# Kongu Engineering College KONGU ENGINEERING COLLEGE Kongu Engineering College (Autonomous)

(Autonomous) Perundurai,Erode – 638060

## DEPARTMENT OF INFORMATION TECHNOLOGY

**GCD CALCULATOR WITH ALGORITHM COMPARISON**

**A MICRO PROJECT REPORT FOR**

**DESIGN AND ANALYSIS OF ALGORITHMS(22ITT31)**

**SUBMITTED BY**

**RAVIBHARATHI E P(23ITR128)**

# Kongu Engineering College (Autonomous)Kongu Engineering College KONGU ENGINEERING COLLEGE

(Autonomous) Perundurai,Erode – 638060

## DEPARTMENT OF INFORMATION TECHNOLOGY

**GCD CALCULATOR WITH ALGORITHM COMPARISON**

**A MICRO PROJECT REPORT FOR**

**DESIGN AND ANALYSIS OF ALGORITHMS(22ITT31)**

**SUBMITTED BY**

**RAVIBHARATHI E P(23ITR128)**

# Kongu Engineering College (Autonomous)Kongu Engineering College KONGU ENGINEERING COLLEGE

### (Autonomous)

Perundurai,Erode – 638060

## DEPARTMENT OF INFORMATION TECHNOLOGY BONAFIED CERTIFICATE

|  |  |
| --- | --- |
| Name | : RAVIBHARATHI E P |
| Course Code | : 22ITT31 |
| Course Name | : DESIGN AND ANALYSIS OF ALGORITHMS |
| Semester | : IV |

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

Submitted for the Viva Voice Examination held on

Faculty Incharge Head of the Department

## ABSTRACT

This project is a web-based GCD (Greatest Common Divisor) Calculator that allows users to compute the GCD of two positive integers using three different algorithms: Euclid's Algorithm, Consecutive Integer Checking, and the Middle School Procedure. It features a modern, responsive user interface where users can input values, select an algorithm, and instantly receive results along with a performance comparison based on average execution time over multiple iterations. Additionally, the tool provides brief educational descriptions of each algorithm to help users understand their working and efficiency. By combining functionality with algorithmic insight, this project serves both practical and educational purposes, highlighting the impact of algorithm selection on computational performance. The application is designed for ease of use, making it suitable for both students and professionals. It encourages users to explore how algorithm efficiency can vary with different inputs. Future enhancements may include support for larger datasets and visualized algorithm steps for better clarity.

## TABLE OF CONTENTS

|  |  |  |
| --- | --- | --- |
| **CHAPTER NO** | **TITLE** | **PAGE NO** |
|  | **ABSTRACT** | **ix** |
| **1.** | **INTRODUCTION** | 6 |
|  | 1.1 PURPOSE | 7 |
|  | 1.2 OBJECTIVE | 7 |
|  | 1.3 METHODOLOGY OVERVIEW | 8 |
| **2.** | **PROBLEM STATEMENT** | 9 |
| **3.** | **METHODOLGY** | 10 |
|  | 3.1 Input & Initialization | 10 |
|  | 3.2 Divide & Compare | 10 |
|  | 3.3 Recursive Detection | 10 |
|  | 3.4 Visualization & Output | 10 |
| **4.** | **IMPLEMENTATION** | 11 |
|  | 4.1 Input & Initialization | 11 |
|  | 4.2 Divide & Compare | 11 |
|  | 4.3 Recursive Detection | 12 |
|  | 4.4 Visualization & Output | 12 |
| **5.** | **RESULTS** | 15 |

* 1. **INTRODUCTION**

The Greatest Common Divisor (GCD) of two integers is the largest number that divides both without leaving a remainder. It is a fundamental concept in number theory with applications in cryptography, computer science, and mathematics. This project aims to build a web-based GCD Calculator that allows users to compute the GCD of two positive integers using three distinct algorithms: Euclid's Algorithm, Consecutive Integer Checking, and the Middle School Procedure. Each algorithm varies in complexity and efficiency, providing an opportunity to compare their performance in real-time. The calculator is designed with an intuitive and responsive interface to facilitate easy input, algorithm selection, and instant result display. Additionally, it includes educational descriptions of the algorithms, making it a useful learning tool for students and enthusiasts alike. This project demonstrates not only the calculation of the GCD but also the importance of algorithmic efficiency in solving mathematical problems.

* 1. **PURPOSE**

The purpose of this project is to develop an interactive and user-friendly web-based tool that calculates the Greatest Common Divisor (GCD) of two positive integers using multiple algorithms. It aims to demonstrate the differences in algorithmic approaches and their computational efficiencies by providing real-time performance comparisons. Additionally, the project serves an educational purpose by explaining each algorithm’s logic and time complexity, helping users deepen their understanding of fundamental mathematical concepts and algorithm design. Ultimately, the tool encourages users to appreciate the impact of algorithm selection on problem-solving efficiency in computer science and mathematics.

* 1. **OBJECTIVE**
* To implement a web-based calculator that computes the Greatest Common Divisor (GCD) of two positive integers using multiple algorithms.
* To compare the efficiency of Euclid's Algorithm, Consecutive Integer Checking, and the Middle School Procedure by measuring their execution times.
* To provide users with educational insights into how each algorithm works and their time complexities.
* To design a clean, intuitive user interface that allows easy input, algorithm selection, and displays results along with performance data.
* To promote understanding of algorithmic efficiency and encourage learning through interactive computation.

**METHODOLOGY OVERVIEW**

The project employs a structured approach to calculate the Greatest Common Divisor (GCD) of two positive integers using multiple algorithms. The methodology includes:

1. Input Collection:  
   Users enter two positive integers and select the preferred algorithm (Euclid’s Algorithm, Consecutive Integer Checking, or Middle School Procedure) via a clean, responsive web interface.
2. Input Validation:  
   The application verifies that the inputs are valid positive integers before proceeding to the computation to prevent errors or invalid operations.
3. Algorithm Execution:  
   Depending on the chosen method, the corresponding algorithm is executed in JavaScript to compute the GCD:
   * *Euclid’s Algorithm* uses repeated division and remainder calculation.
   * *Consecutive Integer Checking* iteratively checks divisors starting from the smaller number.
   * *Middle School Procedure* factors both numbers into primes and multiplies the common factors.
4. Performance Measurement:  
   To ensure accurate timing, the algorithm runs multiple iterations (e.g., 1000 times), and the average execution time is calculated in microseconds to compare efficiency.
5. Result Display:  
   The calculated GCD, performance statistics, and a comparison table of all algorithms run so far are displayed to the user, highlighting the fastest and slowest methods.
6. Educational Descriptions:  
   Brief, clear explanations of each algorithm’s logic and time complexity are provided dynamically, helping users understand the underlying concepts.
7. User Interface:  
   The entire process is wrapped in a visually appealing and intuitive design with smooth interactions, responsive input fields, and animated result displays to enhance user experience.
8. **PROBLEM STATEMENT**

Finding the Greatest Common Divisor (GCD) of two numbers is a fundamental problem in mathematics and computer science with applications in cryptography, number theory, and algorithm design. While there are multiple algorithms to compute the GCD, they differ significantly in terms of efficiency and complexity. Beginners often struggle to understand these differences and how they impact performance. This project aims to provide a user-friendly tool that not only calculates the GCD using different well-known algorithms but also educates users by showing performance comparisons and algorithm explanations. The challenge lies in accurately implementing and measuring these algorithms’ efficiency in a web environment, and presenting the results in an intuitive, informative way.

## METHODOLOGY:

Methodology (Algorithm Implementation and Performance Analysis):

1. **Input Collection:**  
   The system accepts two positive integers as input from the user via a web interface. The user also selects one of the three GCD algorithms (Euclid’s Algorithm, Consecutive Integer Checking, Middle School Procedure) for computation.
2. **Input Validation:**  
   Validate that both inputs are positive integers. If invalid input is detected, the system prompts the user to enter correct values.
3. **Algorithm Execution:**  
   Depending on the user’s choice, the system executes the selected algorithm to compute the GCD:

* **Euclid’s Algorithm:** Uses repeated modulo operations to find the GCD efficiently.
* **Consecutive Integer Checking:** Checks divisors starting from the smaller number downward until the GCD is found.
* **Middle School Procedure:** Performs prime factorization on both numbers and multiplies the common factors to determine the GCD.

1. **Performance Measurement:**  
   The system runs the selected algorithm multiple times (e.g., over several iterations) to measure average execution time, providing a quantitative performance comparison among the algorithms.
2. **Result Display and Explanation:**  
   The system displays the computed GCD, the average execution time, and a brief educational description of the algorithm used, helping users understand its working and efficiency.
3. **IMPLEMENTATION :**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>GCD Calculator</title>

    <link href="https://fonts.googleapis.com/css2?family=Inter:wght@400;500;600;700&display=swap" rel="stylesheet">

    <style>

        body {

            font-family: 'Inter', sans-serif;

            margin: 0;

            padding: 0;

            background: linear-gradient(135deg, #6366f1, #a855f7);

            color: #fff;

            display: flex;

            justify-content: center;

            align-items: center;

            height: 100vh;

            line-height: 1.6;

        } .container {

            background: rgba(255, 255, 255, 0.1);

            backdrop-filter: blur(10px);

            padding: 2.5rem;

            border-radius: 24px;

            box-shadow: 0 20px 40px rgba(0, 0, 0, 0.2);

            width: 90%;

            max-width: 450px;

            text-align: center;

            border: 1px solid rgba(255, 255, 255, 0.2);

        }

        h1 {

            margin-bottom: 1.5rem;

            font-weight: 700;

            font-size: 2.2rem;

            color: #ffffff;

            text-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);

        }

        .input-group {

            margin-bottom: 1rem;

        }input, select {

            width: 100%;

            padding: 1rem;

            margin: 0.5rem 0;

            border: 2px solid rgba(255, 255, 255, 0.2);

            border-radius: 12px;

            font-size: 1rem;

            background: rgba(255, 255, 255, 0.9);

            color: #333;

            transition: all 0.3s ease;

            box-sizing: border-box;

        }     input:focus, select:focus {

            outline: none;

            border-color: rgba(255, 255, 255, 0.5);

            box-shadow: 0 0 0 3px rgba(255, 255, 255, 0.1);

        }

        select {

            appearance: none;

            background-image: url("data:image/svg+xml,%3Csvg xmlns='http://www.w3.org/2000/svg' width='24' height='24' viewBox='0 0 24 24' fill='none' stroke='%23333333' stroke-width='2' stroke-linecap='round' stroke-linejoin='round'%3E%3Cpolyline points='6 9 12 15 18 9'%3E%3C/polyline%3E%3C/svg%3E");

            background-repeat: no-repeat;

            background-position: right 1rem center;

            background-size: 1.2em;

            padding-right: 2.5rem;

        }

        button {

            width: 100%;

            padding: 1rem;

            margin-top: 1rem;

            border: none;

            border-radius: 12px;

            font-size: 1.1rem;

            font-weight: 600;

            color: #fff;

            background: linear-gradient(135deg, #3b82f6, #2563eb);

            cursor: pointer;

            transition: transform 0.2s ease, box-shadow 0.2s ease;

            box-shadow: 0 4px 12px rgba(37, 99, 235, 0.2);

        }  button:hover {

            transform: translateY(-2px);

            box-shadow: 0 6px 16px rgba(37, 99, 235, 0.3);

        } button:active {

            transform: translateY(0);

        }#result {

            margin-top: 1.5rem;

            padding: 1.2rem;

            background: rgba(255, 255, 255, 0.95);

            color: #1f2937;

            border-radius: 16px;

            font-size: 1.25rem;

            font-weight: 600;

            box-shadow: 0 8px 16px rgba(0, 0, 0, 0.1);

            display: none;

            animation: fadeIn 0.3s ease; }

        .performance-comparison {

            margin-top: 1rem;

            padding: 1rem;

            background: rgba(255, 255, 255, 0.95);

            border-radius: 12px;

            color: #1f2937;

            font-size: 0.9rem;

            display: none;

            animation: fadeIn 0.3s ease;

        } .performance-comparison table {

            width: 100%;

            border-collapse: collapse;

            margin-top: 0.5rem;

        } .performance-comparison th,

        .performance-comparison td {

            padding: 0.5rem;

            text-align: left;

            border-bottom: 1px solid rgba(0, 0, 0, 0.1);

        }

        .performance-comparison th {

            font-weight: 600;

            color: #4b5563;

        }  .fastest {

            color: #059669;

            font-weight: 600;

        }.slowest {

            color: #dc2626;

        }  @keyframes fadeIn {

            from { opacity: 0; transform: translateY(10px); }

            to { opacity: 1; transform: translateY(0); }

        }

        ::placeholder {

            color: #9ca3af;

        }.algorithm-info {

            background: rgba(255, 255, 255, 0.95);

            padding: 1rem;

            border-radius: 12px;

            margin-top: 1rem;

            color: #1f2937;

            font-size: 0.9rem;

            text-align: left;

            display: none;

        }

        .algorithm-info.active {

            display: block;

            animation: fadeIn 0.3s ease;

        }  .algorithm-info h3 {

            margin: 0 0 0.5rem 0;

            color: #4b5563;

        }

        .algorithm-info p {

            margin: 0;

            line-height: 1.5;

        }

    </style>

</head>

<body>

    <div class="container">

        <h1>GCD Calculator</h1>

        <div class="input-group">

            <input type="number" id="num1" placeholder="Enter first number" min="1" required>

        </div>

        <div class="input-group">

            <input type="number" id="num2" placeholder="Enter second number" min="1" required>

        </div>

        <div class="input-group">

            <select id="method" onchange="showAlgorithmInfo()">

                <option value="euclid">Euclid's Algorithm (Most Efficient)</option>

                <option value="consecutive">Consecutive Integer Checking (Simple)</option>

                <option value="middle">Middle School Procedure (Educational)</option>

            </select>

        </div>

        <div id="algorithmInfo" class="algorithm-info">

            <h3>Algorithm Description</h3>

            <p id="algorithmDescription"></p>

        </div>

        <button onclick="calculateGCD()">Calculate GCD</button>

        <div id="result"></div>

        <div id="performanceComparison" class="performance-comparison"></div>

    </div>

    <script>

        let performanceHistory = {};

        function calculateGCD() {

            let a = parseInt(document.getElementById('num1').value);

            let b = parseInt(document.getElementById('num2').value);

            let method = document.getElementById('method').value;

            let output;

            if (isNaN(a) || isNaN(b) || a <= 0 || b <= 0) {

                output = "Please enter valid positive integers.";

            } else {

                const iterations = 1000;

                const startTime = performance.now();

                for (let i = 0; i < iterations; i++) {

                    if (method === "euclid") {

                        output = gcdEuclid(a, b);

                    } else if (method === "consecutive") {

                        output = gcdConsecutive(a, b)                }

else {

                        output = gcdMiddleSchool(a, b);

                    }  }

                const endTime = performance.now();

                const totalExecutionTime = endTime - startTime;

                const executionTime = (totalExecutionTime / iterations) \* 1000;

                performanceHistory[method] = {

                    time: executionTime,

                    result: output,

                    numbers: [a, b]

                };

                output = `GCD: ${output}`;

                showPerformanceComparison();

            }

            const resultBox = document.getElementById('result');

            resultBox.innerText = output;

            resultBox.style.display = 'block';

        }

        function showPerformanceComparison() {

            const comparisonDiv = document.getElementById('performanceComparison');

            if (Object.keys(performanceHistory).length === 0) return;

            let html = '<h3>Performance Comparison</h3>';

            html += '<table>';

            html += '<tr><th>Algorithm</th><th>Time (μs)</th><th>Numbers</th><th>Result</th></tr>';

            const sortedAlgorithms = Object.entries(performanceHistory)

                .sort((a, b) => a[1].time - b[1].time);

            const algorithmNames = {

                euclid: "Euclid's Algorithm",

                consecutive: "Consecutive Integer",

                middle: "Middle School"

            }:

sortedAlgorithms.forEach(([algorithm, data], index) => {

                const className = index === 0 ? 'fastest' :

                                index === sortedAlgorithms.length - 1 ? 'slowest' : ''

                let timeDisplay;

                if (data.time < 1) {

                    timeDisplay = data.time.toFixed(4);

                } else if (data.time < 10) {

                    timeDisplay = data.time.toFixed(3);

                } else {

                    timeDisplay = data.time.toFixed(1);

                }

                html += `

                    <tr class="${className}">

                        <td>${algorithmNames[algorithm]}</td>

                        <td>${timeDisplay}</td>

                        <td>${data.numbers.join(', ')}</td>

                        <td>${data.result}</td>

                    </tr>

                `;

            });

            html += '</table>';

            if (sortedAlgorithms.length > 1) {

                const fastest = sortedAlgorithms[0];

                const slowest = sortedAlgorithms[sortedAlgorithms.length - 1];

                const speedDiff = slowest[1].time / fastest[1].time;

                let comparisonText;

                if (speedDiff > 1000) {

                    comparisonText = `${(speedDiff/1000).toFixed(1)}k times`;

                } else {

                    comparisonText = `${speedDiff.toFixed(1)}x`;

                }

                html += `

                    <p style="margin-top: 1rem">

                        ${algorithmNames[fastest[0]]} was ${comparisonText} faster than

                        ${algorithmNames[slowest[0]]} for these numbers.

                    </p> `;

            }

            html += `

                <p style="margin-top: 0.5rem; font-size: 0.8rem; color: #666;">

                    \* Times shown are averages over 1000 iterations to ensure accuracy.

                    μs = microseconds (1 millionth of a second)

                </p>`;

            comparisonDiv.innerHTML = html;

            comparisonDiv.style.display = 'block';

        }function gcdEuclid(a, b) {

            while (b !== 0) {

                let r = a % b;

                a = b;

                b = r;

            }

            return a;

        }

        function gcdConsecutive(a, b) {

            let min = Math.min(a, b);

            while (min > 0) {

                if (a % min === 0 && b % min === 0) return min;

                min--;

            } }

        function primeFactors(n) {

            let factors = [];

            for (let i = 2; i <= n; i++) {

                while (n % i === 0) {

                    factors.push(i);

                    n /= i;

                } }

            return factors;

        }

        function gcdMiddleSchool(a, b) {

            let aFactors = primeFactors(a);

            let bFactors = primeFactors(b);

            let common = [];

            for (let i = 0; i < aFactors.length; i++) {

                let index = bFactors.indexOf(aFactors[i]);

                if (index !== -1) {

                    common.push(aFactors[i]);

                    bFactors.splice(index, 1);

                }

            }

            return common.reduce((acc, val) => acc \* val, 1) || 1;

        }

        function showAlgorithmInfo() {

            const method = document.getElementById('method').value;

            const description = document.getElementById('algorithmDescription');

            const infoBox = document.getElementById('algorithmInfo');

            const descriptions = {

                euclid: "Most efficient method that uses division and remainder. Repeatedly divides the larger number by the smaller one until remainder is zero. Time complexity: O(log min(a,b)).",

                consecutive: "Simple but less efficient method that checks each number from the minimum of the two numbers down to 1, finding the largest common divisor. Time complexity: O(min(a,b)).",

                middle: "Educational method that finds prime factors of both numbers and multiplies their common factors. Similar to how it's taught in schools. Time complexity: O(√max(a,b))."

            };

description.textContent = descriptions[method];

            infoBox.classList.add('active');

        }

        showAlgorithmInfo();

    </script>

</body>

</html>

**DIFFERENCE BETWEEN EUCLID'S ALGORITHM, CONSECUTIVE INTEGER CHECKING, AND MIDDLE SCHOOL PROCEDURE FOR GCD CALCULATION**

**Euclid's Algorithm (Efficient Division-Based Approach):**

* Uses the property that GCD divides the remainder; repeatedly replaces the larger number by the remainder until zero.
* **Time Complexity:** O(log min(a, b)) — very efficient for large numbers.

**Consecutive Integer Checking (Brute Force Linear Scan):**

* Starts from the smaller number and checks downwards to find the largest divisor common to both numbers.
* **Time Complexity:** O(min(a, b)) — inefficient for large inputs.

**Middle School Procedure (Prime Factorization):**

* Finds prime factors of both numbers, multiplies the common factors to get the GCD.
* **Time Complexity:** Approximately O(√max(a, b)) due to factorization.

**Space Complexity:**

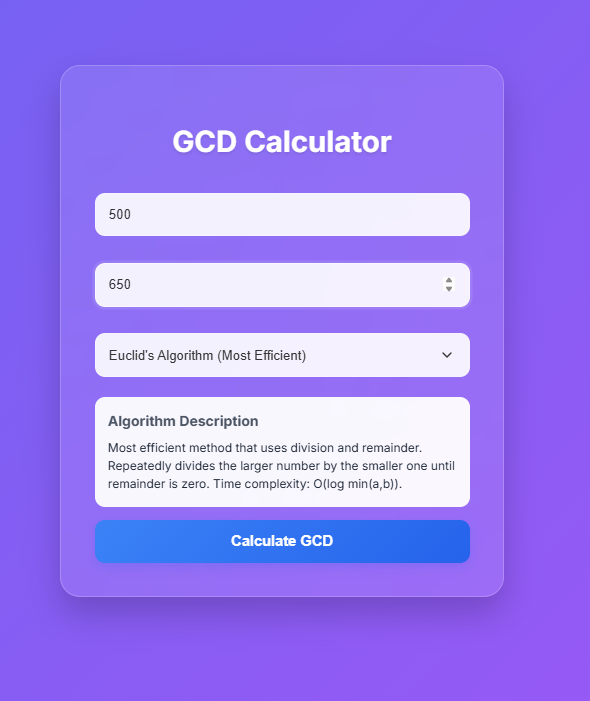
* All methods use constant additional space (O(1)) except middle school procedure, which requires O(k) space to store prime factors, where k is the number of factors.

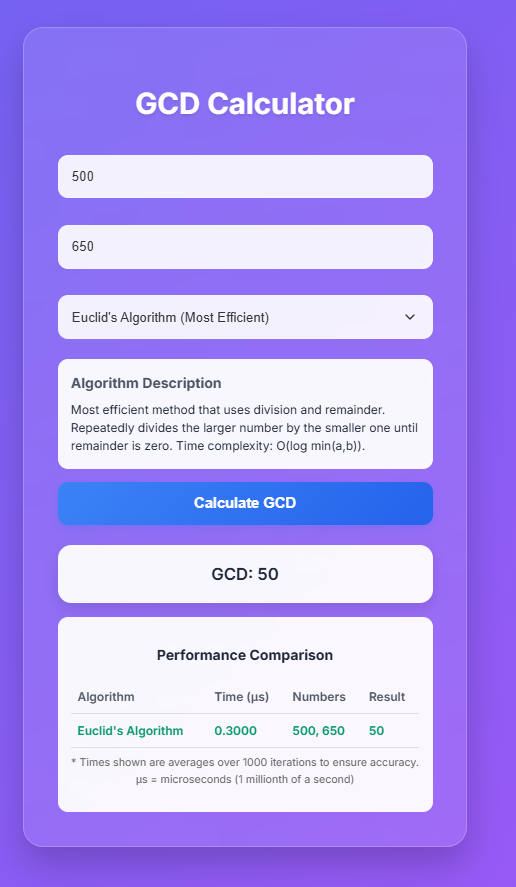
**Pros and Cons:**

| **Approach** | Pros | Cons |
| --- | --- | --- |
| **Euclid's Algorithm** | Very efficient and fast; minimal computations | Slightly more abstract conceptually |
| **Consecutive Integer** | Simple to understand and implement | Very slow for large numbers due to brute force |
| **Middle School** | Educational; shows factorization process | Slower than Euclid, requires factorization, more complex |
| **Basic Idea** | Repeatedly divides and takes remainders | More complex than brute force |
| **Number of Comparisons** | Logarithmic number of division steps | Higher in brute force methods |
| **Time Complexity** | O(log min(a, b)) | Slower in brute-force and factorization-based approaches |
| **Space Complexity** | O(1) (constant space) | Can grow with number of prime factors (Middle School) |
| **Implementation Complexity** | Moderate due to algorithmic logic | Higher for factorization; simpler for brute force |
| **Performance** | Fastest and most practical | Brute force and factorization slower for large inputs |

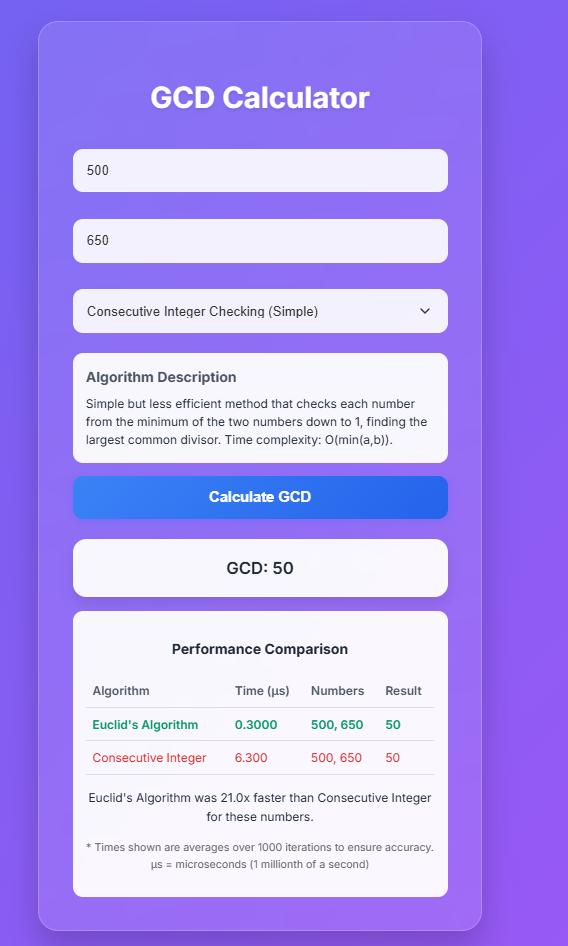
## 5.0. RESULTS:3

1. **Euclid’s algorithm:**

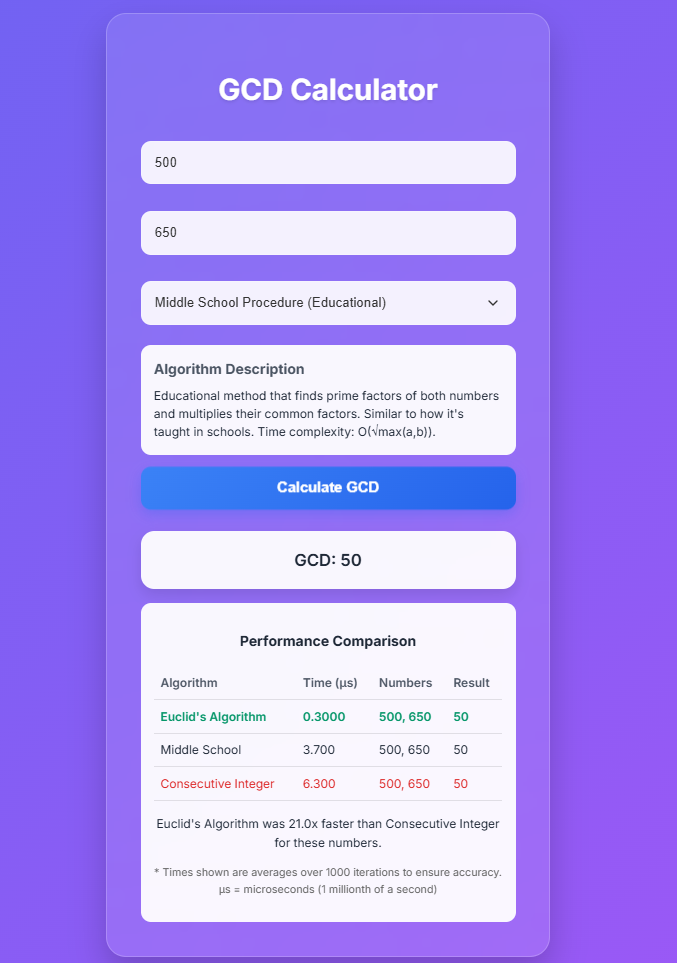
****

****

**2.Consecutive Integer :**

****

**3.Middle School :**

****

**GITHUB LINK:**  **https://github.com/Ravibharathi13/DAA\_project**