KONGU ENGINEERING COLLEGE

(Autonomous)

Perundurai, Erode– 638060

DEPARTMENT OF INFORMATION TECHNOLOGY

Computing Largest and Smallest Elements In An Array

A MICRO PROJECT REPORT

FOR

DESIGN AND ANALYSIS OF ALGORITHMS (22ITT31)

SUBMITTED BY

DHIVYASHRI K K (23ITR037)

KONGU ENGINEERING COLLEGE

(Autonomous)

Perundurai, Erode– 638060

DEPARTMENT OF INFORMATION TECHNOLOGY

Computing Largest and Smallest Elements In An Array

A MICRO PROJECT REPORT

FOR

DESIGN AND ANALYSIS OF ALGORITHMS (22ITT31)

SUBMITTED BY

DHIVYASHRI K K (23ITR037)

KONGU ENGINEERING COLLEGE

(Autonomous)

Perundurai, Erode– 638060

DEPARTMENT OF INFORMATION TECHNOLOGY

BONAFIDE CERTIFICATE

Name : DHIVYASHRI K K

Course Code : 22ITT31

Course Name : DESIGN AND ANALYSIS OF ALGORITHMS

Semester : IV

Certified that this is a bonafide record of work for an application project done by the above student for 22ITT31-DESIGN AND ANALYSIS OF ALGORITHMS during the academic year 2024-2025.

Submitted for the Viva Voce Examination held on _____

Faculty Incharge Head of the Department

ABSTRACT

This project focuses on finding the smallest and largest elements in an array using three different algorithmic approaches: the brute-force method, the presorting-based method, and the divide-and-conquer technique. Each approach is implemented using JavaScript and presented through an interactive HTML interface, allowing users to input an array and view results for all three methods simultaneously.

The brute-force method iterates through the array to track the minimum and maximum values directly, yielding a linear time complexity O(n). The presorting-based method first sorts the array and then selects the smallest and largest elements from the sorted list, achieving a time complexity of O(n log n). The divide-and-conquer technique recursively splits the array into smaller parts, computes local minima and maxima, and then combines the results and also achieving a linear time complexity but with fewer operations.

This implementation helps users understand how different algorithms solve the same problem using varied logic. It also highlights the practical aspects of algorithm selection depending on the problem context. The project serves as a learning tool for exploring core algorithmic strategies and improving problem-solving skills through visual, hands-on interaction.

TABLE OF CONTENTS

Chapter 1

INTRODUCTION

Finding the minimum and maximum elements in an array is a foundational problem in computer science, serving as a practical introduction to algorithm design and efficiency analysis. This project presents an interactive Algorithm Visualizer focused on the classic challenge of identifying the smallest and largest values within a given array. Through a user-friendly web interface, learners and practitioners can explore and compare three prominent algorithmic strategies: the straightforward Brute Force approach, a Presorting-based method, and the Divide-and-Conquer technique.

The application is implemented using HTML5, CSS3, and JavaScript, providing an accessible platform for hands-on experimentation. Users can input custom arrays and select the algorithm they wish to observe, with each approach's results and operational characteristics displayed in real time. This interactive format not only highlights the differences in computational complexity—ranging from O(n) for Brute Force and Divide-and-Conquer to O(n log n) for Presorting—but also illustrates the trade-offs in implementation and performance

1.1 PURPOSE

The main purpose of this project is to:

- To provide an interactive platform for understanding and comparing different algorithms for finding the minimum and maximum elements in an array.
- To bridge the gap between theoretical concepts and practical applications of algorithm analysis.
- To facilitate hands-on learning through visualization and experimentation with various algorithmic approaches.
- To help users appreciate the trade-offs between brute-force, presorting-based, and divide-and-conquer strategies.

1.2 OBJECTIVE

The primary objectives of this Algorithm Visualization project are:

Educational Enhancement

- Provide an interactive platform for learning to find minimum and maximum elements in arrays.
- Demonstrate the practical implementation of three classical algorithms: Brute-Force, Presorting-Based, and Divide-and-Conquer.
- Visualize the differences in time complexity and operational steps among these algorithms.

Algorithm Comparison

- Compare the efficiency and performance of the three distinct approaches for identifying minimum and maximum values in an array
- Measure and display real-time performance metrics
- Illustrate space-time trade-offs between different solutions

1.3 METHODOLOGY OVERVIEW

1. USER INPUT

- The user enters an array of numbers through a designated input field or text area.
- The user selects the desired algorithm (Brute-Force, Presorting-Based, or Divide-and-Conquer) from the buttons below.

2. ARRAY PREPARATION

- The application parses and validates the user input to ensure all elements are valid numbers.
- The input array is prepared and displayed for visualization.

3. ALGORITHM EXECUTION

Based on the user's selection, the corresponding algorithm is executed:

- **Brute-Force:** Iterates through the array to find minimum and maximum values using direct comparisons.
- **Presorting-Based:** Sorts the array (e.g., using built-in sort or a sorting algorithm) and retrieves the first and last elements as the minimum and maximum.
- **Divide-and-Conquer:** Recursively splits the array, finds the minimum and maximum in each part, and combines the results.

4. OPERATION VISUALIZATION

Each step of the selected algorithm is visualized to show comparisons, swaps , or recursion as appropriate.

- Key operations (such as comparisons or array splits) are highlighted for clarity.
- Real-time performance metrics (e.g., execution time, number of comparisons) are collected and displayed.

Chapter 2

PROBLEM STATEMENT

The problem of finding the minimum and maximum elements in an array is a fundamental challenge in computer science and serves as an introduction to algorithm design and analysis. The task is to identify the smallest and largest values within a given array of numbers, which is essential for a wide range of applications, from statistical analysis to optimization problems.

While the brute-force approach solves this problem with a straightforward linear scan—resulting in O(n) time complexity—other methods such as the presorting-based approach (O(n log n)) and divide-and-conquer algorithms (O(n) with fewer comparisons) offer alternative strategies with varying efficiency, trade-offs, and implementation complexity.

The primary objective of this project is to visualize and compare these three algorithmic approaches in real-time using an interactive web-based platform. Users can input custom arrays or generate random arrays of varying sizes, select the algorithm to apply, and observe both the result and the operational steps. The visualizer presents execution steps, highlights performance characteristics, and displays comparative metrics for intuitive understanding.

Chapter 3

METHODOLOGY

3.1 INPUT & INITIALIZATION

- Accept an array of integers from the user.
- Allow users to either manually enter custom arrays or generate random arrays of varying sizes (e.g., 10 to 100,000 elements).
- Provide an option for users to select the algorithm they wish to visualize: Brute Force, Presorting-Based, or Divide and Conquer.
- Set up performance timers to track execution time and number of comparisons.
- Prepare the visualization interface using HTML, CSS, and JavaScript.

3.2 BRUTE-FORCE APPROACH

- Iterate through the array using a single linear scan.
- Compare each element to update both the current minimum and maximum values.
- Track the number of comparisons performed.

3.3 PRESORTING-BASED APPROACH

- Sort the array using an efficient sorting algorithm (e.g., quicksort or built-in sort).
- After sorting, select the first element as the minimum and the last element as the maximum.
- Count the number of key comparisons performed during the sorting process.
- Record the total execution time, which includes sorting and selection steps.

3.4 DIVIDE AND CONQUER APPROACH

- Recursively divide the array into two halves until each subarray contains one or two elements.
- Find the minimum and maximum of each subarray.
- Combine results by comparing the minima and maxima of the subarrays.
- Minimize the number of overall comparisons through this recursive process.

3.5 Visualization & Output

- Display a summary of the input array and the algorithm selected.
- Present the identified minimum and maximum values along with the total number of comparisons and execution time.
- Provide step-by-step or summary visualizations of how each algorithm processes the array, highlighting comparisons and updates to min/max values.

Chapter 4

IMPLEMENTATION

4.1 INPUT AND INITIALIZATION

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Find Smallest and Largest in Array</title>
   <link rel="stylesheet" href="style.css">
</head>
<body>
   <div class="container">
      <h2>Find Smallest and Largest Elements in an Array</h2>
      <label for="arrayInput">Enter numbers separated by commas:</label>
      <input type="text" id="arrayInput" size="50" placeholder="Example: 4, 1, 7, 9,
2">
      <div class="button-group">
         <button onclick="displayAlgorithm('brute')">Brute-force</button>
         <button onclick="displayAlgorithm('presort')">Presorting-based</button>
         <button onclick="displayAlgorithm('divide')">Divide-and-Conquer</button>
      </div>
      <h3>Result</h3>
      <div id="results"></div>
   </div>
   <footer>
      <span>Made for Algorithms study</span>
   </footer>
   <script src="main.js"></script>
</body>
</html>
```

4.2 BRUTE-FORCE ALGORITHM

```
function bruteForceMinMax(arr) {
    let min = arr[0], max = arr[0];
    for (let i = 1; i < arr.length; i++) {
        if (arr[i] < min) min = arr[i];
        if (arr[i] > max) max = arr[i];
    }
    return {min, max, comparisons: 2 * (arr.length - 1)};
}
```

4.3PRESORTING ALGORITHM

```
function presortMinMax(arr) {
    let sorted = [...arr].sort((a, b) => a - b);
    return {min: sorted[0], max: sorted[sorted.length - 1], comparisons: "O(n log n)"};
}
```

4.2 DIVIDE AND CONQUER ALGORITHM

```
function divideAndConquerMinMax(arr) {
    let comparisons = 0;
    function helper(low, high) {
        if (low === high) {
            return {min: arr[low], max: arr[high]};
        }
        if (high === low + 1) {
            comparisons++;
            if (arr[low] < arr[high]) return {min: arr[low], max: arr[high]};
            else return {min: arr[high], max: arr[low]};
        }
        let mid = Math.floor((low + high) / 2);
        let left = helper(low, mid);
        let right = helper(mid + 1, high);
```

```
        comparisons += 2;
        return {
            min: left.min < right.min ? left.min : right.min,
            max: left.max > right.max ? left.max : right.max
        };
    }
    let result = helper(0, arr.length - 1);
    result.comparisons = comparisons;
    return result;
}
```

4.4 VISUALIZATION & OUTPUT

```
function displayAlgorithm(type) {
    const input = document.getElementById('arrayInput').value;
    let arr = input.split(',').map(Number).filter(x => !isNaN(x));
    if (arr.length === 0) {
        document.getElementById('results').innerHTML = '<span
style="color:#ea5753;">Please enter a valid array.</span>';
        return;
    }

    let html = `<b>Input Array:</b> [${arr.join(', ')}]<br><br>`;
    if (type === 'brute') {
        const brute = bruteForceMinMax(arr);
        html += `<b>Brute-force Algorithm</b><br>
            Min = <span style="color:#00897b;">${brute.min}</span>,
            Max = <span style="color:#b71c1c;">${brute.max}</span><br>
            Comparisons = ${brute.comparisons}<br>
            <i>Efficiency:</i> <span style="color:#ea5753;">O(n)</span>`;
    } else if (type === 'presort') {
        const sorted = presortMinMax(arr);
        html += `<b>Presorting-based Algorithm</b><br>
```

```
        Min = <span style="color:#00897b;">${sorted.min}</span>,
        Max = <span style="color:#b71c1c;">${sorted.max}</span><br>
        Comparisons = ${sorted.comparisons}<br>
        <i>Efficiency:</i> <span style="color:#ea5753;">O(n log n)</span>`;
} else if (type === 'divide') {
    const dac = divideAndConquerMinMax(arr);
    html += `<b>Divide-and-Conquer Algorithm</b><br>
        Min = <span style="color:#00897b;">${dac.min}</span>,
        Max = <span style="color:#b71c1c;">${dac.max}</span><br>
        Comparisons = ${dac.comparisons}<br>
        <i>Efficiency:</i> <span style="color:#ea5753;">O(n)</span>`;
}
document.getElementById('results').innerHTML = html;
```

DIFFERENCE BETWEEN BRUTEFORCE , PRESORTING AND DIVIDE AND CONQUER

Brute Force

Concept

- Scan each element of the matrix regardless of ordering.
- Does not utilize the sorted property (just direct comparison).

How it works

1.Initialize `min` and `max` as the first element of the array.

2. Traverse the array from the second element onwards.

3. For each element:

    - If smaller than `min`, update `min`.
    - If larger than `max`, update `max`.

4. Continue until every element is visited

Time Complexity

- Worst-case: O(n)
- Every element is checked exactly once(after the first).

Recurrence Relation

- 2(n-1)

Divide and conquer

Concept

- Recursively divide the array into two halves, solve for each half, and combine results.
- Reduces the total number of comparisons.

How it works

- If the subarray has one element, it's both min and max.
- If two elements, compare once and assign min and max.
- Otherwise:
    - Divide the array into two halves.
    - Recursively find min and max for each half.
    - Compare the two minima and the two maxima.

Time Complexity

- Worst-case: O(n)
- For n = 2^k, total comparisons: 1.5n – 2

Recurrence Relation

- C(1) = 0
- C(2) = 1
- C(n) = C(⌊n/2⌋) + C(⌈n/2⌉) + 2

Presorting

Concept

- Sort the array first, then pick the first and last elements as the minimum and maximum.

How it works

1.Sort the array using a comparison-based sorting algorithm (e.g., merge sort, quicksort).

2. After sorting:
    - The first element is the minimum.
    - The last element is the maximum.

Time Complexity

- Worst-case: O(n log n)
- Extraction of min/max: O(1)

Recurrence Relation

- $T(n)=2T(n/2)+O(n)$

Pros

- Efficient for large sorted matrices.
- Smart elimination reduces unnecessary comparisons.
- Saves time and is space-efficient (O(1) space).

Cons

- Works only if the matrix is sorted both row-wise and column-wise.
- Slightly more complex logic compared to brute force.

Feature	Brute Force	Presorting Based	Divide and conquer
Input Used	Array	Array	Array
Time Complexity	O(n)	O(n log n)	O(n)
Recurrence	2(n-1)	2T(n/2)+O(n)	C(⌊n/2⌋)+C(⌈n/2⌉)+2
Efficiency	Moderate	Lower	Highest
Logic	Full scan	Top-right elimination	Recursiom
Logic Complexity	Very Simple	Simple(with sort)	Moderate
Strength	Universally applicable	Optimized for sorted matrix	Optimal

ALGORITHM ANALYSIS

BRUTE FORCE APPROACH

Input

[7, 2, 9, 4]

Method:

Check each element matrix one by one.

Iterations

- Compare 2: min=2, max=7
- Compare 9: min=2, max=9
- Compare 4: min=2, max=9

Process

1. Set min/max to 7.
2. Check 2: update min to 2.
3. Check 9: update max to 9.
4. Check 4: no change.

Characteristics

- Direct scan.
- Does not use any sorting.
- Number of Comparisons:2(n-1)=6 for n=4.
- Time Complexity**:** O(n)

PRESORTING BASED APPROACH

Input

[7, 2, 9, 4]

Method:

Sort then pick first and last

Iterations

- Sort:[2,4,7,9]
- Min=2(First) , Max=9(Last)

Process

1. Set array.
2. Pick arr[0] and arr[n-1].

Characteristics

- Sorting overhead for just min/max.
- Number of Comparisons:O(n log n) from sort.

DIVIDE AND CONQUER APPROACH

Input

[7 , 2, 9 ,4]

Method:

Break into subarrays, combine min/max.

Step-by-Step Execution

1. Divide: [7,2] and [9,4]
2. [7,2]: 1 comparison → min=2, max=7
3. [9,4]: 1 comparison → min=4, max=9
4. Combine: compare min(2,4)→2; max(7,9)→9 (2 comparisons)
    - Total comparisons: 1+1+2=4

Process

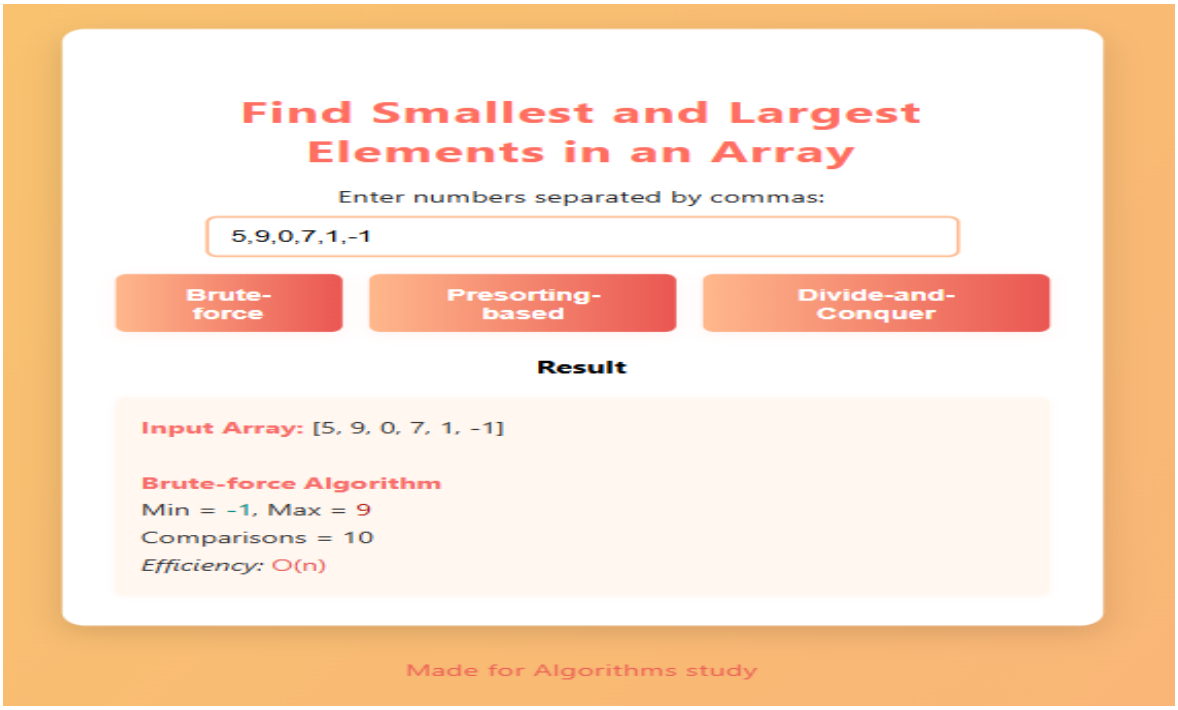
1. Divide problem recursively.
2. Solve for halves.
3. Combine.

Characteristics

- Uses recursion.
- Fewer comparisons than brute force.
- Time Complexity: O(n)

Chapter 5

RESULTS



GITHUB LINK: https://dhivyashri-6.github.io/DAA-MINIPROJECT/