**Institute of Computer Technology**

**B. Tech Computer Science and Engineering**

**Sub: Algorithm Analysis and Design**

**Practical 2**

**(1)** MPSoft Technologies Pvt. Ltd. is a fast growing IT industry and wants to implement a function to calculate the monthly income generated from all projects from their N no of clients like C1,C2,C3,C4….CN. The team wants to compare the time/steps required to execute this function on various inputs and analyse the complexity of each combination. Also draw a comparative chart. In each of the following functions N will be passed by user.

Design the algorithm for the same and implement using the programming language of your choice. Make comparative analysis for various use cases & input size.

1. To calculate the sum of 1 to N number using loop.
2. To calculate the sum of 1 to N number using the equation.
3. To calculate sum of 1 to N numbers using recursion

 Code:

App.py: (Combine code of task 1 and 2)

from flask import Flask, render\_template, request, redirect, url\_for, Response

import time

import matplotlib.pyplot as plt

import io

from matplotlib.backends.backend\_agg import FigureCanvasAgg as FigureCanvas

import sys

app = Flask(\_\_name\_\_)

sys.setrecursionlimit(1000000)

*# Task 1 Functions*

def sum\_using\_loop(N):

    total = 0

    for i in range(1, N + 1):

        total += i

    return total

def sum\_using\_equation(N):

    return N \* (N + 1) // 2

def sum\_using\_recursion(N):

    if N == 1:

        return 1

    return N + sum\_using\_recursion(N - 1)

*# Task 2 Functions*

def fibonacci\_iterative(n):

    if n <= 1:

        return n

    a, b = 0, 1

    for \_ in range(2, n + 1):

        a, b = b, a + b

    return b

def fibonacci\_recursive(n):

    if n <= 1:

        return n

    if n > 30:

        raise RecursionError("Recursion depth limit exceeded.")

    return fibonacci\_recursive(n - 1) + fibonacci\_recursive(n - 2)

*# Utility Function to Measure Time*

def measure\_time(func, n):

    start\_time = time.time()

    try:

        result = func(n)

        if isinstance(result, int):

            end\_time = time.time()

            return end\_time - start\_time, result

        else:

            return float('inf'), None

    except RecursionError:

        return float('inf'), None

*# Main Route*

@app.route("/", methods=["GET", "POST"])

def main():

*# Task 1 - Sum of Numbers*

    if request.method == "POST":

        if 'generate' in request.form:

            return redirect(url\_for('task1'))

*# Task 1 Processing*

        if 'N' in request.form:

            try:

                N = int(request.form["N"])

                if N < 0:

                    return render\_template("task1.html", error="Please enter a non-negative integer.", N=None)

                loop\_time, loop\_result = measure\_time(sum\_using\_loop, N)

                equation\_time, equation\_result = measure\_time(sum\_using\_equation, N)

                recursion\_time, recursion\_result = measure\_time(sum\_using\_recursion, N)

                recursion\_data = (recursion\_result, recursion\_time) if recursion\_result is not None else ("Skipped due to depth limit", None)

                return render\_template(

                    "task1.html",

                    N=N,

                    loop\_data=(loop\_result, loop\_time),

                    equation\_data=(equation\_result, equation\_time),

                    recursion\_data=recursion\_data,

                    error=None

                )

            except ValueError:

                return render\_template("index.html", error="Invalid input. Please enter an integer.", N=None)

*# Task 2 Processing*

        elif 'n' in request.form:

            try:

                n = int(request.form["n"])

                if n < 0:

                    return render\_template("task2.html", error="Please enter a non-negative integer.", n=None)

                iterative\_time, iterative\_result = measure\_time(fibonacci\_iterative, n)

                recursive\_time, recursive\_result = measure\_time(fibonacci\_recursive, n)

                iterative\_data = (iterative\_result, iterative\_time)

                if recursive\_result is None:

                    recursive\_data = (None, float('inf'))

                else:

                    recursive\_data = (recursive\_result, recursive\_time)

                return render\_template(

                    "task2.html",

                    n=n,

                    iterative\_data=iterative\_data,

                    recursive\_data=recursive\_data,

                    error=None

                )

            except ValueError:

                return render\_template("index.html", error="Invalid input. Please enter an integer.", n=None)

    return render\_template("index.html", N=None, error=None)

*# Task 1 Route*

@app.route('/task1', methods=["GET", "POST"])

def task1():

    if request.method == "POST":

        return render\_template('task1.html')

    return render\_template('task1.html')

@app.route("/practical\_1.png")

def practical\_1():

    input\_sizes = [100, 1000, 5000, 10000, 20000, 50000, 100000]

    loop\_times = []

    equation\_times = []

    recursion\_times = []

    for size in input\_sizes:

        loop\_times.append(measure\_time(sum\_using\_loop, size)[0])

        equation\_times.append(measure\_time(sum\_using\_equation, size)[0])

        recursion\_times.append(measure\_time(sum\_using\_recursion, size)[0])

    fig, ax = plt.subplots(figsize=(12, 6))

    ax.plot(input\_sizes, loop\_times, label='Loop', marker='o')

    ax.plot(input\_sizes, equation\_times, label='Equation', marker='o')

    ax.plot(input\_sizes, recursion\_times, label='Recursion', marker='o')

    ax.set\_xlabel('Input Size (N)')

    ax.set\_ylabel('Execution Time (seconds)')

    ax.set\_title('Comparison of Execution Time for Sum of 1 to N')

    ax.legend()

    ax.grid(True)

    output = io.BytesIO()

    FigureCanvas(fig).print\_png(output)

    return Response(output.getvalue(), mimetype='image/png')

*# Task 2 Route*

@app.route('/task2', methods=["GET", "POST"])

def task2():

    iterative\_data = None

    recursive\_data = None

    n = None

    error = None

    if request.method == "POST":

        try:

            n = int(request.form["n"])

            if n < 0:

                error = "Please enter a non-negative integer."

            else:

                iterative\_time, iterative\_result = measure\_time(fibonacci\_iterative, n)

                recursive\_time, recursive\_result = measure\_time(fibonacci\_recursive, n)

                iterative\_data = (iterative\_result, iterative\_time)

                if recursive\_result is None:

                    recursive\_data = (None, float('inf'))

                else:

                    recursive\_data = (recursive\_result, recursive\_time)

        except ValueError:

            error = "Invalid input. Please enter an integer."

    return render\_template(

        "task2.html",

        n=n,

        iterative\_data=iterative\_data,

        recursive\_data=recursive\_data,

        error=error

    )

@app.route("/practical\_2.png")

def practical\_2():

    input\_sizes = [5, 10, 15, 20, 25, 30, 35]

    iterative\_times = []

    recursive\_times = []

    for size in input\_sizes:

        iterative\_times.append(measure\_time(fibonacci\_iterative, size)[0])

        recursive\_times.append(measure\_time(fibonacci\_recursive, size)[0])

    fig, ax = plt.subplots(figsize=(12, 6))

    ax.plot(input\_sizes, iterative\_times, label='Iterative', marker='o')

    ax.plot(input\_sizes, recursive\_times, label='Recursive', marker='o')

    ax.set\_xlabel('Input Size (n)')

    ax.set\_ylabel('Execution Time (seconds)')

    ax.set\_title('Comparison of Execution Time for Fibonacci Calculation')

    ax.legend()

    ax.grid(True)

    output = io.BytesIO()

    FigureCanvas(fig).print\_png(output)

    return Response(output.getvalue(), mimetype='image/png')

if \_\_name\_\_ == "\_\_main\_\_":

    app.run(debug=True)

Task1.html:

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

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

    <title>Sum Calculation Methods</title>

</head>

<body>

    <h1>Sum Calculation Methods</h1>

    <form method="post" action="/">

        <label for="N">Enter a value for N:</label>

        <input type="text" id="N" name="N" required>

        <button type="submit">Calculate</button>

    </form>

    {% if error %}

        <p style="color:red;">{{ error }}</p>

    {% endif %}

    {% if N is not none %}

        <h2>Results for N = {{ N }}</h2>

        {% if loop\_data %}

            <p>Sum using loop: {{ loop\_data[0] }} (Time taken: {{ loop\_data[1] }} seconds)</p>

        {% endif %}

        {% if equation\_data %}

            <p>Sum using equation: {{ equation\_data[0] }} (Time taken: {{ equation\_data[1] }} seconds)</p>

        {% endif %}

        {% if recursion\_data %}

            <p>Sum using recursion: {{ recursion\_data[0] }}

            {% if recursion\_data[1] is not none %}

                (Time taken: {{ recursion\_data[1] }} seconds)

            {% else %}

                (Skipped due to recursion limit)

            {% endif %}

            </p>

        {% endif %}

    {% endif %}

    <h2>Time Complexity Graphs</h2>

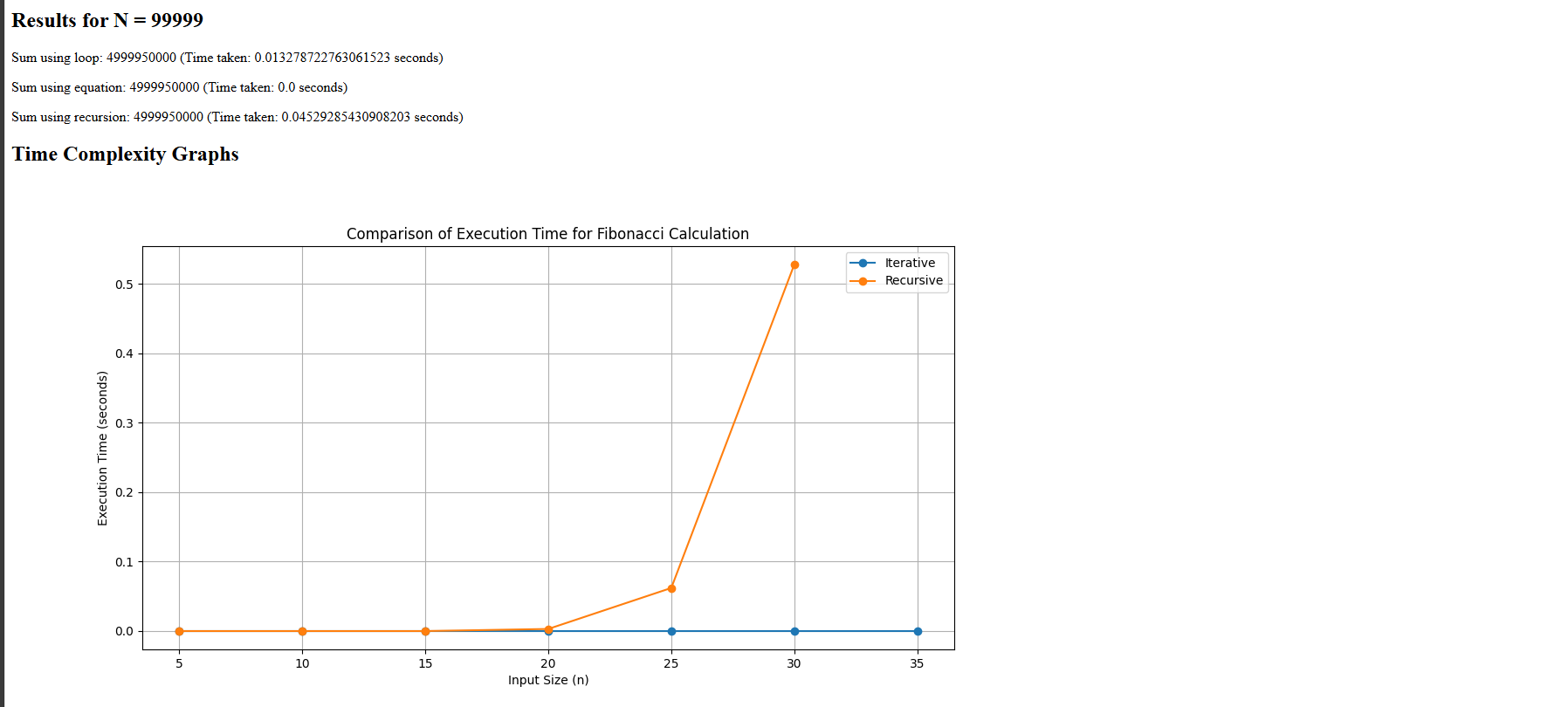
    <img src="{{ url\_for('practical\_2') }}" alt="Time Complexity Graphs">

</body>

</html>

Output:





**(2)** Suppose a newly-born pair of rabbits, one male, one female, are put in a field. Rabbits are able to mate at the age of one month so that at the end of its second month a female can produce another pair of rabbits. Suppose that our rabbits never die and that the female always produces one new pair (one male, one female) every month from the second month on. How many pairs will there be in one year? Apply appropriate algorithm/method to find out the above problem and also solve them using iteration and recursive method. Compare the performance of two methods by counting the number of steps executed on various inputs. Also draw a comparative chart.

Design the algorithm for the same and implement using the programming language of your choice. Make comparative analysis for various use cases & input size.

Task2.html:

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

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

    <title>Fibonacci Calculation</title>

  </head>

  <body>

    <h1>Fibonacci Calculation</h1>

    <form method="post">

      <label for="n">Enter a value for n:</label>

      <input type="number" id="n" name="n" required />

      <button type="submit">Calculate</button>

    </form>

    <h2>Results for n = {{ n }}:</h2>

    <p>

      Iterative Fibonacci Result: {{ iterative\_data[0] }} (Time taken: {{

      iterative\_data[1] }} seconds)

    </p>

    <p>

      Recursive Fibonacci Result: {{ recursive\_data[0] }} (Time taken: {{

      recursive\_data[1] }} seconds)

    </p>

    <p>Recursive calculation took too long or exceeded limits.</p>

    <p style="color: red">{{ error }}</p>

    <h2>Execution Time Comparison</h2>

    <img

      src="{{ url\_for('practical\_2') }}"

      alt="Execution Time

 Comparison"

    />

  </body>

</html>

Output:

