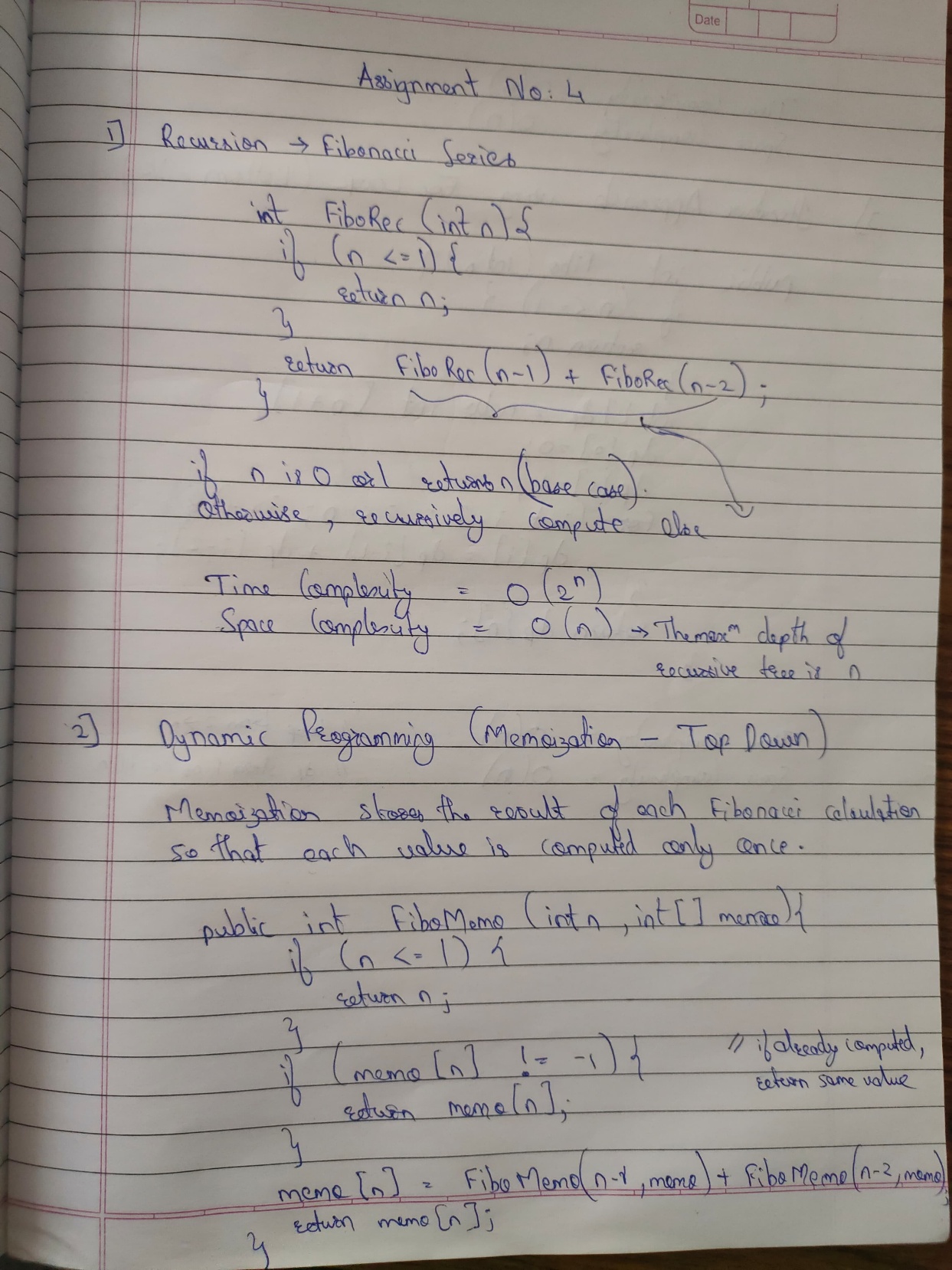
**Experiment Number: 4**

Problem Statement: Compare dynamic programming (top down, bottom up) and divide and conquer for generating Fibonacci sequence. Perform Time complexity analysis and plot graph of actual execution time required for all the approaches.

NAME: Harshwardhan PatilROLLNO: 50

CLASS: IT-B BATCH: B1

A notebook with writing on it

AI-generated content may be incorrect.A notebook with writing on it

AI-generated content may be incorrect. **CODE :**

import java.io.FileWriter;

import java.io.IOException;

import java.util.Arrays;

public class FibonacciTimeMeasurement {

    private static final int MAX\_N = 40;

    // Function to get the current time in nanoseconds

    private static double getTimeNs() {

        return System.nanoTime() / 1e9;  // Convert nanoseconds to seconds

    }

    // Recursive Fibonacci (Exponential time complexity: O(2^n))

    private static long recursiveFib(int n) {

        if (n <= 1) return n;

        return recursiveFib(n - 1) + recursiveFib(n - 2);

    }

    // Memoization (Top-Down DP) (Time Complexity: O(n))

    private static long memoizedFib(int n, long[] memo) {

        if (n <= 1) return n;

        if (memo[n] != -1) return memo[n];  // Return cached result

        return memo[n] = memoizedFib(n - 1, memo) + memoizedFib(n - 2, memo);

    }

    // Bottom-Up Fibonacci (O(n) time complexity)

    private static long bottomUpFib(int n) {

        if (n <= 1) return n;

        long a = 0, b = 1, c;

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

            c = a + b;

            a = b;

            b = c;

        }

        return b;

    }

    // Function to measure execution time for all Fibonacci methods

    private static void measureFibonacci() {

        try (FileWriter writer = new FileWriter("fibonacci\_times.csv")) {

            writer.write("n,Recursive,Memoization,Bottom-Up\n");

            for (int n = 1; n <= MAX\_N; n++) {

                double start, end;

                long result;

                long[] memo = new long[MAX\_N + 1];

                // Recursive Fibonacci

                start = getTimeNs();

                result = recursiveFib(n);

                end = getTimeNs();

                double recursiveTime = end - start;

                // Memoized Fibonacci

                Arrays.fill(memo, -1);

                start = getTimeNs();

                result = memoizedFib(n, memo);

                end = getTimeNs();

                double memoizedTime = end - start;

                // Bottom-Up Fibonacci

                start = getTimeNs();

                result = bottomUpFib(n);

                end = getTimeNs();

                double bottomUpTime = end - start;

                // Save to CSV

                writer.write(String.format("%d,%.9f,%.9f,%.9f\n", n, recursiveTime, memoizedTime, bottomUpTime));

                // Display output for verification

                System.out.printf("n=%d -> Recursive: %.6f s, Memoization: %.6f s, Bottom-Up: %.6f s\n",

                        n, recursiveTime, memoizedTime, bottomUpTime);

            }

            System.out.println("\nResults saved to fibonacci\_times.csv");

        } catch (IOException e) {

            System.err.println("Error writing to file: " + e.getMessage());

        }

    }

    public static void main(String[] args) {

        measureFibonacci();

    }

}.

GRAPH

import pandas as pd

import matplotlib.pyplot as plt

# Read the CSV file

df = pd.read\_csv("fibonacci\_times.csv")

# Plot execution time for each method

plt.figure(figsize=(10, 6))

plt.plot(df["n"], df["Recursive"], label="Recursive (O(2^n))", marker="o", linestyle="dotted", color="red")

plt.plot(df["n"], df["Memoization"], label="Memoization (O(n))", marker="s", linestyle="dashdot", color="green")

plt.plot(df["n"], df["Bottom-Up"], label="Bottom-Up (O(n))", marker="^", linestyle="solid", color="blue")

# Graph Labels

plt.xlabel("n (Fibonacci Number)")

plt.ylabel("Execution Time (seconds)")

plt.title("Fibonacci Execution Time Analysis")

plt.legend()

plt.yscale("log")  # Log scale for better visibility

plt.grid(True)

# Show the plot

plt.show()

OUTPUT:

