45/22
Calculate
Clear
Postfix: 323+4522/-
Result: 4

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.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Ticketing System</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      text-align: center;
      margin: 20px;
    }
    input, select, button {
      margin: 10px;
      padding: 10px;
      font-size: 16px;
    }
    .queue {
      margin-top: 20px;
    }
  </style>
</head>
<body>
  <h2>Ticketing System</h2>
  <input type="text" id="name" placeholder="Customer Name">
  <select id="type">
    <option value="VIP">VIP</option>
    <option value="Regular">Regular</option>
  </select>
  <button onclick="addCustomer()">Add</button>
  <button onclick="serveCustomer()">Serve</button>
  <div class="queue" id="queueDisplay">Queue is empty</div>

  <script>
    let vipQueue = [];
    let regularQueue = [];

    function addCustomer() {
      const name = document.getElementById("name").value.trim();
      const type = document.getElementById("type").value;

      if (!name) {
        alert("Enter a name.");
        return;
      }

      type === "VIP" ? vipQueue.push(name) : regularQueue.push(name);
      document.getElementById("name").value = "";
      updateQueue();
```

```
    }

    function serveCustomer() {
      if (vipQueue.length > 0) {
        alert(`Serving VIP:
${vipQueue.shift()}`);
      } else if (regularQueue.length > 0) {
        alert(`Serving Regular:
${regularQueue.shift()}`);
      } else {
        alert("No customers in the queue.");
      }
      updateQueue();
    }

    function updateQueue() {
      const display =
document.getElementById("queueDisplay"
);
      const vipList = vipQueue.map((name)
=> `VIP: ${name}`).join("<br>");
      const regularList =
regularQueue.map((name) => `Regular:
${name}`).join("<br>");
      display.innerHTML = vipQueue.length
+ regularQueue.length
        ? `${vipList}<br>${regularList}`
        : "Queue is empty";
    }
  </script>
</body>
</html>
```

Output:



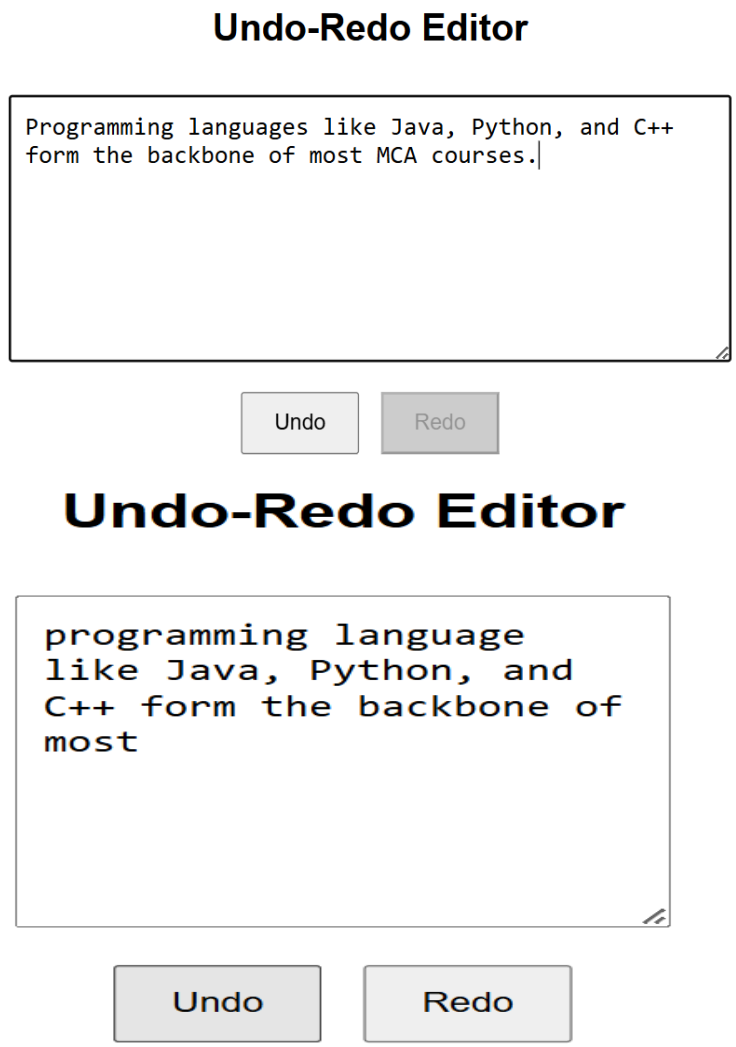
VIP: komal
VIP: vishakha
Regular: anushka
Regular: sanika

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.

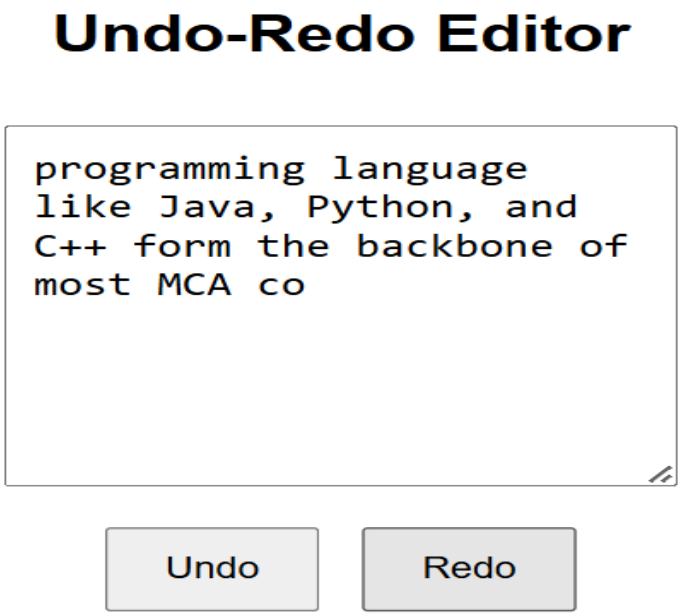
```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Undo-Redo Code Editor</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      text-align: center;
      padding: 20px;
    }
    textarea {
      width: 80%;
      height: 150px;
      margin: 10px;
      font-size: 16px;
      padding: 10px;
    }
    button {
      margin: 5px;
      padding: 8px 12px;
      font-size: 14px;
      background-color: #007BFF;
      color: white;
      border: none;
      cursor: pointer;
    }
    button:hover {
      background-color: #0056b3;
    }
  </style>
</head>
<body>
  <h2>Undo-Redo Code Editor</h2>
  <textarea id="editor" placeholder="Type your code here..."></textarea>
  <br>
  <button onclick="undo()">Undo</button>
  <button onclick="redo()">Redo</button>

  <script>
    const undoStack = []; // Stack to track undo actions
    const redoStack = []; // Stack to track redo actions

    const editor = document.getElementById("editor");

    // Track changes whenever text is modified
    editor.addEventListener("input", () => {
      undoStack.push(editor.value); // Push current state to undo stack
      redoStack.length = 0; // Clear redo stack after a new change
    });

    // Undo functionality
    function undo() {
```

```
      if (undoStack.length > 0) {
        redoStack.push(undoStack.pop()); // Move current state to redo stack
        editor.value = undoStack.length > 0 ? undoStack[undoStack.length - 1] : ""; // Restore previous state
      } else {
        alert("Nothing to undo.");
      }
    }

    // Redo functionality
    function redo() {
      if (redoStack.length > 0) {
        const state = redoStack.pop(); // Restore last undone state
        undoStack.push(state); // Push it back to the undo stack
        editor.value = state; // Update the editor with restored state
      } else {
        alert("Nothing to redo.");
      }
    }
  </script>
</body>
</html>
```

Output:


Undo-Redo Editor
Programming languages like Java, Python, and C++ form the backbone of most MCA courses.
Undo
Redo

Undo-Redo Editor
programming language like Java, Python, and C++ form the backbone of most
Undo
Redo

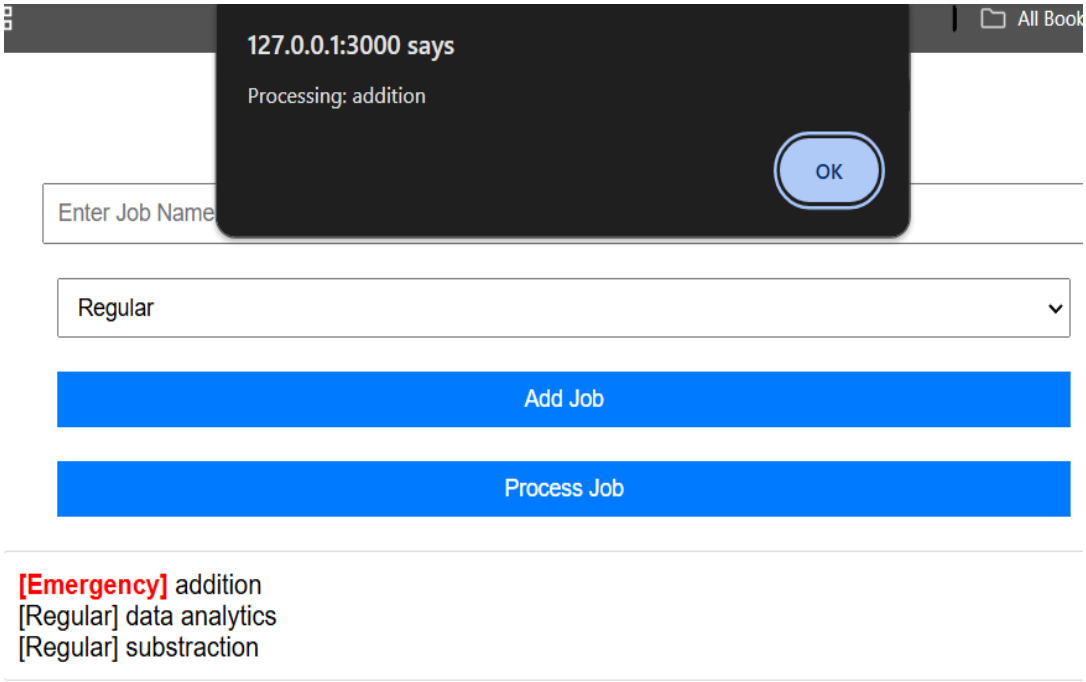

Undo-Redo Editor
programming language like Java, Python, and C++ form the backbone of most MCA co
Undo
Redo

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.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Job Queue</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      text-align: center;
      margin: 20px;
    }
    input, select, button {
      margin: 10px 0;
      padding: 8px;
      width: 90%;
      font-size: 14px;
    }
    button {
      background: #007BFF;
      color: white;
      border: none;
      cursor: pointer;
    }
    button:disabled {
      background: #ccc;
      cursor: not-allowed;
    }
    .queue-display {
      margin-top: 10px;
      text-align: left;
      padding: 10px;
      border: 1px solid #ddd;
      border-radius: 4px;
    }
    .priority {
      color: red;
      font-weight: bold;
    }
  </style>
</head>
<body>
  <h2>Job Queue System</h2>
  <input id="jobName" placeholder="Enter Job Name">
  <select id="priority">
    <option>Regular</option>
    <option>Emergency</option>
  </select>
  <button onclick="addJob()">Add Job</button>
  <button onclick="processJob()">Process Job</button>
  <div class="queue-display" id="queueDisplay">No jobs in the queue.</div>

  <script>
    const emergencyQueue = [], regularQueue = [];
```

```
    function addJob() {
      const job = document.getElementById("jobName").value.trim();
      const priority = document.getElementById("priority").value;

      if (!job) return alert("Enter a job name.");
      (priority === "Emergency" ? emergencyQueue : regularQueue).push(job);
      document.getElementById("jobName").value = "";
      updateQueue();
    }

    function processJob() {
      const job = emergencyQueue.length ? emergencyQueue.shift() : regularQueue.shift();
      if (!job) return alert("No jobs to process.");
      alert(`Processing: ${job}`);
      updateQueue();
    }

    function updateQueue() {
      const display = document.getElementById("queueDisplay");
      const jobs = [
        ...emergencyQueue.map(job => `<div><span class="priority">[Emergency]</span> ${job}</div>`),
        ...regularQueue.map(job => `<div>[Regular] ${job}</div>`)
      ];
      display.innerHTML = jobs.length ? jobs.join('') : "No jobs in the queue.";
    }
  </script>
</body>
</html>
```

Output:


All Book
127.0.0.1:3000 says
Processing: addition
OK
Enter Job Name
Regular
Add Job
Process Job
[Emergency] addition
[Regular] data analytics
[Regular] substraction

5: 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.

```
<!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>
  <style>
    body {
      font-family: Arial, sans-serif;
      text-align: center;
      padding: 20px;
    }
    input, button {
      margin: 10px 0;
      padding: 8px;
      font-size: 14px;
    }
    button {
      background: #007BFF;
      color: white;
      border: none;
      cursor: pointer;
    }
    .result {
      margin-top: 20px;
    }
    table {
      width: 100%;
      margin-top: 20px;
      border-collapse: collapse;
    }
    th, td {
      border: 1px solid #ddd;
      padding: 8px;
      text-align: center;
    }
    th {
      background-color: #f4f4f4;
    }
  </style>
</head>
<body>
  <h2>Stock Span Problem</h2>
  <input type="text" id="prices" placeholder="Enter prices separated by commas">
  <button onclick="calculateSpan()">Calculate Span</button>
  <div class="result" id="result"></div>

  <script>
    function calculateSpan() {
      const input = document.getElementById("prices").value.trim();
      if (!input) {
        alert("Please enter stock prices.");
        return;
      }
```

```
    const prices = input.split(',').map(Number);
    if (prices.some(isNaN)) {
      alert("Invalid input. Please enter valid numbers.");
      return;
    }

    const span = [];
    const stack = [];

    for (let i = 0; i < prices.length; i++) {
      while (stack.length && prices[stack[stack.length - 1]] <= prices[i]) {
        stack.pop();
      }
      span[i] = stack.length === 0 ? i + 1 : i - stack[stack.length - 1];
      stack.push(i);
    }

    displayResult(prices, span);
  }

  function displayResult(prices, span) {
    const resultDiv = document.getElementById("result");
    let tableHtml = `
      <table>
        <tr>
          <th>Day</th>
          <th>Price</th>
          <th>Span</th>
        </tr>
    `;

    for (let i = 0; i < prices.length; i++) {
      tableHtml += `
        <tr>
          <td>${i + 1}</td>
          <td>${prices[i]}</td>
          <td>${span[i]}</td>
        </tr>
      `;
    }

    tableHtml += '</table>';
    resultDiv.innerHTML = tableHtml;
  }
 </script>
</body>
</html>
```

Output:

Stock Span Problem

5,10,25,30,15,5 Calculate Span

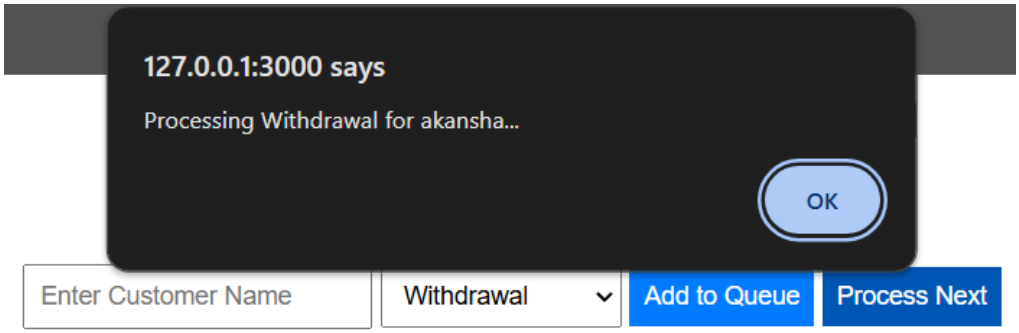
Day	Price	Span
1	5	1
2	10	2
3	25	3
4	30	4
5	15	1
6	5	1

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.

```
<!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>
  <style>
    body {
      font-family: Arial, sans-serif;
      text-align: center;
      padding: 20px;
    }
    input, select, button {
      margin: 10px 0;
      padding: 8px;
      font-size: 14px;
    }
    button {
      background-color: #007BFF;
      color: white;
      border: none;
      cursor: pointer;
    }
    button:hover {
      background-color: #0056b3;
    }
    .queue-display {
      margin-top: 20px;
    }
    ul {
      list-style-type: none;
      padding: 0;
    }
    li {
      margin: 5px 0;
      padding: 8px;
      background-color: #f4f4f4;
      border: 1px solid #ddd;
      border-radius: 4px;
    }
  </style>
</head>
<body>
  <h2>Bank ATM Queue Simulation</h2>
  <input type="text" id="customerName" placeholder="Enter Customer Name">
  <select id="transactionType">
    <option value="Deposit">Deposit</option>
    <option value="Withdrawal">Withdrawal</option>
    <option value="Balance Check">Balance Check</option>
  </select>
  <button onclick="addToQueue()">Add to Queue</button>
  <button onclick="processNext()">Process Next</button>
  <div class="queue-display">
```

```
    <h3>Current Queue:</h3>
    <ul id="queueList">
      <li>No customers in the queue.</li>
    </ul>
  </div>
  <script>
    const queue = [];
    const transactionTimes = {
      "Deposit": 2000,
      "Withdrawal": 3000,
      "Balance Check": 1000
    };
    function addToQueue() {
      const customerName =
document.getElementById("customerNam
e").value.trim();
      const transactionType =
document.getElementById("transactionTy
pe").value;
      if (!customerName) {
        alert("Please enter a customer
name.");
        return;
      }
      queue.push({ name: customerName,
transaction: transactionType });
      document.getElementById("customer
Name").value = "";
      updateQueueDisplay();
    }
    function processNext() {
      if (queue.length === 0) {
        alert("No customers in the queue.");
        return;
      }
      const currentCustomer = queue.shift();
      alert(`Processing
${currentCustomer.transaction} for
${currentCustomer.name}...`);
      setTimeout(() => {
        alert(`${currentCustomer.transaction}
completed for
${currentCustomer.name}.`);
        updateQueueDisplay();
      },
transactionTimes[currentCustomer.transac
tion]);
    }
    function updateQueueDisplay() {
      const queueList =
document.getElementById("queueList");
      queueList.innerHTML = "";
      if (queue.length === 0) {
        queueList.innerHTML = "<li>No
customers in the queue.</li>";
      } else {
        queue.forEach((customer, index) => {
          queueList.innerHTML +=
`<li>${index + 1}. ${customer.name} -
${customer.transaction}</li>`;
        });
      }
    }
  </script>
</body>
</html>
```

Output:



Current Queue:

1. akansha - Withdrawal
2. sahil - Deposit
3. ruchika - Balance Check
4. tanushree - Withdrawal

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.

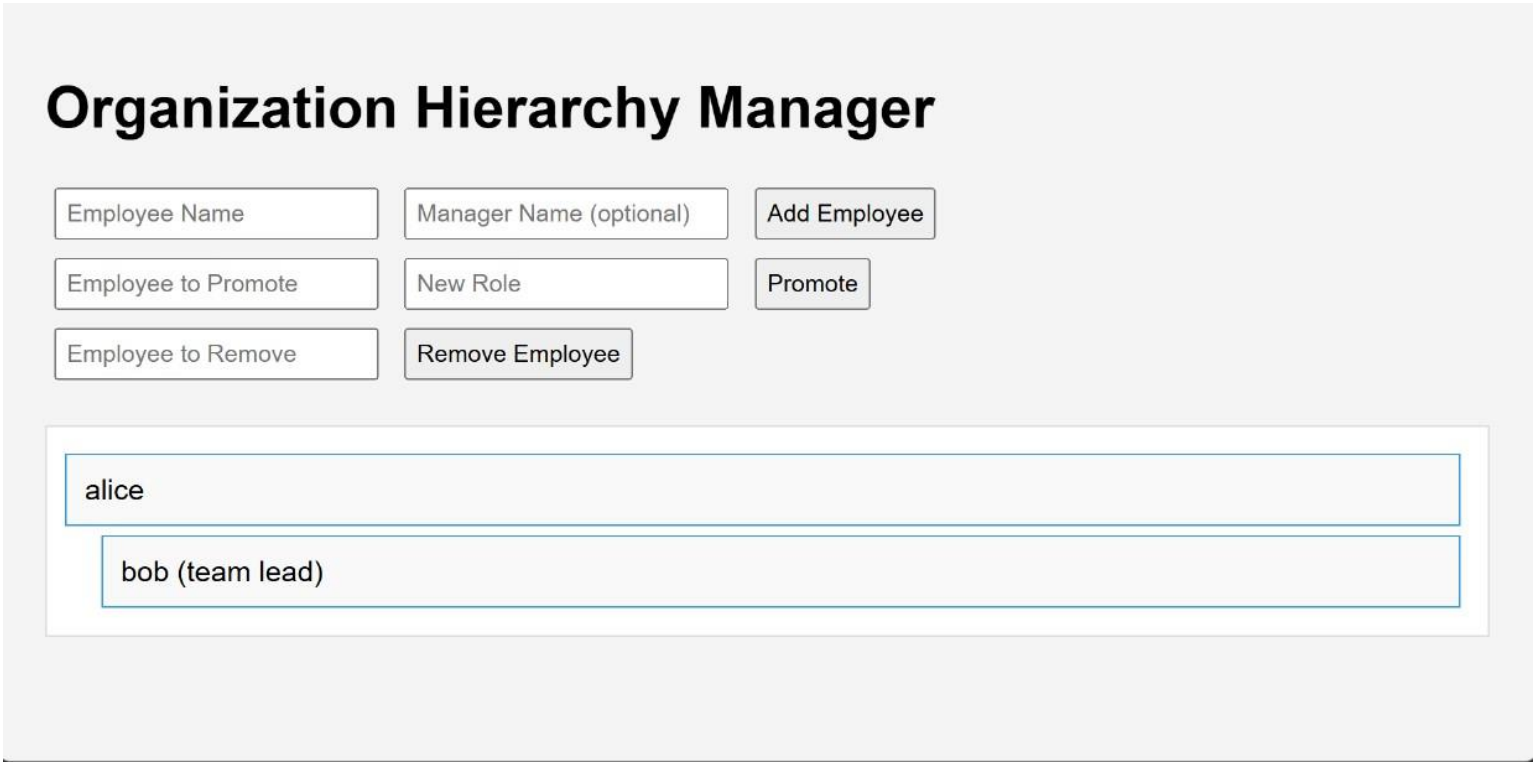
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Org Hierarchy Manager</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            max-width: 800px;
            margin: 0 auto;
            padding: 20px;
            background-color: #f4f4f4;
        }
        #orgChart {
            background-color: white;
            border: 1px solid #ddd;
            padding: 10px;
            margin-top: 20px;
        }
        .employee {
            border: 1px solid #3498db;
            margin: 5px;
            padding: 10px;
            background-color: #f9f9f9;
        }
        input, button {
            margin: 5px;
            padding: 5px;
        }
    </style>
</head>
<body>
    <h1>Organization Hierarchy Manager</h1>
    <div>
        <input type="text" id="employeeName" placeholder="Employee Name">
        <input type="text" id="managerName" placeholder="Manager Name (optional)">
        <button onclick="addEmployee()">Add Employee</button>
    </div>
    <div>
        <input type="text" id="promoteName" placeholder="Employee to Promote">
        <input type="text" id="newRole" placeholder="New Role">
        <button onclick="promoteEmployee()">Promote</button>
    </div>
    <div>
        <input type="text" id="removeName" placeholder="Employee to Remove">
        <button onclick="removeEmployee()">Remove Employee</button>
    </div>
    <div id="orgChart"></div>
    <script>
        class OrgHierarchy {
```

```
        constructor() {
            this.hierarchy = {};
            this.root = null;
        }
        addEmployee(name, managerName = null) {
            const employee = { name, subordinates: [] };
            if (!managerName) {
                if (!this.root) {
                    this.root = employee;
                } else {
                    alert('Root already exists!');
                    return false;
                }
            } else {
                const manager = this.findEmployee(this.root, managerName);
                if (manager) {
                    manager.subordinates.push(employee);
                } else {
                    alert('Manager not found!');
                    return false;
                }
            }
            this.updateDisplay();
            return true;
        }
        findEmployee(node, name) {
            if (!node) return null;
            if (node.name === name) return node;
            for (let subordinate of node.subordinates) {
                const found = this.findEmployee(subordinate, name);
                if (found) return found;
            }
            return null;
        }
        removeEmployee(name) {
            const remove = (node) => {
                if (!node) return false;
                // Check in direct subordinates
                for (let i = 0; i < node.subordinates.length; i++) {
                    if (node.subordinates[i].name === name) {
                        node.subordinates.splice(i, 1);
                        this.updateDisplay();
                        return true;
                    }
                }
                // Recursively search in subtrees
                for (let subordinate of node.subordinates) {
                    if (remove(subordinate)) return true;
                }
                return false;
            };
            const result = remove(this.root);
            if (!result) alert('Employee not found!');
            this.updateDisplay();
```

```
            return result;
        }
        promoteEmployee(name,
newRole) {
            const employee =
this.findEmployee(this.root, name);
            if (employee) {
                employee.name = ${name}
(${newRole});
                this.updateDisplay();
                return true;
            }
            alert('Employee not found!');
            return false;
        }
        updateDisplay() {
            const chart =
document.getElementById('orgChart');
            chart.innerHTML = '';
            this.renderNode(this.root, chart);
        }
        renderNode(node, parentElement,
level = 0) {
            if (!node) return;
            const employeeDiv =
document.createElement('div');

employeeDiv.classList.add('employee');
            employeeDiv.style.marginLeft =
${level * 20}px;
            employeeDiv.textContent =
node.name;

parentElement.appendChild(employeeDiv)
;
            for (let subordinate of
node.subordinates) {
                this.renderNode(subordinate,
parentElement, level + 1);
            }
        }
    }
    const orgHierarchy = new
OrgHierarchy();

    function addEmployee() {
        const name =
document.getElementById('employeeNam
e').value;
        const managerName =
document.getElementById('managerName
').value;
        if (name) {

orgHierarchy.addEmployee(name,
managerName || null);

document.getElementById('employeeNam
e').value = '';

document.getElementById('managerName
').value = '';
        }
    }
    function removeEmployee() {
        const name =
document.getElementById('removeName')
.value;
        if (name) {

orgHierarchy.removeEmployee(name);

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

```
        }
        function promoteEmployee() {
            const name =
document.getElementById('promoteName'
).value;
            const newRole =
document.getElementById('newRole').valu
e;
            if (name && newRole) {

orgHierarchy.promoteEmployee(name,
newRole);

document.getElementById('promoteName'
).value = '';

document.getElementById('newRole').valu
e = '';
            }
        }
    </script>
</body>
</html>
```

Output:


Organization Hierarchy Manager
Employee Name
Manager Name (optional)
Add Employee
Employee to Promote
New Role
Promote
Employee to Remove
Remove Employee
alice
bob (team lead)

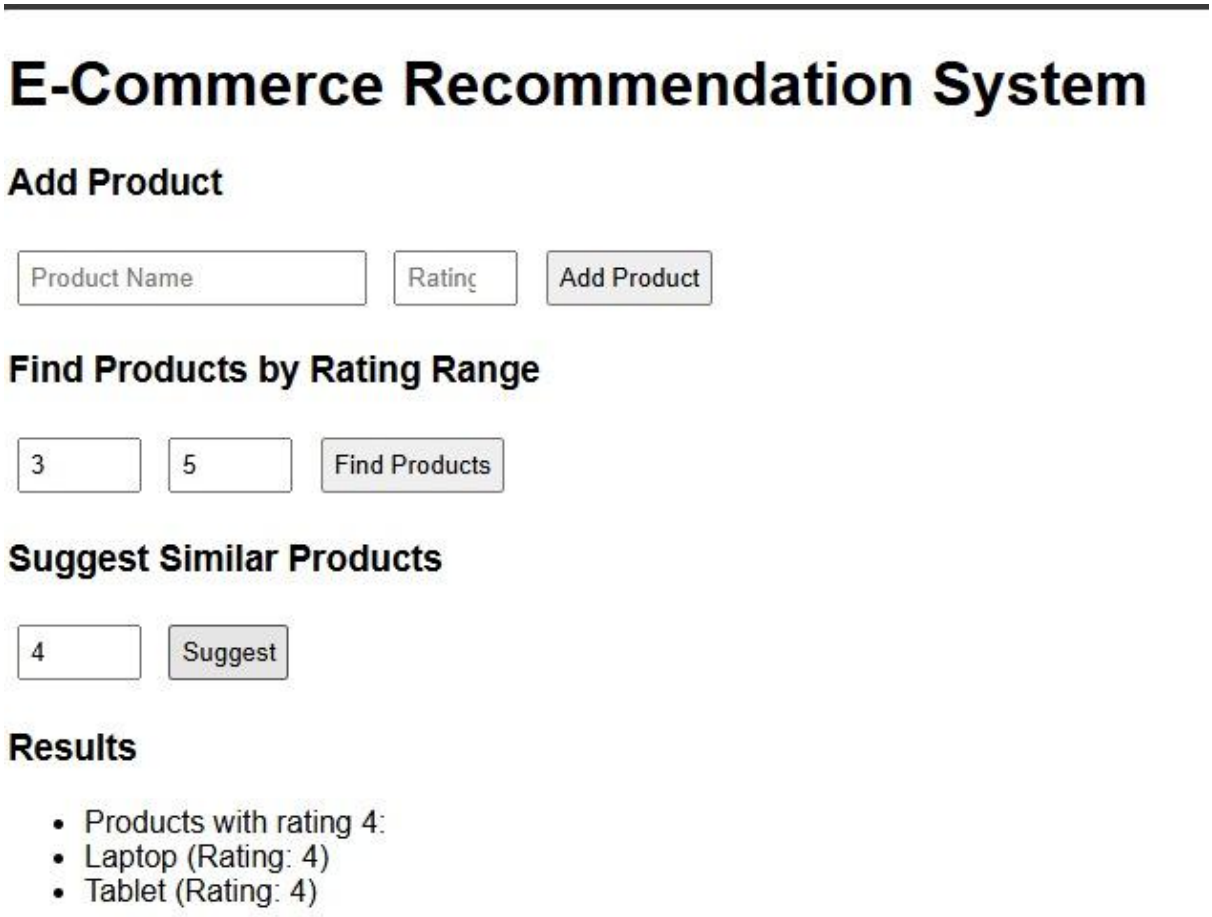
2: E-Commerce Recommendation System (Binary Search Tree): Build an e-commerce recommendation system where products are stored in a binary search tree (BST) based on customer ratings. Implement operations to find products within a specific rating range and suggest similar products.

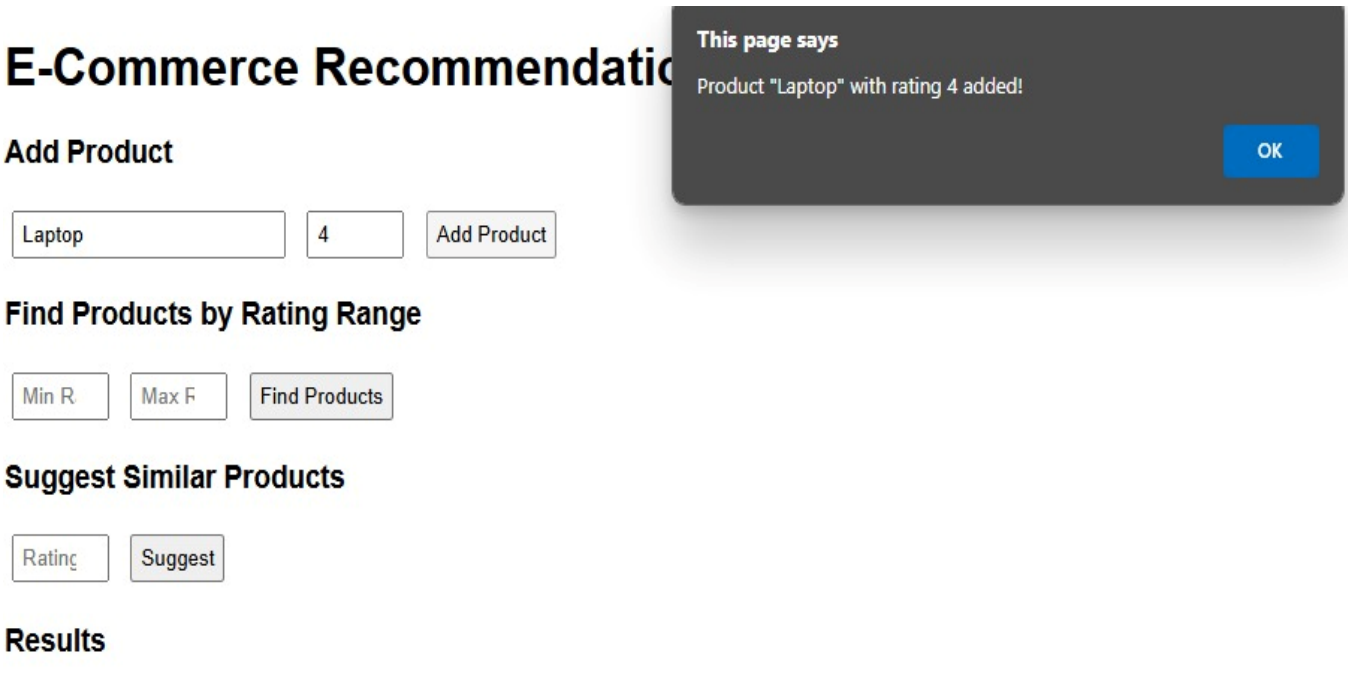
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>E-Commerce Recommendation System</title>
    <style>
        body { font-family: Arial, sans-serif; }
        input, button { margin: 5px; padding: 5px; }
        #results { margin-top: 20px; }
    </style>
</head>
<body>
    <div class="container">
        <h1>E-Commerce Recommendation System</h1>
        <div>
            <h3>Add Product</h3>
            <input type="text" id="product-name" placeholder="Product Name">
            <input type="number" id="product-rating" placeholder="Rating (1-5)" min="1" max="5">
            <button onclick="addProduct()">Add Product</button>
        </div>
        <div>
            <h3>Find Products by Rating Range</h3>
            <input type="number" id="min-rating" placeholder="Min Rating" min="1" max="5">
            <input type="number" id="max-rating" placeholder="Max Rating" min="1" max="5">
            <button onclick="findProductsInRange()">Find Products</button>
        </div>
        <div>
            <h3>Suggest Similar Products</h3>
            <input type="number" id="similar-rating" placeholder="Rating for Similar Products" min="1" max="5">
            <button onclick="suggestSimilarProducts()">Suggest</button>
        </div>
        <div id="results">
            <h3>Results</h3>
            <ul id="result-list"></ul>
        </div>
    </div>
    <script>
        class ProductNode {
            constructor(name, rating) {
                this.name = name;
                this.rating = rating;
                this.left = null;
                this.right = null;
            }
```

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

    add(name, rating) {
        const newNode = new ProductNode(name, rating);
        if (!this.root) {
            this.root = newNode;
        } else {
            this.insertNode(this.root, newNode);
        }
    }
    insertNode(node, newNode) {
        if (newNode.rating < node.rating) {
            if (!node.left) {
                node.left = newNode;
            } else {
                this.insertNode(node.left, newNode);
            }
        } else {
            if (!node.right) {
                node.right = newNode;
            } else {
                this.insertNode(node.right, newNode);
            }
        }
    }
    findInRange(min, max) {
        const results = [];
        this.findInRangeHelper(this.root, min, max, results);
        return results;
    }
    findInRangeHelper(node, min, max, results) {
        if (!node) return;
        if (node.rating >= min) this.findInRangeHelper(node.left, min, max, results);
        if (node.rating >= min && node.rating <= max) results.push({ name: node.name, rating: node.rating });
        if (node.rating <= max) this.findInRangeHelper(node.right, min, max, results);
    }
    suggestSimilar(rating) {
        const results = [];
        this.suggestSimilarHelper(this.root, rating, results);
        return results;
    }
    suggestSimilarHelper(node, rating, results) {
        if (!node) return;
        if (node.rating === rating) results.push({ name: node.name, rating: node.rating });
        this.suggestSimilarHelper(node.left, rating, results);
        this.suggestSimilarHelper(node.right, rating, results);
    }
}
```

```
    const productBST = new ProductBST();

    function addProduct() {

        const name = document.getElementById('product-name').value;

        const rating = parseFloat(document.getElementById('product-rating').value);

        if (name && rating >= 1 && rating <= 5) {

            productBST.add(name, rating);

            alert(Product "${name}" with rating ${rating} added!);


document.getElementById('product-name').value = '';


document.getElementById('product-rating').value = '';

        } else {

            alert('Please enter a valid product name and rating (1-5).');

        }

    }

    function findProductsInRange() {

        const minRating = parseFloat(document.getElementById('min-rating').value);

        const maxRating = parseFloat(document.getElementById('max-rating').value);

        if (minRating >= 1 && maxRating <= 5 && minRating <= maxRating) {

            const results = productBST.findInRange(minRating, maxRating);

            displayResults(results, Products with ratings between ${minRating} and ${maxRating}:);

        } else {

            alert('Please enter a valid rating range (1-5).');

        }

    }

    function suggestSimilarProducts() {

        const rating = parseFloat(document.getElementById('similar-rating').value);

        if (rating >= 1 && rating <= 5) {

            const results = productBST.suggestSimilar(rating);

            displayResults(results, Products with rating ${rating}:);

        } else {

            alert('Please enter a valid rating (1-5).');

        }

    }

    function displayResults(results, message) {

        const resultList = document.getElementById('result-list');

        resultList.innerHTML = '';

        const header = document.createElement('li');

        header.textContent = message;

        resultList.appendChild(header);

        results.forEach(product => {

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

            li.textContent = ${product.name} (Rating: ${product.rating});

            resultList.appendChild(li);

        });

    }
```

```
    </script>
</body>
</html>
```





E-Commerce Recommendation System

Add Product

Product Name | Rating | Add Product

Find Products by Rating Range

3 | 5 | Find Products

Suggest Similar Products

Rating | Suggest

Results

- Products with ratings between 3 and 5:
- Headphones (Rating: 3)
- Laptop (Rating: 4)
- Tablet (Rating: 4)
- Smartphone (Rating: 5)

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

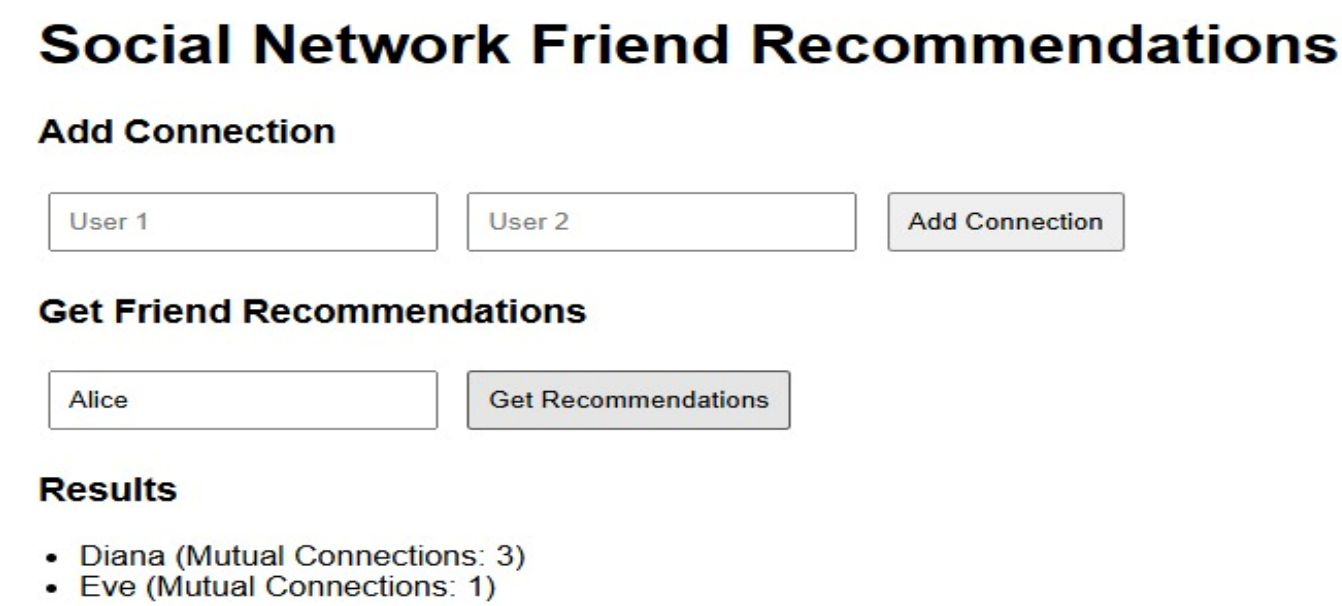
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Social Network Friend Recommendations</title>
    <style>
        body { font-family: Arial, sans-serif; margin: 20px; }
        .container { margin: 20px 0; }
        input, button { margin: 5px; padding: 8px; }
        ul { margin-top: 10px; padding-left: 20px; }
    </style>
</head>
<body>
    <h1>Social Network Friend Recommendations</h1>
    <div class="container">
        <h3>Add Connection</h3>
        <input type="text" id="user1" placeholder="User 1">
        <input type="text" id="user2" placeholder="User 2">
        <button onclick="addConnection()">Add Connection</button>
    </div>
    <div class="container">
        <h3>Get Friend Recommendations</h3>
        <input type="text" id="user" placeholder="Enter User">
        <button onclick="recommendFriends()">Get Recommendations</button>
    </div>
    <div id="results" class="container">
        <h3>Results</h3>
        <ul id="result-list"></ul>
    </div>
    <script>
        class SocialNetwork {
            constructor() {
                this.adjacencyList = new Map();
            }

            addUser(user) {
                if (!this.adjacencyList.has(user)) this.adjacencyList.set(user, []);
            }

            addConnection(user1, user2) {
                this.addUser(user1);
                this.addUser(user2);
                this.adjacencyList.get(user1).push(user2);
                this.adjacencyList.get(user2).push(user1);
            }

            recommend(user) {
```

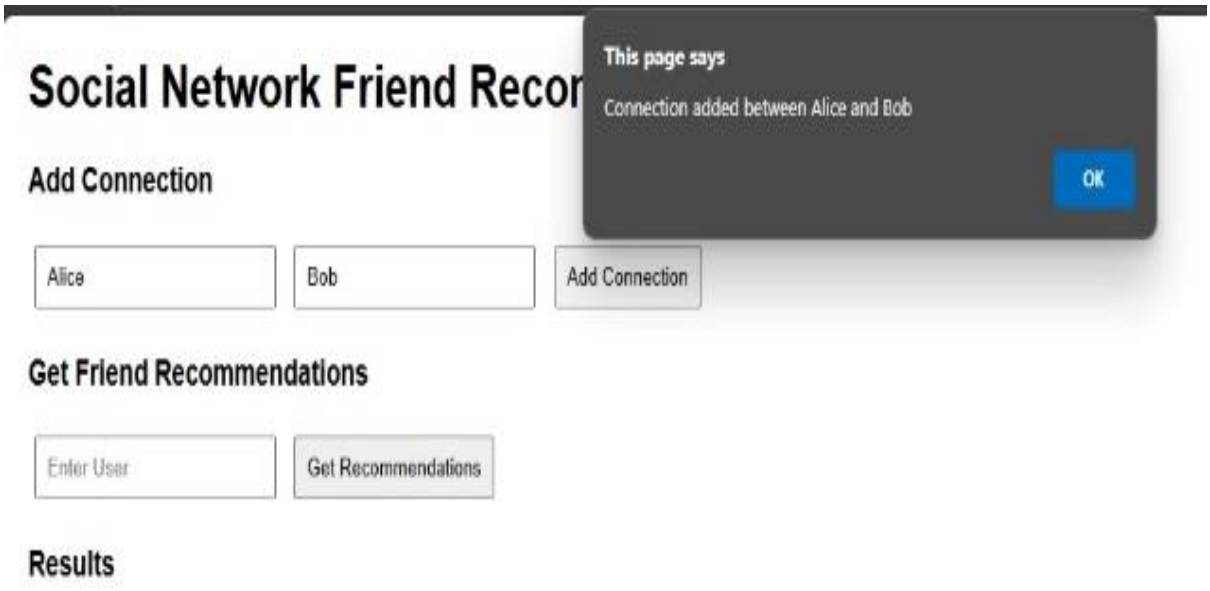
```
        if (!this.adjacencyList.has(user))
{
            return [User "${user}" does
not exist.];
        }
        const visited = new Set();
        const queue = [user];
        const connections = new Map();
        visited.add(user);

        while (queue.length > 0) {
            const currentUser =
queue.shift();
            for (const friend of
this.adjacencyList.get(currentUser)) {
                if (!visited.has(friend)) {
                    visited.add(friend);
                    queue.push(friend);
                }
                if (friend !== user &&
!this.adjacencyList.get(user).includes(frien
d)) {
                    connections.set(friend,
(connections.get(friend) || 0) + 1);
                }
            }
        }
        return
Array.from(connections.entries())
            .sort((a, b) => b[1] - a[1])
            .map(([friend, count]) =>
${friend} (Mutual Connections: ${count}));
    }
}
```

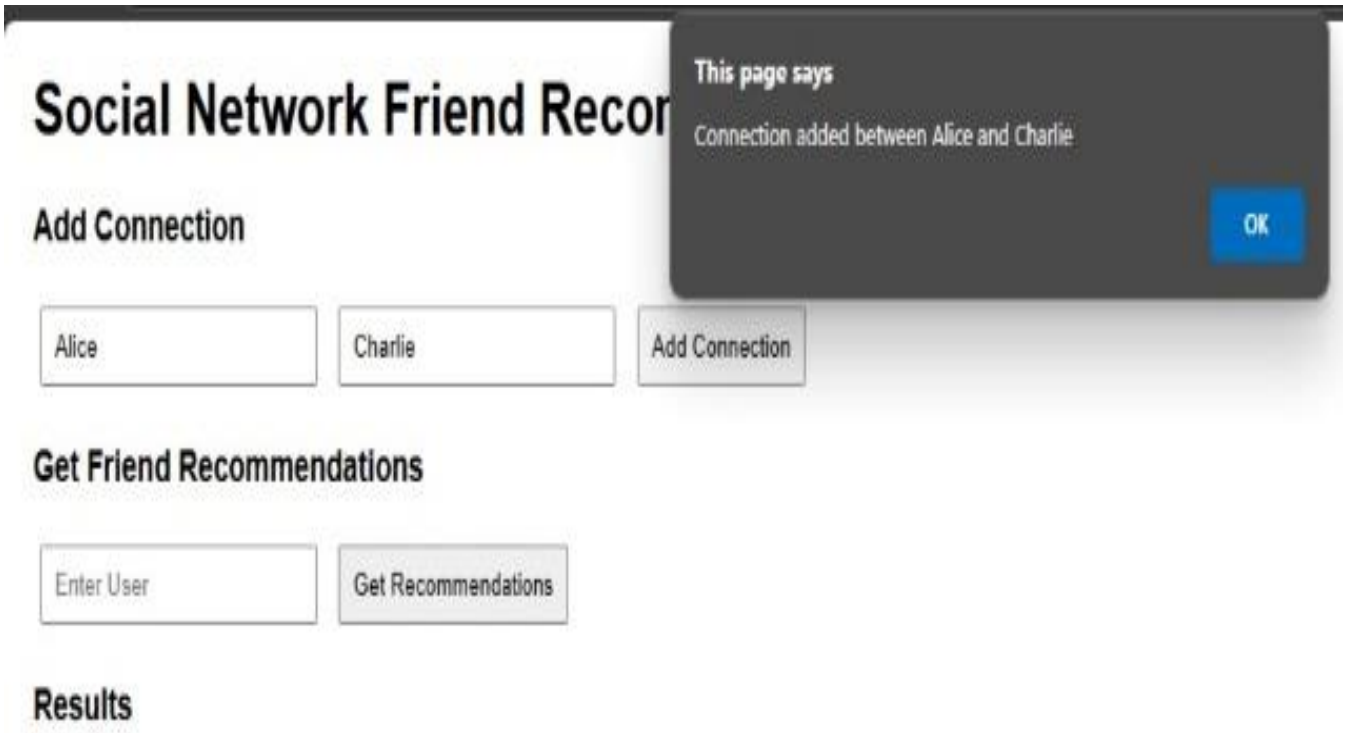
```
const socialNetwork = new
SocialNetwork();

function addConnection() {
    const user1 =
document.getElementById('user1').value.tr
im();
    const user2 =
document.getElementById('user2').value.tr
im();
    if (user1 && user2 && user1 !==
user2) {

socialNetwork.addConnection(user1,
user2);
        alert(Connection added between
${user1} and ${user2});

document.getElementById('user1').value =
'';

document.getElementById('user2').value =
'';
    } else {
        alert('Please enter valid, distinct
user names.');
    }
}

function recommendFriends() {
    const user =
document.getElementById('user').value.tri
m();
    const results =
socialNetwork.recommend(user);
    const resultList =
document.getElementById('result-list');
    resultList.innerHTML = '';
    results.forEach(rec => {
```

```
            const li =
document.createElement('li');
            li.textContent = rec;
            resultList.appendChild(li);
        });
    }
  </script>
</body>
</html>
```

Output:






Social Network Friend Recor
This page says
Connection added between Alice and Charlie
OK
Add Connection
Alice
Charlie
Add Connection
Get Friend Recommendations
Enter User
Get Recommendations
Results

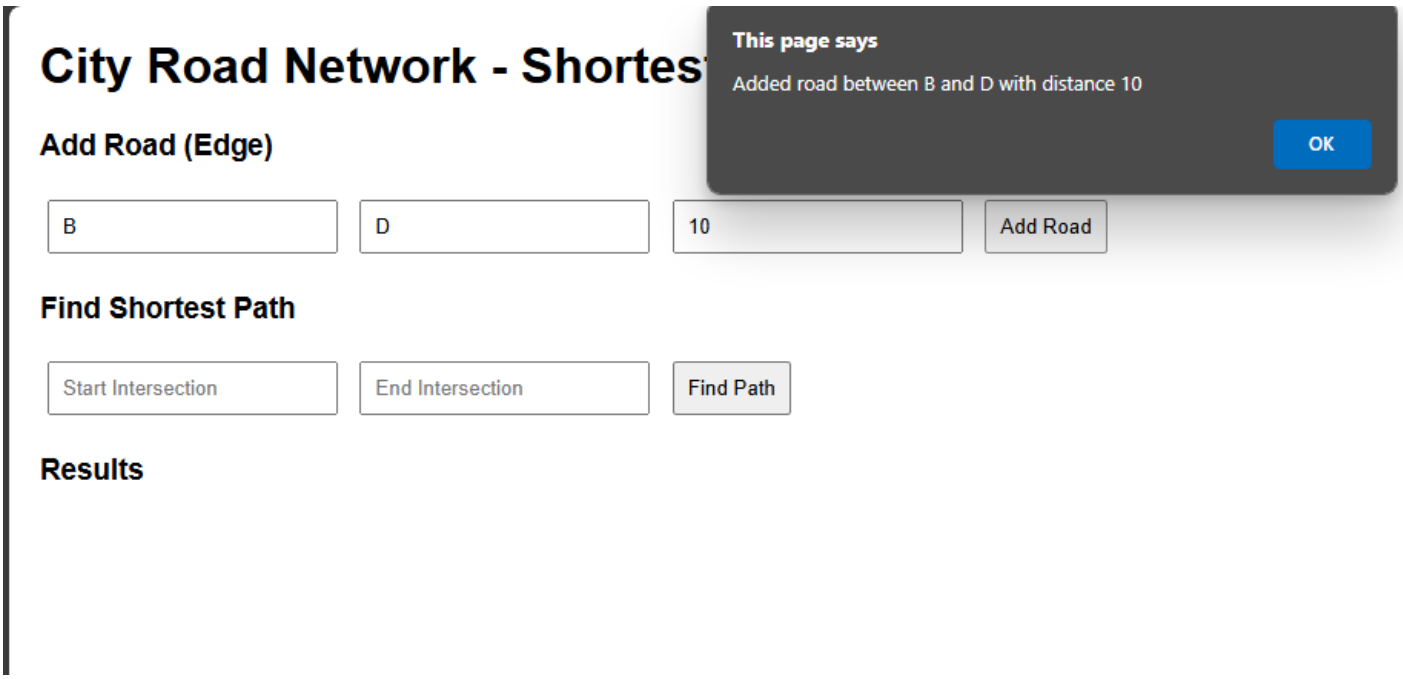
4: Shortest Path in a City (Graph): Given a city represented as a graph with road networks (nodes for intersections, edges for roads), use Dijkstra's algorithm to find the shortest path between any two intersections.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>City Road Network - Shortest Path</title>
    <style>
        body { font-family: Arial, sans-serif; margin: 20px; }
        .container { margin: 20px 0; }
        input, button { margin: 5px; padding: 8px; }
        ul { margin-top: 10px; padding-left: 20px; }
    </style>
</head>
<body>
    <h1>City Road Network - Shortest Path</h1>
    <div class="container">
        <h3>Add Road (Edge)</h3>
        <input type="text" id="intersection1" placeholder="Intersection 1">
        <input type="text" id="intersection2" placeholder="Intersection 2">
        <input type="number" id="distance" placeholder="Distance (weight)" min="1">
        <button onclick="addRoad()">Add Road</button>
    </div>
    <div class="container">
        <h3>Find Shortest Path</h3>
        <input type="text" id="start" placeholder="Start Intersection">
        <input type="text" id="end" placeholder="End Intersection">
        <button onclick="findShortestPath()">Find Path</button>
    </div>
    <div id="results" class="container">
        <h3>Results</h3>
        <ul id="result-list"></ul>
    </div>
    <script>
        class Graph {
            constructor() {
                this.adjacencyList = new Map();
            }
            addNode(node) {
                if (!this.adjacencyList.has(node)) {
                    this.adjacencyList.set(node, []);
                }
            }
            addEdge(node1, node2, weight) {
                this.addNode(node1);
                this.addNode(node2);
                this.adjacencyList.get(node1).push({ node: node2, weight });
                this.adjacencyList.get(node2).push({ node: node1, weight });
```

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


            this.adjacencyList.forEach((_,
node) => {
                distances[node] = node ===
start ? 0 : Infinity;
                previous[node] = null;
                priorityQueue.enqueue(node,
distances[node]);
            });
            while (!priorityQueue.isEmpty())
{
                smallest =
priorityQueue.dequeue().element;
                if (smallest === end) {
                    while (previous[smallest]) {
                        path.push(smallest);
                        smallest =
previous[smallest];
                    }
                    path.push(start);
                    return path.reverse();
                }
                if (smallest ||
distances[smallest] !== Infinity) {
                    for (const neighbor of
this.adjacencyList.get(smallest)) {
                        const alt =
distances[smallest] + neighbor.weight;
                        if (alt <
distances[neighbor.node]) {

distances[neighbor.node] = alt;

previous[neighbor.node] = smallest;

priorityQueue.enqueue(neighbor.node,
alt);
                    }    } } }
            return [];
        }
    }
    class MinPriorityQueue {
        constructor() {
            this.values = [];
        }
        enqueue(element, priority) {
            this.values.push({ element,
priority });
            this.sort();
        }
        dequeue() {
            return this.values.shift();
        }
        isEmpty() {
            return this.values.length === 0;
        }
        sort() {
            this.values.sort((a, b) =>
a.priority - b.priority);
        }
    }
    const cityGraph = new Graph();
    function addRoad() {
```

```
        const intersection1 =
document.getElementById('intersection1').
value.trim();

        const intersection2 =
document.getElementById('intersection2').
value.trim();

        const distance =
parseFloat(document.getElementById('dist
ance').value);

        if (intersection1 && intersection2
&& distance > 0) {


cityGraph.addEdge(intersection1,
intersection2, distance);

            alert(Added road between
${intersection1} and ${intersection2} with
distance ${distance});


document.getElementById('intersection1').
value = '';


document.getElementById('intersection2').
value = '';


document.getElementById('distance').valu
e = '';

        } else {

            alert('Please enter valid
intersections and a positive distance.');

        }

    }

    function findShortestPath() {

        const start =
document.getElementById('start').value.tri
m();

        const end =
document.getElementById('end').value.tri
m();

        if (start && end) {

            const path =
cityGraph.findShortestPath(start, end);

            const resultList =
document.getElementById('result-list');

            resultList.innerHTML = '';

            if (path.length > 0) {

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

                li.textContent = Shortest Path:
${path.join(' → ')};

                resultList.appendChild(li);

            } else {

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

                li.textContent = No path found
between ${start} and ${end}.;

                resultList.appendChild(li);

            }

        } else {

            alert('Please enter valid start
and end intersections.');

        }

    }

  </script>

</body>

</html>
```

Output:



City Road Network - Shortest Path

Add Road (Edge)

Intersection 1 | Intersection 2 | Distance (weight) | Add Road

Find Shortest Path

A | D | Find Path

Results

- Shortest Path: A → C → D

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

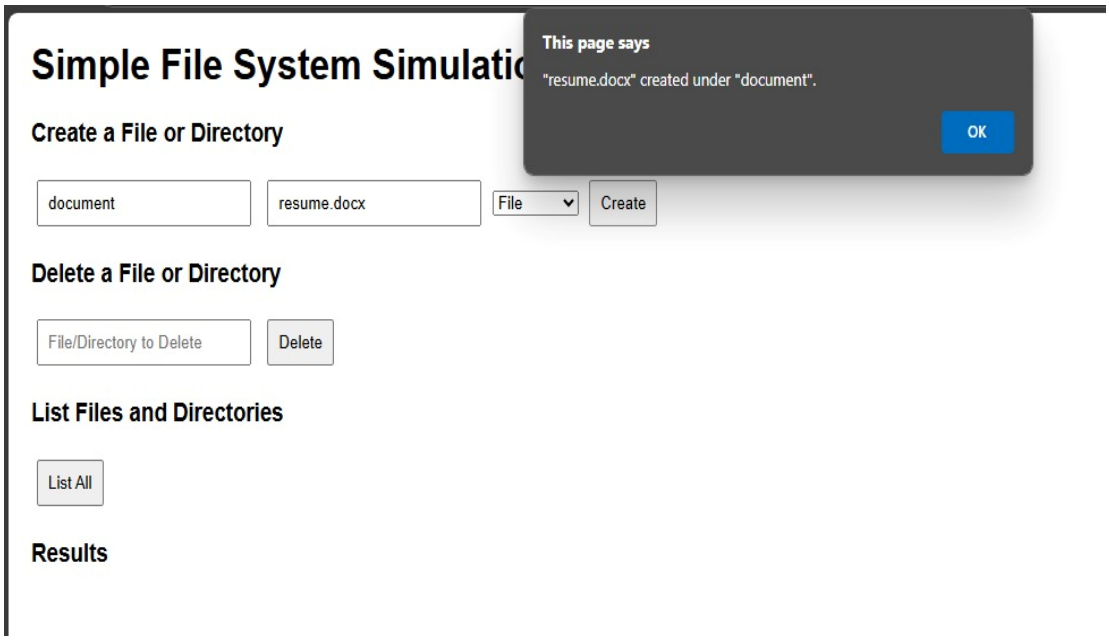
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>File System Simulation</title>
    <style>
        body { font-family: Arial, sans-serif; margin: 20px; }
        .container { margin: 20px 0; }
        input, button { margin: 5px; padding: 8px; }
    </style>
</head>
<body>
    <h1>Simple File System Simulation</h1>
    <div class="container">
        <h3>Create a File or Directory</h3>
        <input type="text" id="parentDir" placeholder="Parent Directory (e.g., root)">
        <input type="text" id="itemName" placeholder="New File/Directory Name">
        <select id="isFile">
            <option value="false">Directory</option>
            <option value="true">File</option>
        </select>
        <button onclick="createItem()">Create</button>
    </div>
    <div class="container">
        <h3>Delete a File or Directory</h3>
        <input type="text" id="deleteItem" placeholder="File/Directory to Delete">
        <button onclick="deleteItem()">Delete</button>
    </div>
    <div class="container">
        <h3>List Files and Directories</h3>
        <button onclick="listFiles()">List All</button>
    </div>
    <div id="results" class="container">
        <h3>Results</h3>
        <ul id="result-list"></ul>
    </div>
    <script>
        class Node {
            constructor(name, isFile = false) {
                this.name = name;
                this.isFile = isFile;
                this.children = [];
            }
        }
        class FileSystem {
            constructor() {
                this.root = new Node("root");
            }
            find(parentName) {
                if (parentName === "root") return this.root;
```

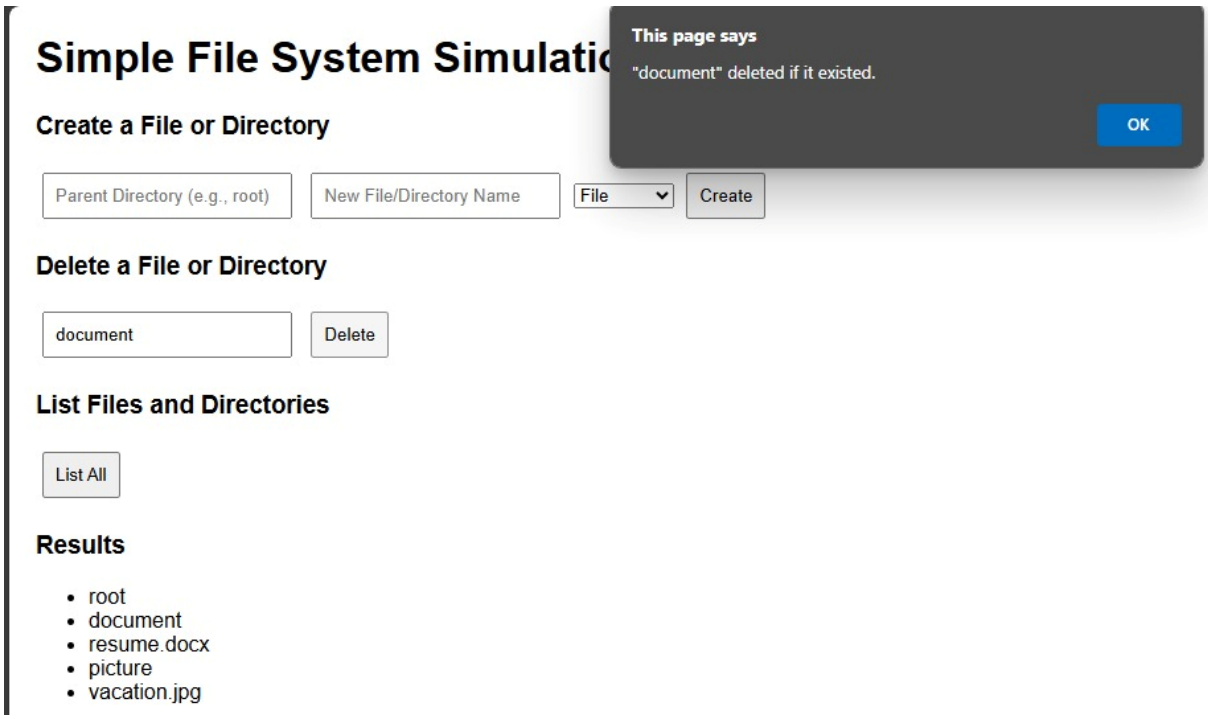
```
        const parent = this._find(this.root, parentName);
        return parent ? parent : null;
    }
    _find(node, name) {
        if (node.name === name) return node;
        for (const child of node.children) {
            const found = this._find(child, name);
            if (found) return found;
        }
        return null;
    }
    create(parentName, itemName, isFile = false) {
        const parent = this.find(parentName);
        if (!parent) return Parent directory "${parentName}" not found.;
        const newItem = new Node(itemName, isFile);
        parent.children.push(newItem);
        return "${itemName}" created under "${parentName}".;
    }

    delete(itemName) {
        const deleteRecursively = (parent, name) => {
            parent.children = parent.children.filter(child => {
                if (child.name === name) return false;
                deleteRecursively(child, name);
                return true;
            });
        };
        deleteRecursively(this.root, itemName);
        return "${itemName}" deleted if it existed.;
    }
    list(node = this.root) {
        const result = [];
        result.push(node.name);
        node.children.forEach(child => result.push(...this.list(child)));
        return result;
    }
}
const fileSystem = new FileSystem();

function createItem() {
    const parentDir = document.getElementById("parentDir").value.trim();
    const itemName = document.getElementById("itemName").value.trim();
    const isFile = document.getElementById("isFile").value === "true";
    if (parentDir && itemName) {
        const message = fileSystem.create(parentDir, itemName, isFile);
        alert(message);

document.getElementById("parentDir").value = '';
```

```
document.getElementById("itemName").value = '';
        } else {
            alert("Please provide valid parent directory and item name.");
        }
    }

    function deleteItem() {
        const itemName = document.getElementById("deleteItem").value.trim();
        if (itemName) {
            const message = fileSystem.delete(itemName);
            alert(message);

document.getElementById("deleteItem").value = '';
        } else {
            alert("Please provide a valid item name to delete.");
        }
    }
    function listFiles() {
        const resultList = document.getElementById("result-list");
        resultList.innerHTML = '';
        const result = fileSystem.list();
        result.forEach(item => {
            const li = document.createElement("li");
            li.textContent = item;
            resultList.appendChild(li);
        });
    }
  </script>
</body>
</html>
```

Output:


Simple File System Simulation
Create a File or Directory
Parent Directory (e.g., root)
New File/Directory Name
File
Create
Delete a File or Directory
File/Directory to Delete
Delete
List Files and Directories
List All
Results
root
picture
vacation.jpg


Simple File System Simulatic
This page says
"resume.docx" created under "document".
OK
Create a File or Directory
document
resume.docx
File
Create
Delete a File or Directory
File/Directory to Delete
Delete
List Files and Directories
List All
Results


Simple File System Simulatic
This page says
"document" deleted if it existed.
OK
Create a File or Directory
Parent Directory (e.g., root)
New File/Directory Name
File
Create
Delete a File or Directory
document
Delete
List Files and Directories
List All
Results
root
document
resume.docx
picture
vacation.jpg


Simple File System Simulation
Create a File or Directory
Parent Directory (e.g., root)
New File/Directory Name
File
Create
Delete a File or Directory
File/Directory to Delete
Delete
List Files and Directories
List All
Results
root
document
resume.docx
picture
vacation.jpg

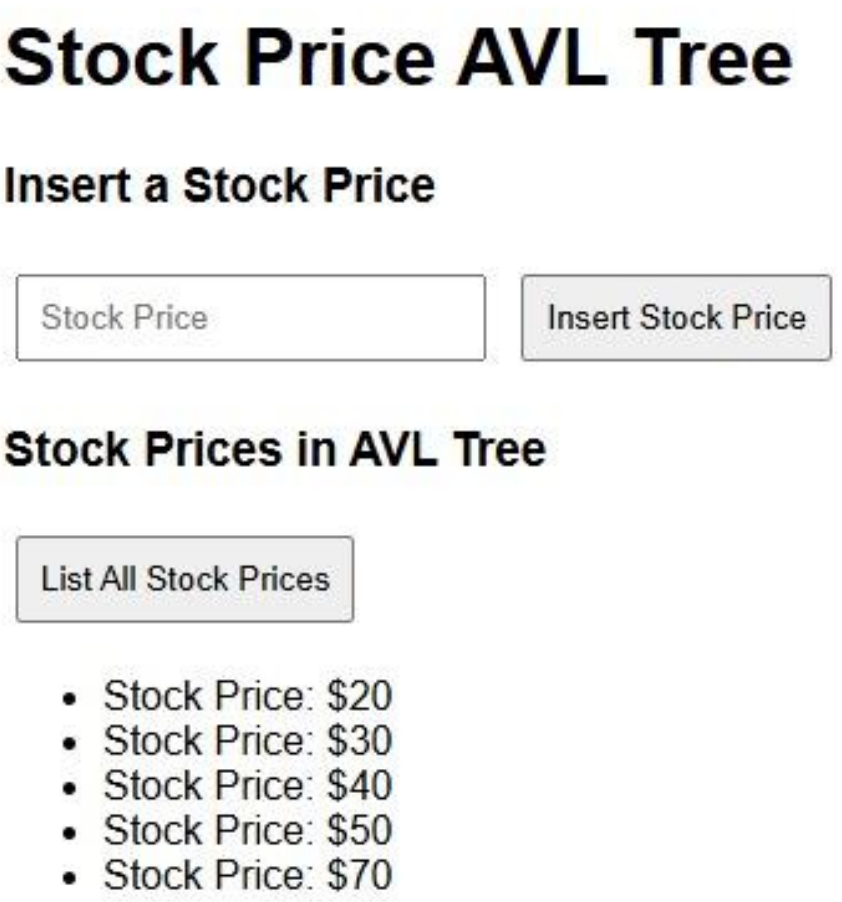
6: AVL Tree for Stock Price Management: Use an AVL tree to maintain stock prices. Ensure that after each insertion, the tree remains balanced by performing rotations.

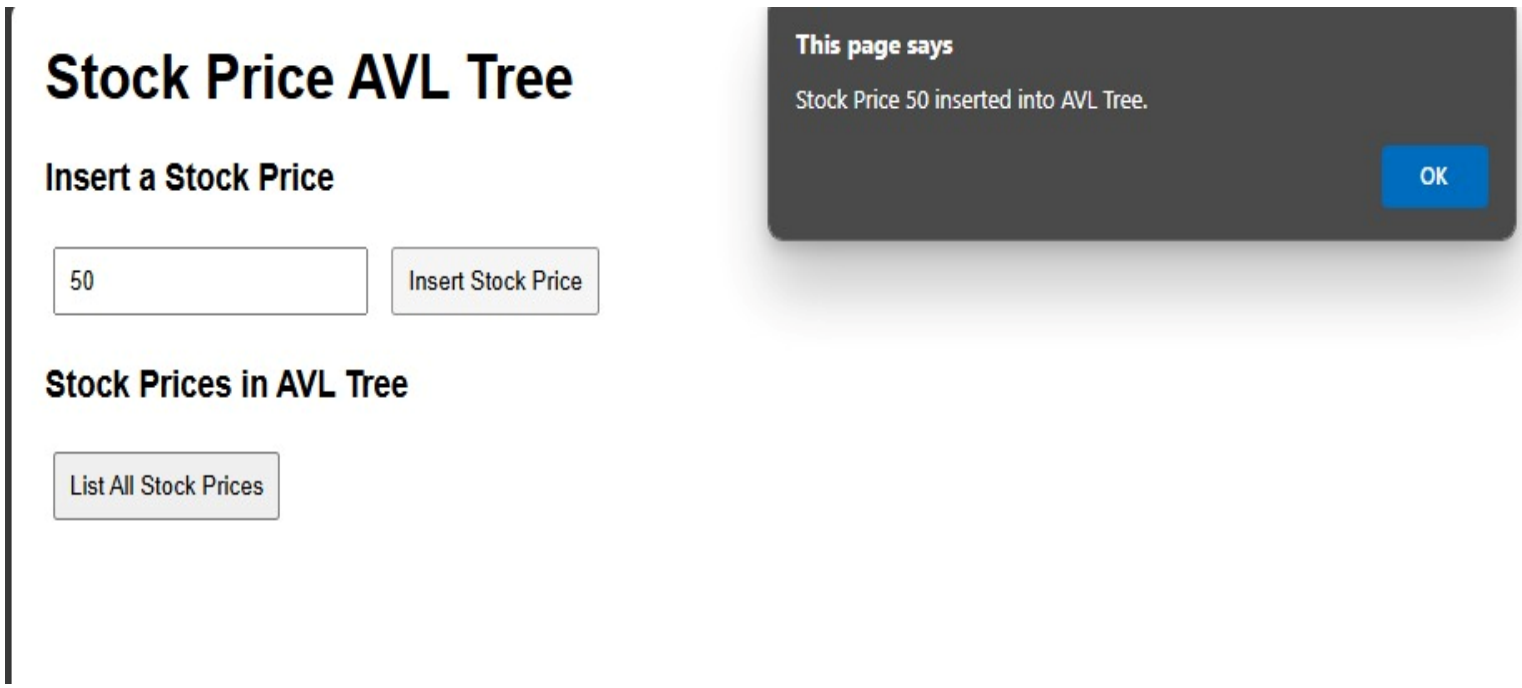
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Stock Price AVL Tree</title>
    <style>
        body { font-family: Arial, sans-serif; margin: 20px; }
        .container { margin: 20px 0; }
        input, button { margin: 5px; padding: 8px; }
    </style>
</head>
<body>
    <h1>Stock Price AVL Tree</h1>
    <div class="container">
        <h3>Insert a Stock Price</h3>
        <input type="number" id="stockPrice" placeholder="Stock Price">
        <button onclick="insertStockPrice()">Insert Stock Price</button>
    </div>
    <div class="container">
        <h3>Stock Prices in AVL Tree</h3>
        <button onclick="listStockPrices()">List All Stock Prices</button>
        <ul id="stock-list"></ul>
    </div>
    <script>
        class Node {
            constructor(price) {
                this.price = price;
                this.left = null;
                this.right = null;
                this.height = 1;
            }
        }

        class AVLTree {
            constructor() {
                this.root = null;
            }
            getHeight(node) {
                return node ? node.height : 0;
            }
            getBalance(node) {
                return node ? this.getHeight(node.left) - this.getHeight(node.right) : 0;
            }
            rightRotate(y) {
                const x = y.left;
                const T2 = x.right;
                x.right = y;
                y.left = T2;
                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;
    }
    leftRotate(x) {
        const y = x.right;
        const T2 = y.left;
        y.left = x;
        x.right = T2;
        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;
    }
    insert(node, price) {
        if (!node) return new
Node(price);
        if (price < node.price) {
            node.left =
this.insert(node.left, price);
        } else if (price > node.price) {
            node.right =
this.insert(node.right, price);
        } else {
            return node; // Duplicate
prices are not allowed
        }
        node.height =
Math.max(this.getHeight(node.left),
this.getHeight(node.right)) + 1;
        const balance =
this.getBalance(node);
        // Left Left Case
        if (balance > 1 && price <
node.left.price) {
            return this.rightRotate(node);
        }
        // Right Right Case
        if (balance < -1 && price >
node.right.price) {
            return this.leftRotate(node);
        }
        // Left Right Case
        if (balance > 1 && price >
node.left.price) {
            node.left =
this.leftRotate(node.left);
            return this.rightRotate(node);
        }
        // Right Left Case
        if (balance < -1 && price <
node.right.price) {
            node.right =
this.rightRotate(node.right);
            return this.leftRotate(node);
        }
        return node;
    }
    inOrder(node, result = []) {
        if (node) {
            this.inOrder(node.left, result);
            result.push(node.price);
            this.inOrder(node.right,
result);
        }
        return result;
    }
    insertStockPrice(price) {
```

```
            this.root = this.insert(this.root,
price);
        }
        listStockPrices() {
            return this.inOrder(this.root);
        }
    }
    const avlTree = new AVLTree();
    function insertStockPrice() {
        const price =
parseFloat(document.getElementById("sto
ckPrice").value);
        if (!isNaN(price)) {
            avlTree.insertStockPrice(price);
            alert(Stock Price ${price}
inserted into AVL Tree.);

document.getElementById("stockPrice").v
alue = '';
        } else {
            alert("Please enter a valid stock
price.");
        }
    }
    function listStockPrices() {
        const prices =
avlTree.listStockPrices();
        const resultList =
document.getElementById("stock-list");
        resultList.innerHTML = '';
        prices.forEach(price => {
            const li =
document.createElement("li");
            li.textContent = Stock Price:
$${price};
            resultList.appendChild(li);
        });
    }
  </script>
</body>
</html>
```

Output:


Stock Price AVL Tree
Insert a Stock Price
Stock Price
Insert Stock Price
Stock Prices in AVL Tree
List All Stock Prices
Stock Price: $20
Stock Price: $30
Stock Price: $40
Stock Price: $50
Stock Price: $70

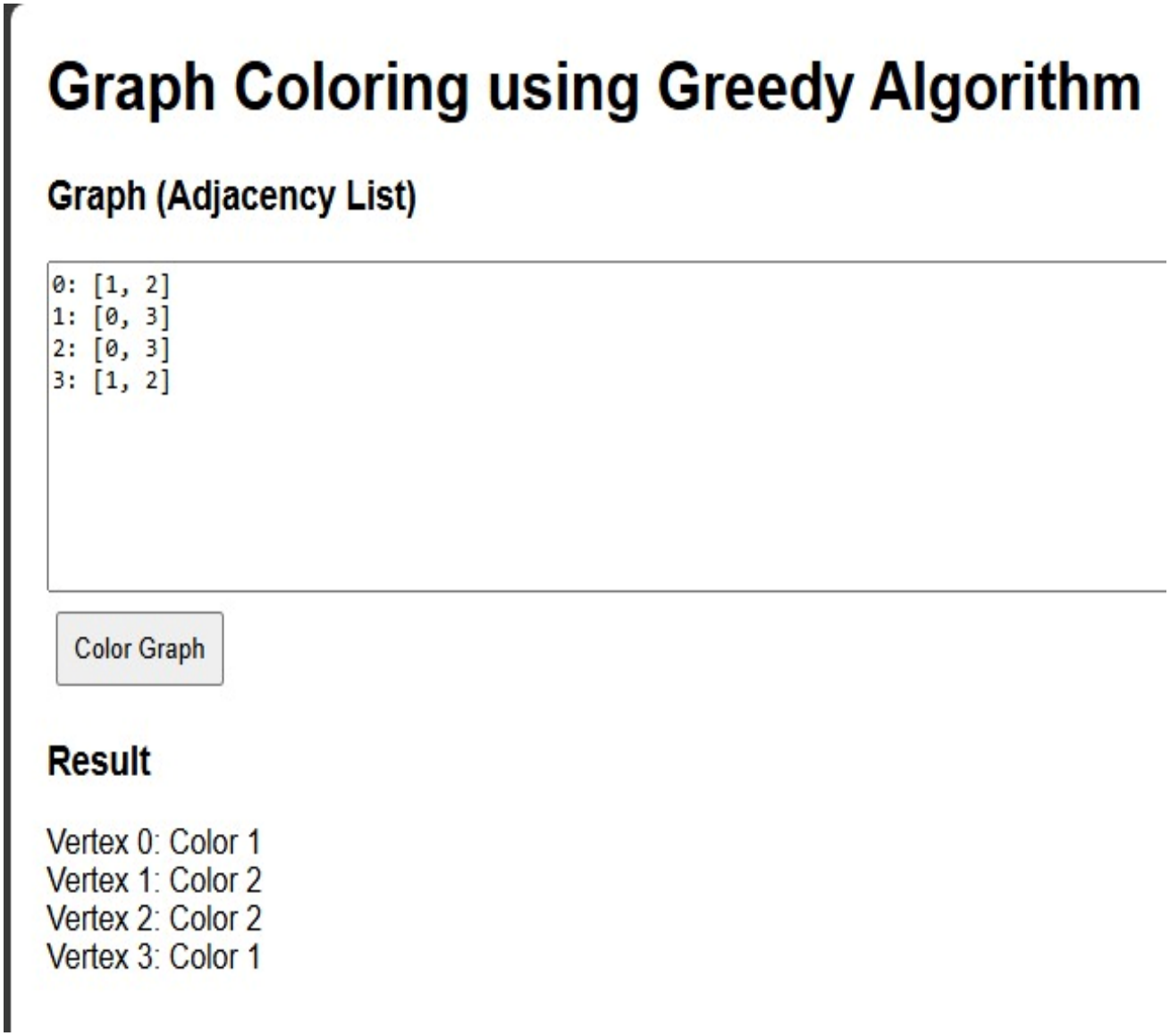

Stock Price AVL Tree
Insert a Stock Price
50
Insert Stock Price
Stock Prices in AVL Tree
List All Stock Prices
This page says
Stock Price 50 inserted into AVL Tree.
OK

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

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Graph Coloring using Greedy Algorithm</title>
    <style>
        body { font-family: Arial, sans-serif; margin: 20px; }
        .container { margin: 20px 0; }
        input, button { margin: 5px; padding: 8px; }
        textarea { width: 100%; }
    </style>
</head>
<body>
    <h1>Graph Coloring using Greedy Algorithm</h1>
    <div class="container">
        <h3>Graph (Adjacency List)</h3>
        <textarea id="graph" rows="10" placeholder="Enter graph as adjacency list (e.g., 0: [1, 2], 1: [0, 3], 2: [0, 3], 3: [1, 2])"></textarea>
        <br>
        <button onclick="colorGraph()">Color Graph</button>
    </div>
    <div class="container">
        <h3>Result</h3>
        <div id="result"></div>
    </div>
    <script>
        function parseGraphInput(input) {
            const graph = {};
            const lines = input.trim().split('\n');
            for (const line of lines) {
                const parts = line.split(':');
                if (parts.length === 2) {
                    const node = parseInt(parts[0].trim());
                    const neighbors = parts[1].trim().slice(1, -1).split(',').map(x => parseInt(x.trim()));
                    graph[node] = neighbors;
                }
            }
            return graph;
        }
        function greedyColoring(graph) {
            const colors = {};
            const vertices = Object.keys(graph).map(v => parseInt(v));
            for (const vertex of vertices) {
                const neighbors = graph[vertex];
                const assignedColors = new Set();
                for (const neighbor of neighbors) {
                    if (colors[neighbor] !== undefined) {
                        assignedColors.add(colors[neighbor]);
                    }
```

```
            }
            let color = 1;
            while
(assignedColors.has(color)) {
                color++;
            }
            colors[vertex] = color;
        }
        return colors;
    }

    function displayResult(colors) {
        const resultDiv =
document.getElementById("result");
        const result = [];
        for (const vertex in colors) {
            result.push(Vertex ${vertex}:
Color ${colors[vertex]});
        }
        resultDiv.innerHTML =
result.join("<br>");
    }
    function colorGraph() {
        const graphInput =
document.getElementById("graph").value.t
rim();
        const graph =
parseGraphInput(graphInput);
        if (Object.keys(graph).length ===
0) {
            alert("Invalid graph input. Please
follow the correct format.");
            return;
        }
        const colors =
greedyColoring(graph);
            displayResult(colors);
        }
    </script>
</body>
</html>
```

Output:


Graph Coloring using Greedy Algorithm
Graph (Adjacency List)
0: [1, 2]
1: [0, 3]
2: [0, 3]
3: [1, 2]
Color Graph
Result
Vertex 0: Color 1
Vertex 1: Color 2
Vertex 2: Color 2
Vertex 3: Color 1

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.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Kruskal's Algorithm - Minimum Spanning Tree (MST)</title>
    <style>
        body { font-family: Arial, sans-serif; margin: 20px; }
        .container { margin: 20px 0; }
        input, button { margin: 5px; padding: 8px; }
        textarea { width: 100%; }
    </style>
</head>
<body>
    <h1>Kruskal's Algorithm - Minimum Spanning Tree (MST)</h1>
    <div class="container">
        <h3>Graph (City Connections)</h3>
        <textarea id="edges" rows="10" placeholder="Enter edges as (u, v, cost), e.g., (0, 1, 10), (1, 2, 15)"></textarea>
        <br>
        <button onclick="findMST()">Find MST</button>
    </div>
    <div class="container">
        <h3>Result</h3>
        <div id="result"></div>
    </div>
    <script>
        function find(parent, i) {
            if (parent[i] === i) return i;
            parent[i] = find(parent, parent[i]); // Path compression
            return parent[i];
        }
        function union(parent, rank, x, y) {
            const rootX = find(parent, x);
            const rootY = find(parent, y);
            if (rootX !== rootY) {
                if (rank[rootX] > rank[rootY]) {
                    parent[rootY] = rootX;
                } else if (rank[rootX] < rank[rootY]) {
                    parent[rootX] = rootY;
                } else {
                    parent[rootY] = rootX;
                    rank[rootX]++;
                } } }
        function kruskal(numCities, edges) {
            const parent = [];
            const rank = [];
            for (let i = 0; i < numCities; i++) {
                parent[i] = i;
                rank[i] = 0;
            }
            edges.sort((a, b) => a[2] - b[2]);
            const mst = [];
            let mstCost = 0;
```

```
        for (const edge of edges) {
            const [u, v, cost] = edge;
            if (find(parent, u) !== find(parent, v)) {
                mst.push(edge);
                mstCost += cost;
                union(parent, rank, u, v);
            }
        }
        return { mst, mstCost };
    }
    function parseGraphInput(input) {
        const edges = [];
        const lines = input.trim().split('\n');
        for (const line of lines) {
            const match = line.match(/\((\d+),\s*(\d+),\s*(\d+)\)/);
            if (match) {

edges.push([parseInt(match[1]), parseInt(match[2]), parseInt(match[3])]);
            }
        }
        return edges;
    }
    function getNumCities(edges) {
        const cities = new Set();
        for (const edge of edges) {
            cities.add(edge[0]);
            cities.add(edge[1]);
        }
        return cities.size;
    }
    function displayResult(mst) {
        const resultDiv = document.getElementById("result");
        if (mst.mst.length === 0) {
            resultDiv.innerHTML = "No MST can be formed!";
            return;
        }
        let mstResult = MST Cost: ${mst.mstCost}<br>Edges in MST:<br>;
        mst.mst.forEach(edge => {
            mstResult += (${edge[0]}, ${edge[1]}) with cost ${edge[2]}<br>;
        });
        resultDiv.innerHTML = mstResult;
    }
    function findMST() {
        const edgesInput = document.getElementById("edges").value.trim();
        const edges = parseGraphInput(edgesInput);
        if (edges.length === 0) {
            alert("Invalid input. Please enter edges in the correct format.");
            return;
        }
        const numCities = getNumCities(edges);
        const mst = kruskal(numCities, edges);
        displayResult(mst);
    }
  </script>
</body>
</html>
```

Output:

Kruskal's Algorithm - Minimum Spanning Tree (MST)

Graph (City Connections)

```
(0, 1, 10)
(0, 2, 6)
(0, 3, 5)
(1, 3, 15)
(2, 3, 4)
```

Find MST

Result

MST Cost: 19
Edges in MST:
(2, 3) with cost 4
(0, 3) with cost 5
(0, 1) with cost 10

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.

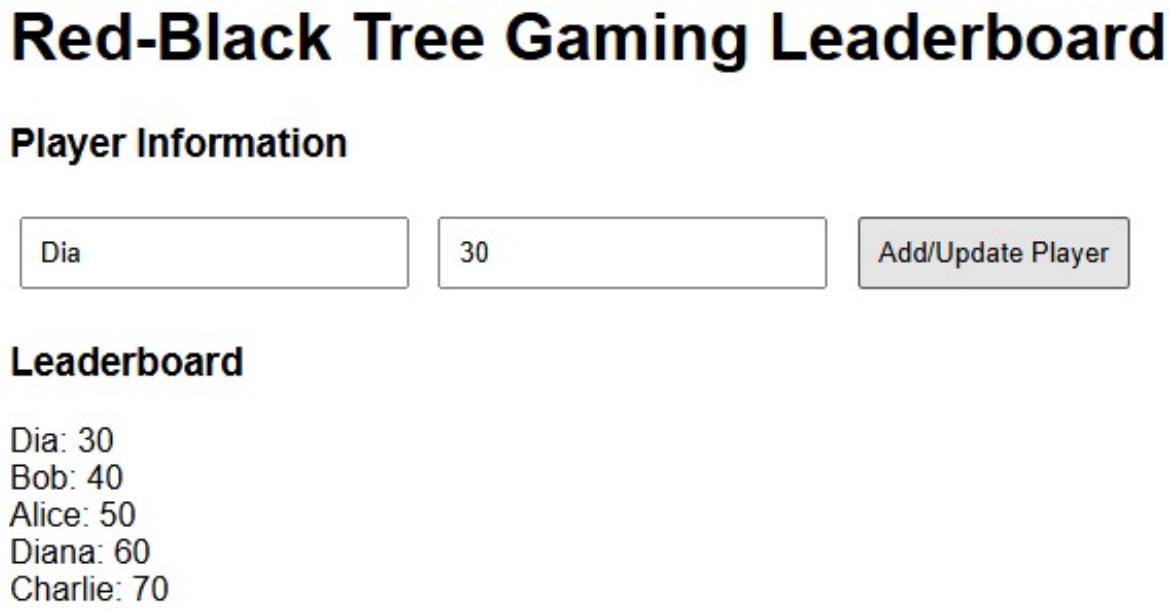
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Red-Black Tree Gaming Leaderboard</title>
    <style>
        body { font-family: Arial, sans-serif; margin: 20px; }
        .container { margin: 20px 0; }
        input, button { margin: 5px; padding: 8px; }
        textarea { width: 100%; }
    </style>
</head>
<body>
    <h1>Red-Black Tree Gaming Leaderboard</h1>
    <div class="container">
        <h3>Player Information</h3>
        <input type="text" id="playerName" placeholder="Player Name">
        <input type="number" id="playerScore" placeholder="Player Score">
        <button onclick="addPlayer()">Add/Update Player</button>
    </div>
    <div class="container">
        <h3>Leaderboard</h3>
        <div id="leaderboard"></div>
    </div>
    <script>
        class Node {
            constructor(playerName, playerScore, color = 'RED') {
                this.playerName = playerName;
                this.playerScore = playerScore;
                this.color = color;
                this.left = null;
                this.right = null;
                this.parent = null;
            }
        }
        class RedBlackTree {
            constructor() {
                this.NIL = new Node(null, null, 'BLACK');
                this.root = this.NIL;
            }

            rightRotate(y) {
                const x = y.left;
                y.left = x.right;
                if (x.right !== this.NIL) {
                    x.right.parent = y;
                }
                x.parent = y.parent;
                if (y.parent === this.NIL) {
                    this.root = x;
                } else if (y === y.parent.right) {
```

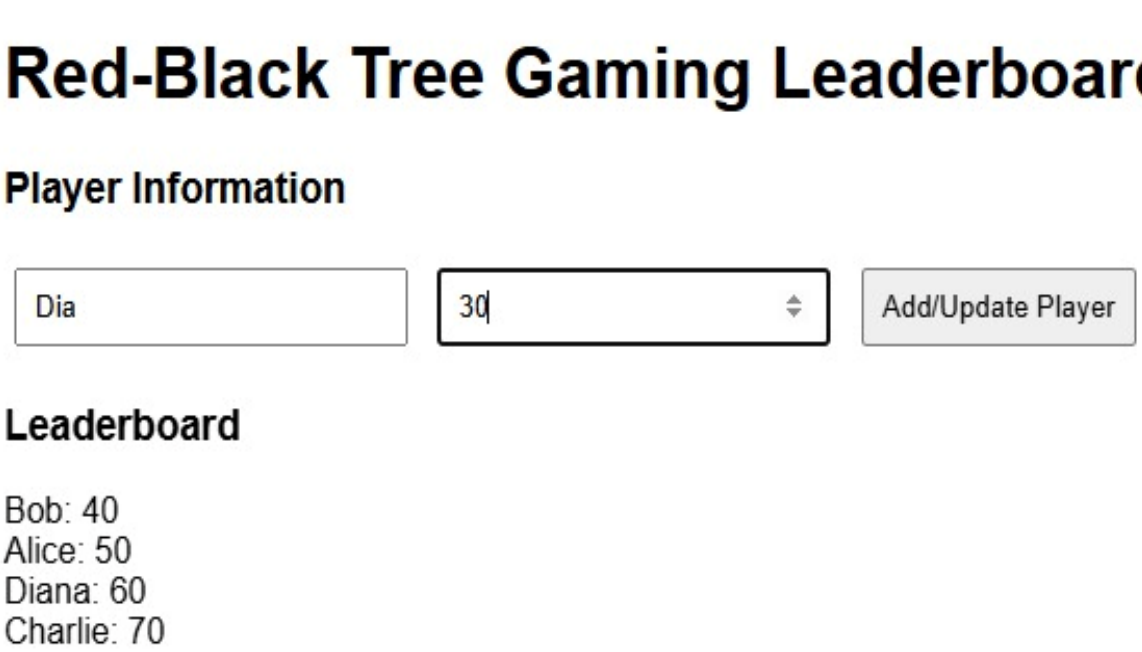
```
            y.parent.right = x;
        } else {
            y.parent.left = x;
        }
        x.right = y;
        y.parent = x;
    }
    leftRotate(x) {
        const y = x.right;
        x.right = y.left;
        if (y.left !== this.NIL) {
            y.left.parent = x;
        }
        y.parent = x.parent;
        if (x.parent === this.NIL) {
            this.root = y;
        } else if (x === x.parent.left) {
            x.parent.left = y;
        } else {
            x.parent.right = y;
        }
        y.left = x;
        x.parent = y;
    }
    fixInsert(z) {
        while (z.parent.color === 'RED')
{
            if (z.parent ===
z.parent.parent.left) {
                const y =
z.parent.parent.right;
                if (y.color === 'RED') {
                    z.parent.color =
'BLACK';
                    y.color = 'BLACK';
                    z.parent.parent.color =
'RED';
                    z = z.parent.parent;
                } else {
                    if (z === z.parent.right) {
                        z = z.parent;
                        this.leftRotate(z);
                    }
                    z.parent.color =
'BLACK';
                    z.parent.parent.color =
'RED';

this.rightRotate(z.parent.parent);
                }
            } else {
                const y =
z.parent.parent.left;
                if (y.color === 'RED') {
                    z.parent.color =
'BLACK';
                    y.color = 'BLACK';
                    z.parent.parent.color =
'RED';
                    z = z.parent.parent;
                } else {
                    if (z === z.parent.left) {
                        z = z.parent;
                        this.rightRotate(z);
                    }
                    z.parent.color =
'BLACK';
                    z.parent.parent.color =
'RED';
```

```
this.leftRotate(z.parent.parent);
                    }
                }
                if (z === this.root) break;
            }
            this.root.color = 'BLACK';
        }
        insert(playerName, playerScore) {
            let z = new Node(playerName,
playerScore, 'RED');
            let y = this.NIL;
            let x = this.root;

            while (x !== this.NIL) {
                y = x;
                if (playerScore <
x.playerScore) {
                    x = x.left;
                } else if (playerScore >
x.playerScore) {
                    x = x.right;
                } else {
                    x.playerScore =
playerScore;
                    return;
                }
            }
            z.parent = y;
            if (y === this.NIL) {
                this.root = z;
            } else if (playerScore <
y.playerScore) {
                y.left = z;
            } else {
                y.right = z;
            }
            z.left = this.NIL;
            z.right = this.NIL;
            this.fixInsert(z);
        }
        inOrderTraversal(node, result = [])
{
            if (node !== this.NIL) {

this.inOrderTraversal(node.left, result);

result.push(${node.playerName}:
${node.playerScore});

this.inOrderTraversal(node.right, result);
            }
            return result;
        }
    }
    const leaderboardTree = new
RedBlackTree();
    function addPlayer() {
        const playerName =
document.getElementById('playerName').
value.trim();
        const playerScore =
parseInt(document.getElementById('playe
rScore').value.trim(), 10);
        if (!playerName ||
isNaN(playerScore)) {
            alert("Please enter valid player
name and score.");
            return;
        }
```

```
leaderboardTree.insert(playerName,
playerScore);
            displayLeaderboard();
        }
        function displayLeaderboard() {
            const leaderboardDiv =
document.getElementById('leaderboard');
            const leaderboard =
leaderboardTree.inOrderTraversal(leaderb
oardTree.root);
            leaderboardDiv.innerHTML =
leaderboard.length > 0 ?
leaderboard.join("<br>") : "Leaderboard is
empty!";
        }
    </script>
</body>
</html>
```

Output:


Red-Black Tree Gaming Leaderboard
Player Information
Dia
30
Add/Update Player
Leaderboard
Dia: 30
Bob: 40
Alice: 50
Diana: 60
Charlie: 70

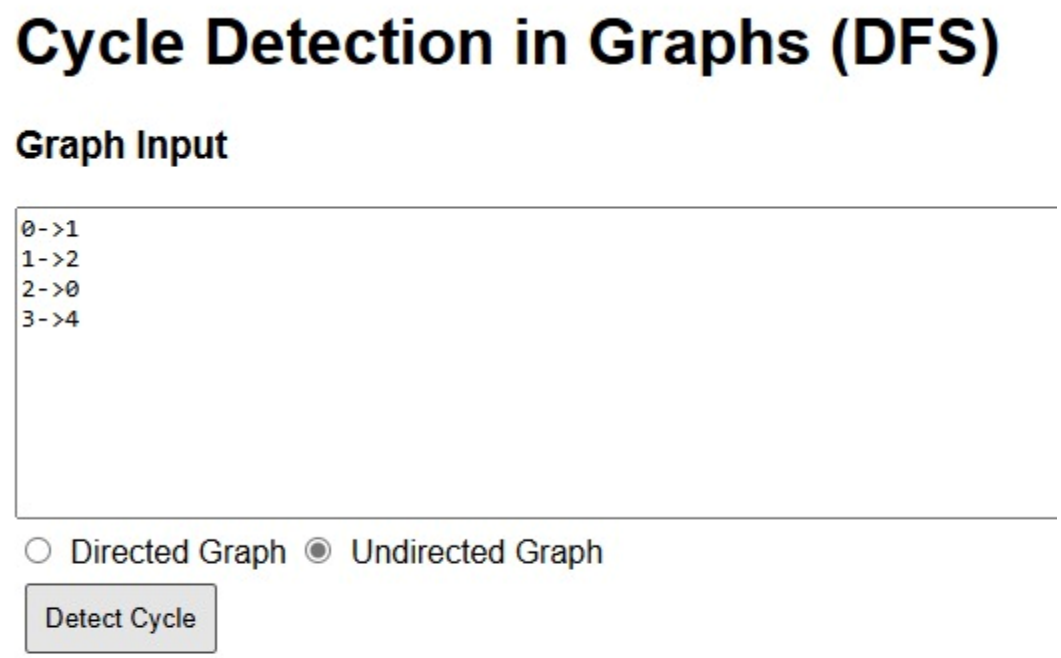

Red-Black Tree Gaming Leaderboard
Player Information
Dia
30
Add/Update Player
Leaderboard
Bob: 40
Alice: 50
Diana: 60
Charlie: 70

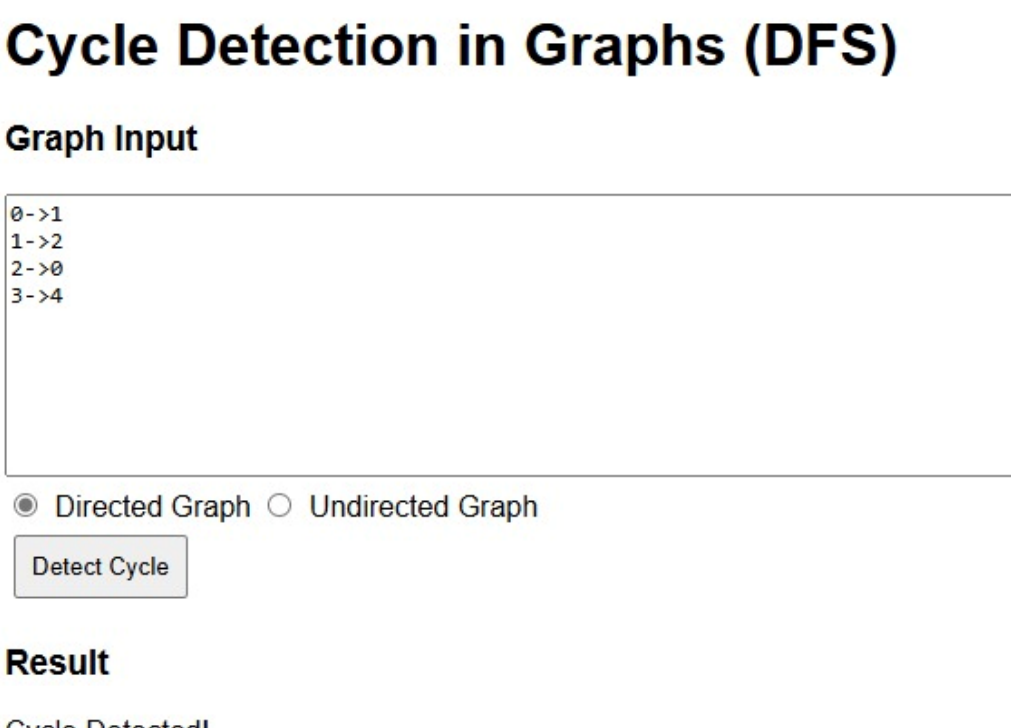
10: Cycle Detection in Graph: Implement a graph traversal algorithm (DFS) to detect cycles in a directed and undirected graph, simulating dependencies between software modules.

```
<!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 Graphs (DFS)</title>
    <style>
        body { font-family: Arial, sans-serif; margin: 20px; }
        .container { margin: 20px 0; }
        input, button { margin: 5px; padding: 8px; }
        textarea { width: 100%; }
    </style>
</head>
<body>
    <h1>Cycle Detection in Graphs (DFS)</h1>
    <div class="container">
        <h3>Graph Input</h3>
        <textarea id="graphInput" rows="10" placeholder="Enter graph edges, one per line: e.g., 0->1, 1->2"></textarea>
        <br>
        <label>
            <input type="radio" name="graphType" value="directed" checked> Directed Graph
        </label>
        <label>
            <input type="radio" name="graphType" value="undirected"> Undirected Graph
        </label>
        <br>
        <button onclick="detectCycle()">Detect Cycle</button>
    </div>
    <div class="container">
        <h3>Result</h3>
        <div id="result"></div>
    </div>
    <script>
        function parseGraphInput(input) {
            const graph = {};
            const edges = input.split('\n').map(line => line.trim()).filter(line => line);
            edges.forEach(edge => {
                const [start, end] = edge.split('->').map(e => parseInt(e.trim()));
                if (!graph[start]) graph[start] = [];
                graph[start].push(end);
            });
            return graph;
        }
        function dfsDirected(graph, node, visited, recStack) {
            visited[node] = 1;
            recStack[node] = 1;
            if (graph[node]) {
                for (let neighbor of graph[node]) {
```

```
                if (!visited[neighbor] &&
dfsDirected(graph, neighbor, visited,
recStack)) {
                    return true;
                } else if (recStack[neighbor]) {
                    return true;
                }
            }
        }
        recStack[node] = 0;
        return false;
    }
    function dfsUndirected(graph, node,
visited, parent) {
        visited[node] = true;
        if (graph[node]) {
            for (let neighbor of graph[node])
{
                if (!visited[neighbor]) {
                    if (dfsUndirected(graph,
neighbor, visited, node)) {
                        return true;
                    }
                } else if (neighbor !== parent)
{
                    return true;
                }
            }
        }
        return false;
    }
    function detectCycle() {
        const graphInput =
document.getElementById('graphInput').va
lue.trim();
        const graph =
parseGraphInput(graphInput);
        const isDirected =
document.querySelector('input[name="gra
phType"]:checked').value === 'directed';
        const resultDiv =
document.getElementById('result');
        let cycleDetected = false;
        if (isDirected) {
            const visited = {};
            const recStack = {};
            for (const node in graph) {
                if (!visited[node]) {
                    if (dfsDirected(graph,
parseInt(node), visited, recStack)) {
                        cycleDetected = true;
                        break;
                    }
                }
            }
        } else {
            const visited = {};
            for (const node in graph) {
                if (!visited[node]) {
                    if (dfsUndirected(graph,
parseInt(node), visited, -1)) {
                        cycleDetected = true;
                        break;
                    } } } }
        resultDiv.innerHTML =
cycleDetected ? "Cycle Detected!" : "No
Cycle Detected!";
    }
  </script>
</body></html>
```

output:


Cycle Detection in Graphs (DFS)
Graph Input
0->1
1->2
2->0
3->4
Directed Graph
Undirected Graph
Detect Cycle
Result
Cycle Detected!


Cycle Detection in Graphs (DFS)
Graph Input
0->1
1->2
2->0
3->4
Directed Graph
Undirected Graph
Detect Cycle
Result
Cycle Detected!

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

Fan Search

Found "Fan" at position 6

Product Catalog

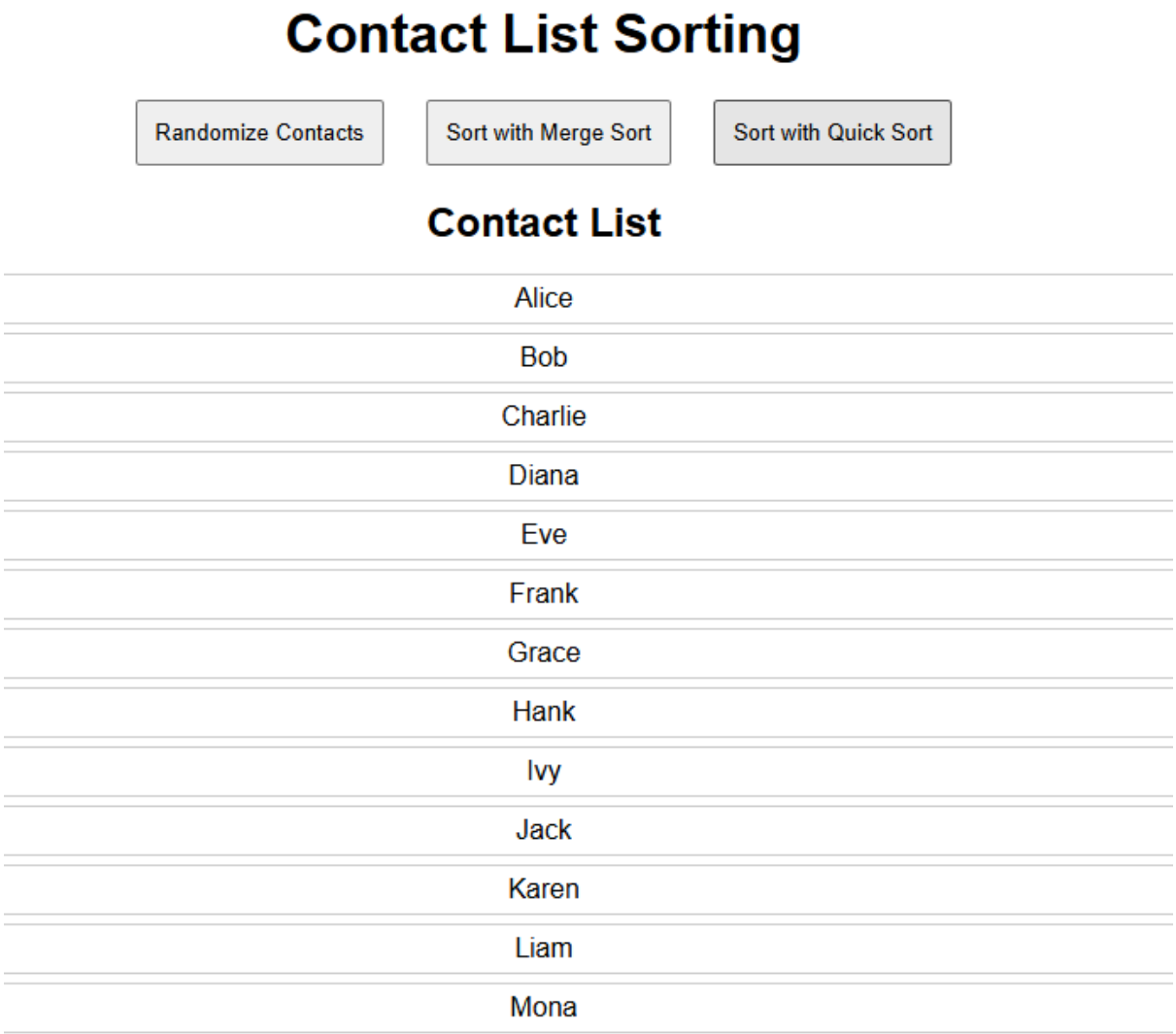
#	Product
1	Air Conditioner
2	Blender
3	Camera
4	Desk
5	Earphones
6	Fan
7	Guitar
8	Headphones
9	Iron

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;
text-align: center; padding: 20px; }
        button { padding: 10px; margin: 10px;
}
        ul { list-style-type: none; padding: 0; }
        li { padding: 5px; border: 1px solid
#ccc; margin: 5px; }
        h1, h2 { margin: 10px 0; }
    </style>
</head>
<body>
    <h1>Contact List Sorting</h1>
    <button
onclick="shuffleContacts()">Randomize
Contacts</button>
    <button
onclick="sortContacts('merge')">Sort with
Merge Sort</button>
    <button
onclick="sortContacts('quick')">Sort with
Quick Sort</button>
    <h2>Contact List</h2>
    <ul id="contactList"></ul>

    <script>
        const contacts = [
            "Alice", "Bob", "Charlie", "Diana",
"Eve", "Frank",
            "Grace", "Hank", "Ivy", "Jack",
"Karen", "Liam",
            "Mona", "Nancy", "Oscar", "Paul",
"Quincy", "Rose",
            "Steve", "Tina", "Uma", "Victor",
"Wendy", "Xander",
            "Yara", "Zane"
        ];

        const contactList =
document.getElementById("contactList");

        function displayContacts(list) {
            contactList.innerHTML =
list.map(contact =>
`<li>${contact}</li>`).join('');
        }
        displayContacts(contacts);

        function shuffleContacts() {
            for (let i = contacts.length - 1; i > 0;
i--) {
                const j =
Math.floor(Math.random() * (i + 1));
                [contacts[i], contacts[j]] =
[contacts[j], contacts[i]];
            }
            displayContacts(contacts);
        }

        function mergeSort(arr) {
            if (arr.length <= 1) return arr;
            const mid = Math.floor(arr.length /
2);
            const left = mergeSort(arr.slice(0,
mid));
            const right =
mergeSort(arr.slice(mid));
            return merge(left, right);
        }

        function merge(left, right) {
            const sorted = [];
            while (left.length && right.length) {
                sorted.push(left[0] < right[0] ?
left.shift() : right.shift());
            }
            return [...sorted, ...left, ...right];
        }

        function quickSort(arr) {
            if (arr.length <= 1) return arr;
```

```
            const pivot = arr[arr.length - 1];
            const left = arr.filter((x, i) => x <=
pivot && i !== arr.length - 1);
            const right = arr.filter(x => x >
pivot);
            return [...quickSort(left), pivot,
...quickSort(right)];
        }

        function sortContacts(method) {
            const sorted = method === 'merge'
? mergeSort(contacts) :
quickSort(contacts);
            displayContacts(sorted);
        }
    </script>
</body>
</html>
```

Output:


Contact List Sorting
Randomize Contacts
Sort with Merge Sort
Sort with Quick Sort
Contact List
Alice
Bob
Charlie
Diana
Eve
Frank
Grace
Hank
Ivy
Jack
Karen
Liam
Mona

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</title>
    <style>
        body { font-family: Arial, sans-serif; text-align: center; padding: 20px; }
        button { padding: 10px; margin: 10px; }
        ul { list-style-type: none; padding: 0; }
        li { padding: 5px; border: 1px solid #ccc; margin: 5px; }
        h1, h2 { margin: 10px 0; }
    </style>
</head>
<body>
    <h1>Event Ranking System (Heap Sort)</h1>
    <button onclick="sortParticipants()">Rank Participants</button>
    <h2>Participants</h2>
    <ul id="participantList"></ul>
    <h2>Participant Rankings</h2>
    <ul id="rankingList"></ul>

    <script>
        // Sample participant data with scores
        const participants = [
            { name: "Alice", score: 90 },
            { name: "Bob", score: 85 },
            { name: "Charlie", score: 92 },
            { name: "Diana", score: 88 },
            { name: "Eve", score: 79 },
            { name: "Frank", score: 95 },
            { name: "Grace", score: 87 },
            { name: "Hank", score: 91 }
        ];

        const participantList = document.getElementById("participantList");
        const rankingList = document.getElementById("rankingList");

        function displayParticipants(list, element) {
            element.innerHTML = list.map(participant =>
                `<li>${participant.name} - Score: ${participant.score}</li>`
            ).join('');
        }

        function displayRankings(list) {
            rankingList.innerHTML = list.map((participant, index) =>
                `<li>Rank ${index + 1}: ${participant.name} - Score: ${participant.score}</li>`
            ).join('');
        }

        // Heap Sort function
        function heapSort(arr) {
            const n = arr.length;

            // Build a max heap
            for (let i = Math.floor(n / 2) - 1; i >= 0; i--) {
                heapify(arr, n, i);
            }

            // One by one extract elements from heap
            for (let i = n - 1; i > 0; i--) {
                // Swap root (max element) with last element
                [arr[0], arr[i]] = [arr[i], arr[0]];

                // Call heapify on the reduced heap
                heapify(arr, i, 0);
            }
```

```
        // Reverse the array to get it in
descending order
        arr.reverse();
    }

    // Heapify a subtree rooted with node
i
    function heapify(arr, n, i) {
        let largest = i;
        const left = 2 * i + 1;
        const right = 2 * i + 2;

        // If left child is larger than root
        if (left < n && arr[left].score >
arr[largest].score) {
            largest = left;
        }

        // If right child is larger than largest
so far
        if (right < n && arr[right].score >
arr[largest].score) {
            largest = right;
        }

        // If largest is not root
        if (largest !== i) {
            [arr[i], arr[largest]] = [arr[largest],
arr[i]]; // Swap

            // Recursively heapify the
affected sub-tree
            heapify(arr, n, largest);
        }
    }

    // Function to trigger the ranking
process
    function sortParticipants() {
        const participantsCopy =
[...participants]; // Create a copy to avoid
modifying original data
        heapSort(participantsCopy); // Sort
participants based on score

displayRankings(participantsCopy); //
Display ranked participants
    }

    // Initial display
    displayParticipants(participants,
participantList);
  </script>
</body>
</html>
```

Output:

Alice - Score: 90

Bob - Score: 85

Charlie - Score: 92

Diana - Score: 88

Eve - Score: 79

Frank - Score: 95

Grace - Score: 87

Hank - Score: 91

Participant Rankings

Rank 1: Frank - Score: 95

Rank 2: Charlie - Score: 92

Rank 3: Hank - Score: 91

Rank 4: Alice - Score: 90

Rank 5: Diana - Score: 88

Rank 6: Grace - Score: 87

Rank 7: Bob - Score: 85

Rank 8: Eve - Score: 79

Event Ranking System (Heap Sort)

Rank Participants

Participants

Alice - Score: 90

Bob - Score: 85

Charlie - Score: 92

Diana - Score: 88

Eve - Score: 79

Frank - Score: 95

Grace - Score: 87

Hank - Score: 91

Participant Rankings

Rank 1: Frank - Score: 95

Rank 2: Charlie - Score: 92

Rank 3: Hank - Score: 91

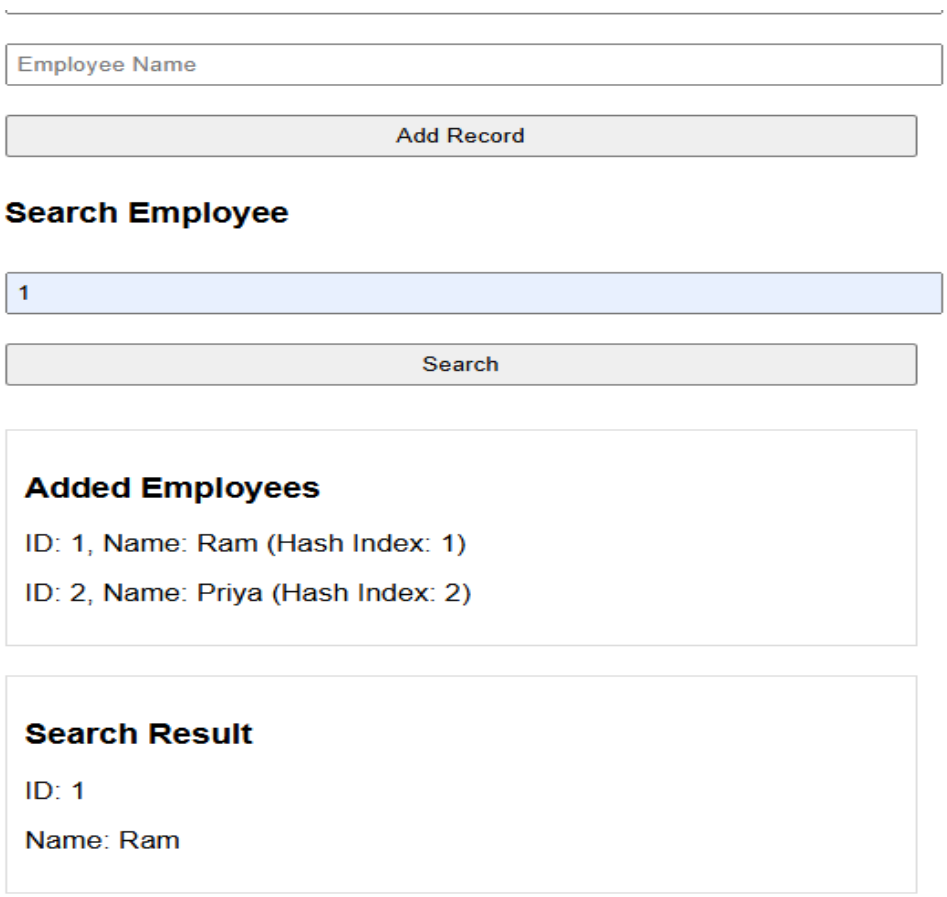
Rank 4: Alice - Score: 90

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>
    <title>Employee Record System</title>
    <style>
        body {
            font-family: Arial;
            max-width: 500px;
            margin: 20px auto;
            padding: 20px;
        }
        input, button {
            width: 100%;
            margin: 10px 0;
            padding: 5px;
        }
        #recordList, #searchResult {
            margin-top: 20px;
            border: 1px solid #ddd;
            padding: 10px;
        }
    </style>
</head>
<body>
    <h2>Employee Record System</h2>

    <h3>Add Employee</h3>
    <input type="text" id="empId" placeholder="Employee ID">
    <input type="text" id="empName" placeholder="Employee Name">
    <button onclick="addRecord()">Add Record</button>

    <h3>Search Employee</h3>
    <input type="text" id="searchId" placeholder="Search by Employee ID">
    <button onclick="searchRecord()">Search</button>

    <div id="recordList">
        <h3>Added Employees</h3>
    </div>

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

    <script>
        class HashTable {
            constructor(size = 10) {
                this.size = size;
                this.table = new Array(size).fill(null).map(() => []); // Initialize each index as an empty array
            }

            // Simple hash function
            hash(key) {
                return parseInt(key) % this.size;
            }

            // Insert record using chaining
            insert(record) {
                const index = this.hash(record.id);
                const bucket = this.table[index];

                // Check for duplicate ID in the bucket
                for (let item of bucket) {
                    if (item.id === record.id) {
                        alert('Error: Duplicate Employee ID');
                        return null;
                    }
                }

                // Add record to the bucket
                bucket.push(record);
                return index;
            }

            // Search record using chaining
            search(id) {
                const index = this.hash(id);
                const bucket = this.table[index];
```

```
            // Search through the bucket for the record
            for (let item of bucket) {
                if (item.id === id) {
                    return item;
                }
            }

            return null; // Record not found
        }
    }

    // Create hash table instance
    const employeeHashTable = new HashTable();

    // Function to add record
    function addRecord() {
        const id = document.getElementById('empId').value;
        const name = document.getElementById('empName').value;
        const recordList = document.getElementById('recordList');

        if (!id || !name) {
            alert('Please enter both ID and Name');
            return;
        }

        // Create record
        const record = { id, name };

        // Insert into hash table
        const index = employeeHashTable.insert(record);

        if (index === null) return; // Skip if duplicate ID

        // Create new record element
        const recordElement = document.createElement('div');
        recordElement.innerHTML = `
            <p>ID: ${id}, Name: ${name} (Hash Index: ${index})</p>
        `;

        // Append to record list
        recordList.appendChild(recordElement);

        // Clear input fields
        document.getElementById('empId').value = '';
        document.getElementById('empName').value = '';
    }

    // Function to search record
    function searchRecord() {
        const searchId = document.getElementById('searchId').value;
        const searchResultDiv = document.getElementById('searchResult');

        if (!searchId) {
            alert('Please enter an Employee ID to search');
            return;
        }

        // Search in hash table
        const record = employeeHashTable.search(searchId);

        // Display search result
        if (record) {
            searchResultDiv.innerHTML = `
                <h3>Search Result</h3>
                <p>ID: ${record.id}</p>
                <p>Name: ${record.name}</p>
            `;
        } else {
            searchResultDiv.innerHTML = '<p>No employee found with this ID</p>';
        }
    }
</script></body></html>
```

Output:


Employee Name
Add Record
Search Employee
1
Search
Added Employees
ID: 1, Name: Ram (Hash Index: 1)
ID: 2, Name: Priya (Hash Index: 2)
Search Result
ID: 1
Name: Ram

Employee Record System

Add Employee

Employee ID

Employee Name

Add Record

Search Employee

1

Search

Added Employees

ID: 1, Name: Ram (Hash Index: 1)

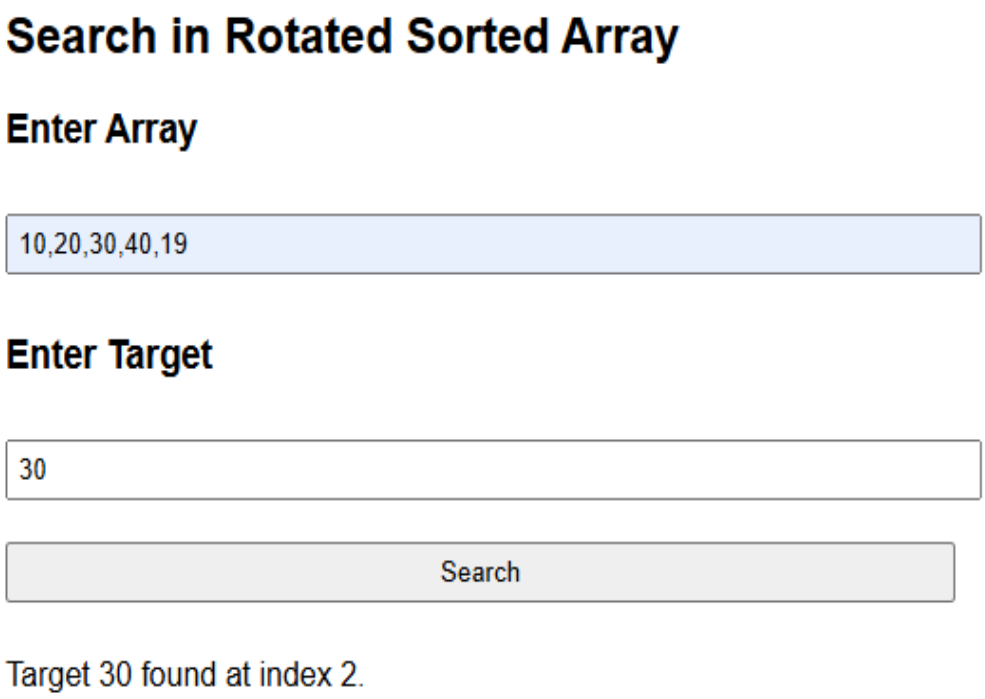
ID: 2, Name: Priya (Hash Index: 2)

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);

                // Check if the target is at mid
                if (arr[mid] === target) {
                    return mid;
                }

                // Determine if the left half is sorted
                if (arr[left] <= arr[mid]) {
                    // Check if the target lies in the sorted left half
```

```
                if (target >= arr[left] && target
< arr[mid]) {
                    right = mid - 1;
                } else {
                    left = mid + 1;
                }
            }
            // Otherwise, the right half is
sorted
            else {
                // Check if the target lies in
the sorted right half
                if (target > arr[mid] && target
<= arr[right]) {
                    left = mid + 1;
                } else {
                    right = mid - 1;
                }
            }
        }

        // Target not found
        return -1;
    }
  </script>
</body>
</html>
```

output:

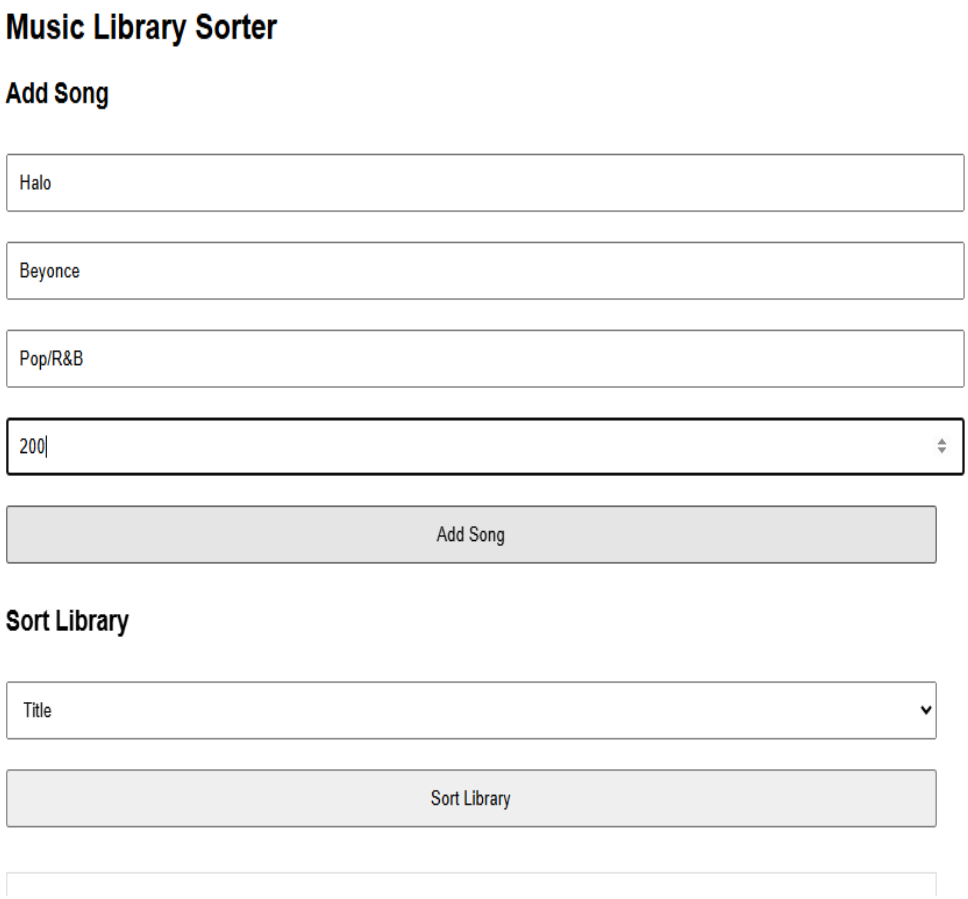

Search in Rotated Sorted Array
Enter Array
10,20,30,40,19
Enter Target
30
Search
Target 30 found at index 2.

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>
    <title>Music Library Sorter</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            max-width: 800px;
            margin: 20px auto;
            padding: 20px;
        }
        input, select, button {
            width: 100%;
            margin: 10px 0;
            padding: 10px;
        }
        #sortedLibrary {
            margin-top: 20px;
            border: 1px solid #ddd;
            padding: 10px;
        }
    </style>
</head>
<body>
    <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>

    <script>
        // Music library array
        const musicLibrary = [];

        // Add a new song to the library
        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();

            // Clear input fields
```

```
document.getElementById('songTitle').value = '';

document.getElementById('songArtist').value = '';

document.getElementById('songGenre').value = '';

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

        // Display the library
        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);
            });
        }

        // Quick Sort implementation
        function quickSort(array, key) {
            if (array.length <= 1) {
                return array;
            }

            // Randomized pivot selection
            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)];
        }

        // Sort the library and display it
        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
Halo
Beyonce
Pop/R&B
200
Add Song
Sort Library
Title
Sort Library

Artist

Genre

Duration (in seconds)

Add Song

Sort Library

Genre

Sort Library

Music Library

- Title: Shape Of You, Artist: Ed Sheeran, Genre: Pop, Duration: 190s
- Title: Believer, Artist: Imagine Dragons, Genre: Pop-Rock, Duration: 180s
- Title: Halo, Artist: Beyonce, Genre: Pop/R&B, Duration: 200s
- Title: Rolling in the Deep, Artist: Adele, Genre: Soul/Pop, Duration: 170s
- Title: Blinding Lights, Artist: The Weeknd, Genre: Synth-pop, Duration: 170s

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>
    <title>LRU Cache</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            max-width: 600px;
            margin: 20px auto;
            padding: 20px;
        }
        input, button {
            margin: 10px 0;
            padding: 10px;
            width: 100%;
        }
        #cacheOutput {
            margin-top: 20px;
            border: 1px solid #ddd;
            padding: 10px;
        }
    </style>
</head>
<body>
    <h2>LRU Cache Simulator</h2>
    <input type="number"
id="cacheCapacity" placeholder="Enter
cache capacity">
    <button
onclick="initializeCache()">Initialize
Cache</button>
    <input type="text" id="cacheKey"
placeholder="Key">
    <input type="text" id="cacheValue"
placeholder="Value">
    <button onclick="putToCache()">Add to
Cache</button>
    <input type="text" id="retrieveKey"
placeholder="Key">
    <button
onclick="getFromCache()">Retrieve</butt
on>
    <div id="cacheOutput">
        <h3>Cache State</h3>
        <ul id="cacheState"></ul>
    </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;
            }

            moveToHead(node) {
                this.removeNode(node);
                this.addNode(node);
            }

            addNode(node) {
                node.next = this.head.next;
                node.prev = this.head;
                this.head.next.prev = node;
                this.head.next = node;
            }

            removeNode(node) {
                node.prev.next = node.next;
                node.next.prev = node.prev;
            }

            removeTail() {
                const tail = this.tail.prev;
                this.removeNode(tail);
```

```
            return tail;
        }

        get(key) {
            if (!this.cache.has(key)) return -
1;
            const node =
this.cache.get(key);
            this.moveToHead(node);
            return node.value;
        }

        put(key, value) {
            if (this.cache.has(key)) {
                const node =
this.cache.get(key);
                node.value = value;
                this.moveToHead(node);
            } else {
                const newNode = new
Node(key, value);
                this.cache.set(key,
newNode);
                this.addNode(newNode);
                if (this.cache.size >
this.capacity) {
                    const tail =
this.removeTail();
                    this.cache.delete(tail.key);
                }
            }
        }
    }

    let lruCache;

    function initializeCache() {
        const capacity =
parseInt(document.getElementById('cache
Capacity').value);
        if (!capacity || capacity <= 0) {
            alert('Please enter a valid
capacity.');
            return;
        }
        lruCache = new
LRUCache(capacity);
document.getElementById('cacheState').in
nerHTML = '<li>Cache initialized.</li>';
    }

    function putToCache() {
        if (!lruCache) {
            alert('Initialize the cache first.');
            return;
        }
        const key =
document.getElementById('cacheKey').val
ue;
        const value =
document.getElementById('cacheValue').v
alue;
        if (!key || !value) {
            alert('Please enter both key and
value.');
            return;
        }
        lruCache.put(key, value);
        displayCacheState();

document.getElementById('cacheKey').val
ue = '';

document.getElementById('cacheValue').v
alue = '';
    }

    function getFromCache() {
        if (!lruCache) {
            alert('Initialize the cache first.');
            return;
        }
        const key =
document.getElementById('retrieveKey').v
alue;
        const value = lruCache.get(key);
        const output =
document.getElementById('cacheOutput');
        output.innerHTML = `<h3>Cache
State</h3><p>${value === -1 ? `No value
found for key "${key}".` : `Value for key
"${key}": ${value}`}</p>`;
        displayCacheState();
    }
```

```
        function displayCacheState() {
            const cacheState =
document.getElementById('cacheState');
            cacheState.innerHTML = '';
            let current = lruCache.head.next;
            while (current !== lruCache.tail) {
                const listItem =
document.createElement('li');
                listItem.textContent = `Key:
${current.key}, Value: ${current.value}`;

cacheState.appendChild(listItem);
                current = current.next;
            }
        }
    </script>
</body>
</html>
```

LRU Cache Simulator

5

Initialize Cache

4

Division

Add to Cache

Key

Retrieve

Cache State

- Key: 3, Value: Multiplication
- Key: 2, Value: Subtraction
- Key: 1, Value: Addition

LRU Cache Simulator

5

Initialize Cache

4

Division

Add to Cache

1

Retrieve

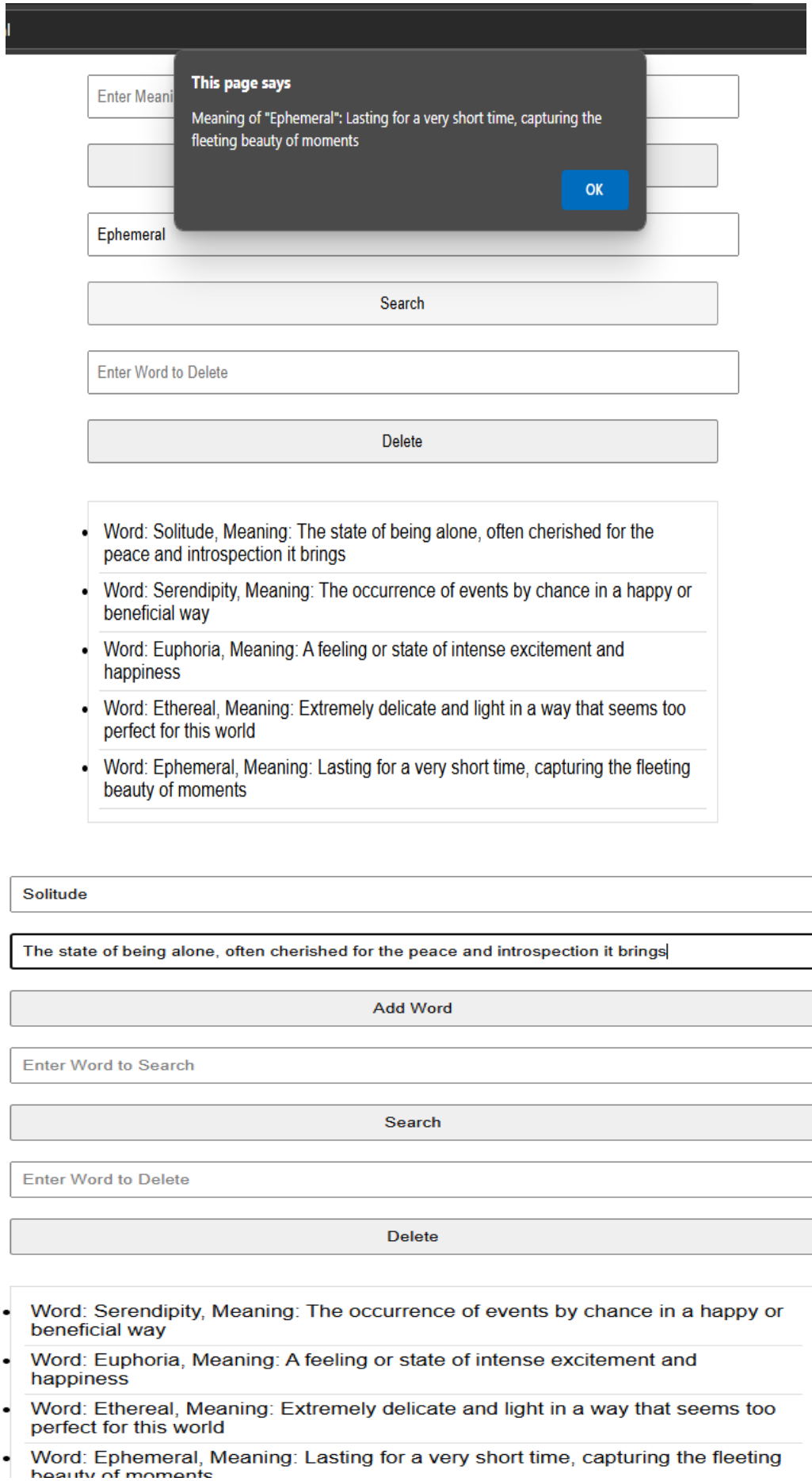
Cache State

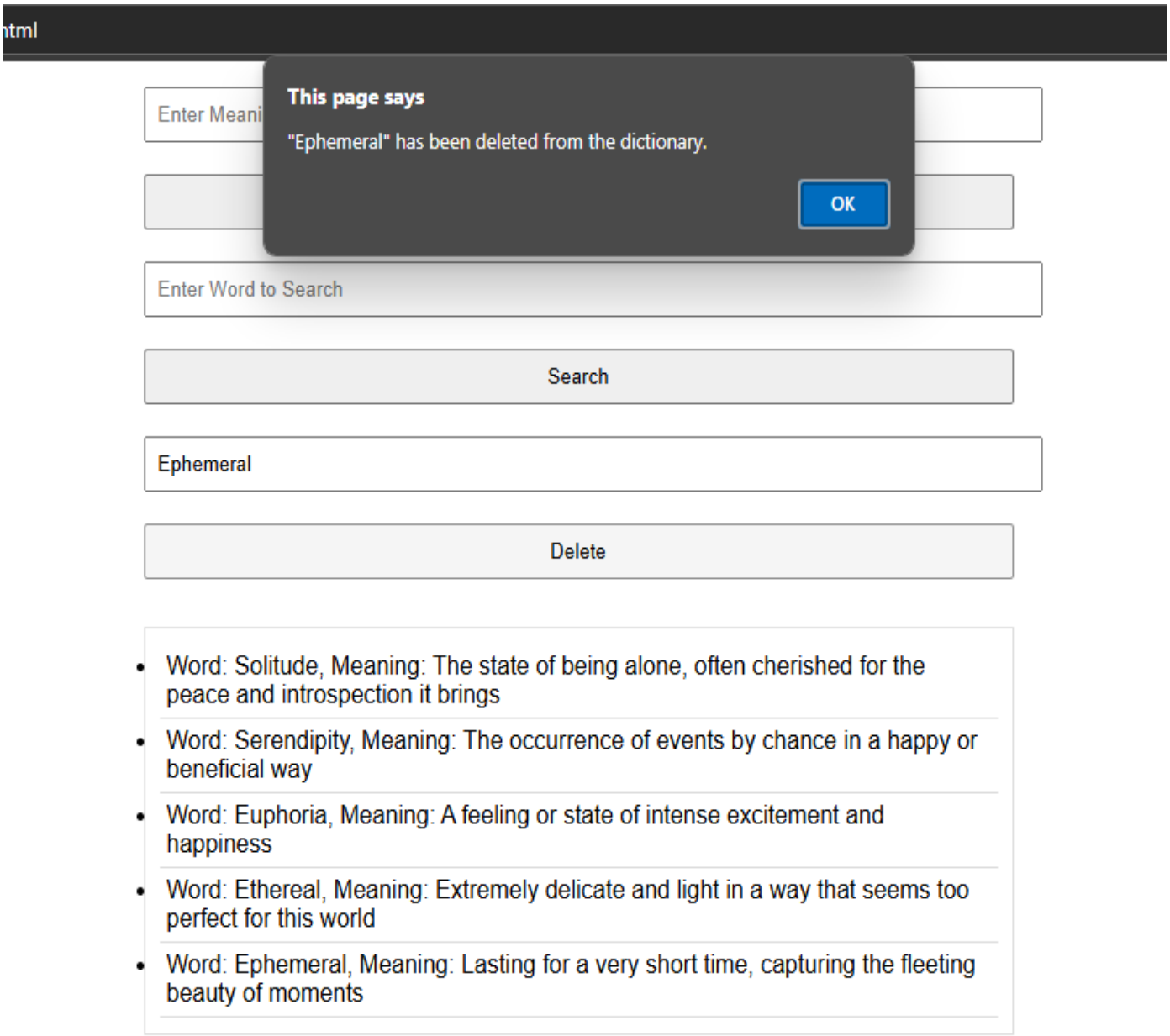
Value for key "1": Addition

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;
max-width: 600px; margin: 20px auto;
padding: 20px; }
        input, button { margin: 10px 0;
padding: 8px; width: 100%; }
        #dictionaryList { margin-top: 20px;
border: 1px solid #ddd; padding: 10px; }
        li { padding: 5px; border-bottom: 1px
solid #ddd; }
    </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').val
ue.trim();
        const meaning =
document.getElementById('meaningInput')
.value.trim();
        if (word && meaning) {
            dictionary.insert(word,
meaning);

document.getElementById('wordInput').val
ue = '';

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').v
alue.trim();
        if (word) {
            dictionary.delete(word);
            alert(`"${word}" has been
deleted from the dictionary.`);

document.getElementById('deleteWord').v
alue = '';
        } else {
            alert('Please enter a word to
delete!');
        }
    }
  </script>
</body>
</html>
```


This page says
Meaning of "Ephemeral": Lasting for a very short time, capturing the fleeting beauty of moments
OK
Enter Meani
Ephemeral
Search
Enter Word to Delete
Delete
Word: Solitude, Meaning: The state of being alone, often cherished for the peace and introspection it brings
Word: Serendipity, Meaning: The occurrence of events by chance in a happy or beneficial way
Word: Euphoria, Meaning: A feeling or state of intense excitement and happiness
Word: Ethereal, Meaning: Extremely delicate and light in a way that seems too perfect for this world
Word: Ephemeral, Meaning: Lasting for a very short time, capturing the fleeting beauty of moments
Solitude
The state of being alone, often cherished for the peace and introspection it brings
Add Word
Enter Word to Search
Search
Enter Word to Delete
Delete
Word: Serendipity, Meaning: The occurrence of events by chance in a happy or beneficial way
Word: Euphoria, Meaning: A feeling or state of intense excitement and happiness
Word: Ethereal, Meaning: Extremely delicate and light in a way that seems too perfect for this world
Word: Ephemeral, Meaning: Lasting for a very short time, capturing the fleeting beauty of moments


tml
This page says
"Ephemeral" has been deleted from the dictionary.
OK
Enter Meani
Enter Word to Search
Search
Ephemeral
Delete
Word: Solitude, Meaning: The state of being alone, often cherished for the peace and introspection it brings
Word: Serendipity, Meaning: The occurrence of events by chance in a happy or beneficial way
Word: Euphoria, Meaning: A feeling or state of intense excitement and happiness
Word: Ethereal, Meaning: Extremely delicate and light in a way that seems too perfect for this world
Word: Ephemeral, Meaning: Lasting for a very short time, capturing the fleeting beauty of moments

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</butt
on>
    <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').va
lue.trim();
            const itemPrice =
parseFloat(document.getElementById('ite
mPrice').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').va
lue = '';

document.getElementById('itemPrice').val
ue = '';
        }

        function displayInventory() {

document.querySelector('#inventoryTable
tbody').innerHTML =
                inventory.map(item =>
`<tr><td>${item.name}</td><td>${item.pri
ce}</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 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('emer
gencyLevel').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('emergencyLev
el').value = '';
        }

        function displayPatients() {

document.querySelector('#patientTable
tbody').innerHTML =
                patients.map(patient =>
`<tr><td>${patient.name}</td><td>${patie
nt.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

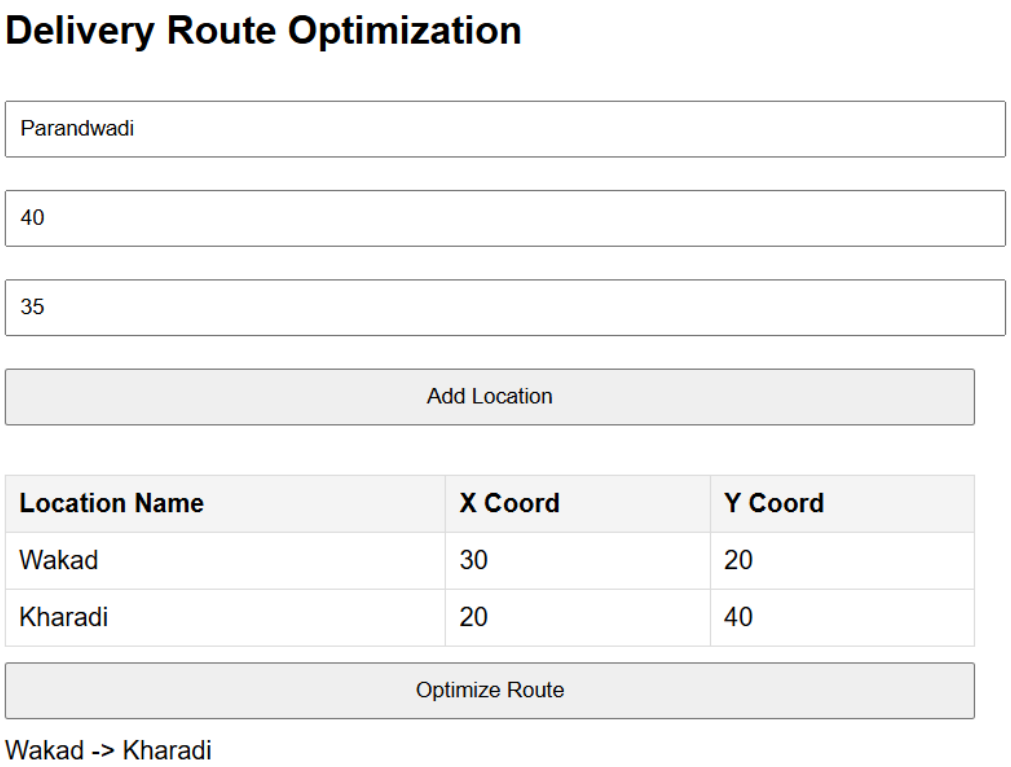
MODULE 6

1. Delivery Route Optimization (Greedy Algorithm)

```
<!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</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>Delivery Route Optimization</h2>
  <input type="text" id="locationName"
placeholder="Location Name">
  <input type="number" id="xCoord"
placeholder="X Coordinate">
  <input type="number" id="yCoord"
placeholder="Y Coordinate">
  <button onclick="addLocation()">Add
Location</button>
  <table id="locationsTable">
    <thead>
      <tr><th>Location Name</th><th>X
Coord</th><th>Y Coord</th></tr>
    </thead>
    <tbody></tbody>
  </table>
  <button
onclick="optimizeRoute()">Optimize
Route</button>
  <div id="optimizedRoute"></div>

  <script>
    let locations = [];

    function addLocation() {
      const name =
document.getElementById('locationName').
value.trim();
      const x =
parseFloat(document.getElementById('xCo
ord').value.trim());
      const y =
parseFloat(document.getElementById('yCo
ord').value.trim());
      if (!name || isNaN(x) || isNaN(y))
return alert('Enter valid details.');

      locations.push({ name, x, y });
      displayLocations();

document.getElementById('locationName').
value = '';

document.getElementById('xCoord').value
= '';

document.getElementById('yCoord').value
= '';
    }

    function displayLocations() {

document.querySelector('#locationsTable
tbody').innerHTML =
        locations.map(loc =>
`<tr><td>${loc.name}</td><td>${loc.x}</t
d><td>${loc.y}</td></tr>`).join('');
    }

    function calculateDistance(loc1, loc2)
{
      return Math.sqrt((loc1.x - loc2.x) **
2 + (loc1.y - loc2.y) ** 2);
    }

    function optimizeRoute() {
      if (locations.length < 2) return
alert('Add at least two locations.');

      let visited = new Set();
      let current = locations[0];
      let route = [current];
```

```
        visited.add(current);

        while (visited.size < locations.length) {
            let nearest = null;
            let minDistance = Infinity;
            for (let loc of locations) {
                if (!visited.has(loc)) {
                    let dist = calculateDistance(current, loc);
                    if (dist < minDistance) {
                        minDistance = dist;
                        nearest = loc;
                    }
                }
            }
            route.push(nearest);
            visited.add(nearest);
            current = nearest;
        }


document.getElementById('optimizedRoute').innerHTML = route.map(loc => loc.name).join(' -> ');
    }
  </script>
</body>
</html>
```

OUTPUT:


Delivery Route Optimization
Parandwadi
40
35
Add Location
Location Name
X Coord
Y Coord
Wakad
30
20
Kharadi
20
40
Optimize Route
Wakad -> Kharadi

2. Knapsack Problem (Dynamic Programming)

```
<!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;
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>0/1 Knapsack Problem</h2>
  <input type="number" id="weightLimit"
placeholder="Weight Limit">
  <input type="text" id="itemName"
placeholder="Item Name">
  <input type="number" id="itemWeight"
placeholder="Item Weight">
  <input type="number" id="itemValue"
placeholder="Item Value">
  <button onclick="addItem()">Add
Item</button>
  <table id="itemsTable">
    <thead>
      <tr><th>Item
Name</th><th>Weight</th><th>Value</th
></tr>
    </thead>
    <tbody></tbody>
  </table>
  <button
onclick="solveKnapsack()">Solve
Knapsack</button>
  <h3>Optimal Items</h3>
  <div id="knapsackResult"></div>

  <script>
    const items = [];

    function addItem() {
      const name =
document.getElementById('itemName').val
ue.trim();
      const weight =
parseFloat(document.getElementById('item
Weight').value.trim());
      const value =
parseFloat(document.getElementById('item
Value').value.trim());
      if (!name || isNaN(weight) ||
isNaN(value)) return alert('Enter valid item
details.');

      items.push({ name, weight, value });
      displayItems();

document.getElementById('itemName').val
ue = '';

document.getElementById('itemWeight').va
lue = '';

document.getElementById('itemValue').val
ue = '';
    }

    function displayItems() {

document.querySelector('#itemsTable
tbody').innerHTML =
        items.map(item =>
`<tr><td>${item.name}</td><td>${item.w
eight}</td><td>${item.value}</td></tr>`).j
oin('');
    }

    function solveKnapsack() {
      const weightLimit =
parseFloat(document.getElementById('weig
htLimit').value.trim());
      if (isNaN(weightLimit)) return
alert('Enter a valid weight limit.');
```

```
        const n = items.length;
        const dp = Array.from({ length: n + 1 }, () => Array(weightLimit + 1).fill(0));

        for (let i = 1; i <= n; i++) {
            for (let w = 0; w <= weightLimit; w++) {
                if (items[i - 1].weight <= w) {
                    dp[i][w] = Math.max(dp[i - 1][w], dp[i - 1][w - items[i - 1].weight] + items[i - 1].value);
                } else {
                    dp[i][w] = dp[i - 1][w];
                }
            }
        }

        let w = weightLimit;
        const result = [];
        for (let i = n; i > 0 && w > 0; i--) {
            if (dp[i][w] !== dp[i - 1][w]) {
                result.push(items[i - 1]);
                w -= items[i - 1].weight;
            }
        }

        document.getElementById('knapsackResult').innerHTML =
            result.length ? result.map(item => `${item.name} (Weight: ${item.weight}, Value: ${item.value})`).join('<br>') : 'No items selected.';
    }
  </script>
</body>
</html>
```

OUTPUT:

0/1 Knapsack Problem

15

Cotton

10

6

Add Item

Item Name	Weight	Value
Potato	8	50
Clothes	5	100
Iron	20	5

Solve Knapsack

Optimal Items

Clothes (Weight: 5, Value: 100)
Potato (Weight: 8, Value: 50)

3. Divide and Conquer Approach for Matrix Multiplication

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Matrix Multiplication</title>
  <style>
    body { font-family: Arial, sans-serif; margin: 20px; }
    button { margin: 20px 0; }
    table { margin: auto; border-collapse: collapse; }
    th, td { border: 1px solid #ddd; padding: 8px; }
    th { background-color: #f2f2f2; }
  </style>
</head>
<body>
  <h1>Matrix Multiplication</h1>
  <button onclick="comparePerformance()">Compare Performance</button>
  <table id="results">
    <tr><th>Algorithm</th><th>Time (seconds)</th></tr>
  </table>
  <script>
    function multiplyMatrices(A, B) {
      const n = A.length, C = Array(n).fill(0).map(() => 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++)
            C[i][j] += A[i][k] * B[k][j];
      return C;
    }

    function addMatrices(A, B) {
      return A.map((row, i) => row.map((val, j) => val + B[i][j]));
    }

    function subtractMatrices(A, B) {
      return A.map((row, i) => row.map((val, j) => val - B[i][j]));
    }

    function strassenMultiply(A, B) {
      const n = A.length;
      if (n === 1) return [[A[0][0] * B[0][0]]];
      const mid = n / 2,
        A11 = A.slice(0, mid).map(r => r.slice(0, mid)),
        A12 = A.slice(0, mid).map(r => r.slice(mid)),
        A21 = A.slice(mid).map(r => r.slice(0, mid)),
        A22 = A.slice(mid).map(r => r.slice(mid)),
        B11 = B.slice(0, mid).map(r => r.slice(0, mid)),
        B12 = B.slice(0, mid).map(r => r.slice(mid)),
        B21 = B.slice(mid).map(r => r.slice(0, mid)),
        B22 = B.slice(mid).map(r => r.slice(mid)),
        M1 = strassenMultiply(addMatrices(A11, A22), addMatrices(B11, B22)),
        M2 = strassenMultiply(addMatrices(A21, A22), B11),
        M3 = strassenMultiply(A11, subtractMatrices(B12, B22)),
        M4 = strassenMultiply(A22, subtractMatrices(B21, B11)),
        M5 = strassenMultiply(addMatrices(A11, A12), B22),
        M6 = strassenMultiply(subtractMatrices(A21, A11), addMatrices(B11, B12)),
        M7 = strassenMultiply(subtractMatrices(A12, A22), addMatrices(B21, B22)),
        C11 = addMatrices(subtractMatrices(addMatrices(M1, M4), M5), M7),
        C12 = addMatrices(M3, M5),
        C21 = addMatrices(M2, M4),
```

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

        return Array(mid * 2).fill(0).map((_,
i) =>
            (i < mid ? C11[i] : C21[i -
mid]).concat(i < mid ? C12[i] : C22[i -
mid])
        );
    }

    function generateMatrix(n) {
        return Array(n).fill(0).map(() =>
Array(n).fill(0).map(() =>
Math.floor(Math.random() * 10)));
    }

    function comparePerformance() {
        const n = 128, A =
generateMatrix(n), B = generateMatrix(n);

        const time = (fn) => {
            const start = performance.now();
            fn(A, B);
            return (performance.now() -
start).toFixed(4);
        };

        const standardTime =
time(multiplyMatrices),
              strassenTime =
time(strassenMultiply);


document.getElementById('results').innerH
TML += `

<tr><td>Standard</td><td>${standardTime
}</td></tr>

<tr><td>Strassen</td><td>${strassenTime
}</td></tr>
        `;
    }
  </script>
</body>
</html>
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport"
content="width=device-width, initial-
scale=1.0">
  <title>Traveling Salesman
Approximation</title>
  <style>
    body {
      display: flex;
      flex-direction: column;
      align-items: center;
      justify-content: center;
      height: 100vh;
      margin: 0;
    }

    /* Add this to keep elements from
overlapping */
    body > * {
      margin-bottom: 20px;
    }

    /* Add this to center text elements */
    body > * {
      text-align: center;
    }
     button { margin: 20px 0; }
     table { margin: auto; border-collapse:
collapse; align-items: center;}
    th, td { border: 1px solid #ddd;
padding: 8px; }
     th { background-color: #f2f2f2; }
     form { margin-bottom: 20px; }
     input { margin: 5px 0; width: 50px;
text-align: center; }
  </style>
</head>
<body>
  <h1>Traveling Salesman
Approximation</h1>
  <form id="graphForm">
    <label for="size">Matrix Size (n x
n):</label>
```

```
        <input type="number" id="size"
name="size" min="2" value="4">
        <button type="button"
onclick="generateMatrixInputs()">Generat
e Matrix Inputs</button>
        <div id="matrixInputs"></div>
        <button type="submit">Run
Approximation</button>
    </form>
    <table id="results">
        <tr><th>Path</th><th>Length</th></t
r>
    </table>
    <script>
        function generateMatrixInputs() {
            const size =
document.getElementById('size').value;
            const matrixInputs =
document.getElementById('matrixInputs');
            matrixInputs.innerHTML = '';
            for (let i = 0; i < size; i++) {
                for (let j = 0; j < size; j++) {
                    matrixInputs.innerHTML +=
`<input type="number" name="cell-${i}-
${j}" value="0">`;
                }
                matrixInputs.innerHTML +=
'<br>';
            }
        }

        function getMatrixFromInputs(size) {
            const matrix = Array.from({ length:
size }, () => Array(size).fill(0));
            for (let i = 0; i < size; i++) {
                for (let j = 0; j < size; j++) {
                    matrix[i][j] =
parseFloat(document.querySelector(`[name
="cell-${i}-${j}"]`).value);
                }
            }
            return matrix;
        }

        function approximateTSP(graph) {
            const n = graph.length;
            const visited = Array(n).fill(false);
            let path = [];

            function minKey(key, mstSet) {
                let min = Infinity, minIndex;
                for (let v = 0; v < n; v++)
                    if (!mstSet[v] && key[v] <
min)
                        min = key[v], minIndex = v;
                return minIndex;
            }

            function primMST() {
                let parent = Array(n).fill(-1);
                let key = Array(n).fill(Infinity);
                let mstSet = Array(n).fill(false);
                key[0] = 0;

                for (let count = 0; count < n - 1;
count++) {
                    let u = minKey(key, mstSet);
                    mstSet[u] = true;

                    for (let v = 0; v < n; v++)
                        if (graph[u][v] &&
!mstSet[v] && graph[u][v] < key[v])
                            parent[v] = u, key[v] =
graph[u][v];
                }
                return parent;
            }

            function dfs(u, parent) {
                visited[u] = true;
                path.push(u);
                for (let v = 0; v < n; v++)
                    if (parent[v] === u &&
!visited[v])
                        dfs(v, parent);
            }

            const parent = primMST();
            dfs(0, parent);
            path.push(0); // Complete the tour

            let length = 0;
            for (let i = 0; i < path.length - 1;
i++)
                length += graph[path[i]][path[i +
1]];

            document.getElementById('results').
innerHTML = `
```

```
            <tr><td>${path.join(' -> ')}</td><td>${length}</td></tr>
            `;
        }

        document.getElementById('graphForm').addEventListener('submit', function(event) {
            event.preventDefault();
            const size = document.getElementById('size').value;
            const graph = getMatrixFromInputs(size);
            approximateTSP(graph);
        });

        generateMatrixInputs();
    </script>
</body>
</html>
```

OUTPUT:

Matrix Multiplication

Compare Performance

Algorithm	Time (seconds)
Standard	21.5000
Strassen	864.8000
Standard	14.9000
Strassen	872.8000
Standard	15.8000
Strassen	835.0000
Standard	12.8000
Strassen	874.8000

4. Approximation Algorithms for NP-Complete Problems

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport"
content="width=device-width, initial-
scale=1.0">
  <title>Traveling Salesman
Approximation</title>
  <style>
    body {
      display: flex;
      flex-direction: column;
      align-items: center;
      justify-content: center;
      height: 100vh;
      margin: 0;
    }

    body > * {
      margin-bottom: 20px;
    }

    body > * {
      text-align: center;
    }
     button { margin: 20px 0; }
     table { margin: auto; border-collapse:
collapse; align-items: center;}
     th, td { border: 1px solid #ddd;
padding: 8px; }
     th { background-color: #f2f2f2; }
     form { margin-bottom: 20px; }
     input { margin: 5px 0; width: 50px;
text-align: center; }
  </style>
</head>
<body>
  <h1>Traveling Salesman
Approximation</h1>
  <form id="graphForm">
    <label for="size">Matrix Size (n x
n):</label>
    <input type="number" id="size"
name="size" min="2" value="4">
    <button type="button"
onclick="generateMatrixInputs()">Generat
e Matrix Inputs</button>
    <div id="matrixInputs"></div>
    <button type="submit">Run
Approximation</button>
  </form>
  <table id="results">
    <tr><th>Path</th><th>Length</th></t
r>
  </table>
  <script>
    function generateMatrixInputs() {
      const size =
document.getElementById('size').value;
      const matrixInputs =
document.getElementById('matrixInputs');
      matrixInputs.innerHTML = '';
      for (let i = 0; i < size; i++) {
        for (let j = 0; j < size; j++) {
          matrixInputs.innerHTML +=
`<input type="number" name="cell-${i}-
${j}" value="0">`;
        }
        matrixInputs.innerHTML +=
'<br>';
      }
    }

    function getMatrixFromInputs(size) {
      const matrix = Array.from({ length:
size }, () => Array(size).fill(0));
      for (let i = 0; i < size; i++) {
        for (let j = 0; j < size; j++) {
          matrix[i][j] =
parseFloat(document.querySelector(`[name
="cell-${i}-${j}"]`).value);
        }
      }
      return matrix;
    }

    function approximateTSP(graph) {
      const n = graph.length;
      const visited = Array(n).fill(false);
      let path = [];

      function minKey(key, mstSet) {
        let min = Infinity, minIndex;
        for (let v = 0; v < n; v++)
```

```
            if (!mstSet[v] && key[v] <
min)
                min = key[v], minIndex = v;
            return minIndex;
        }

        function primMST() {
            let parent = Array(n).fill(-1);
            let key = Array(n).fill(Infinity);
            let mstSet = Array(n).fill(false);
            key[0] = 0;

            for (let count = 0; count < n - 1;
count++) {
                let u = minKey(key, mstSet);
                mstSet[u] = true;

                for (let v = 0; v < n; v++)
                    if (graph[u][v] &&
!mstSet[v] && graph[u][v] < key[v])
                        parent[v] = u, key[v] =
graph[u][v];
            }
            return parent;
        }

        function dfs(u, parent) {
            visited[u] = true;
            path.push(u);
            for (let v = 0; v < n; v++)
                if (parent[v] === u &&
!visited[v])
                    dfs(v, parent);
        }

        const parent = primMST();
        dfs(0, parent);
        path.push(0); // Complete the tour

        let length = 0;
        for (let i = 0; i < path.length - 1;
i++)
            length += graph[path[i]][path[i +
1]];

        document.getElementById('results').
innerHTML = `
            <tr><td>${path.join(' ->
')}</td><td>${length}</td></tr>
        `;
    }

    document.getElementById('graphForm
').addEventListener('submit',
function(event) {
        event.preventDefault();
        const size =
document.getElementById('size').value;
        const graph =
getMatrixFromInputs(size);
        approximateTSP(graph);
    });

    generateMatrixInputs();
  </script>
</body>
</html>
```

OUTPUT:

Traveling Salesman Approximation

Matrix Size (n x n): 3 Generate Matrix Inputs

1	2	3
5	7	9
4	8	6

Run Approximation

0 -> 1 -> 2 -> 0	15