Objective: To implement and analyze different approaches for generating Fibonacci numbers and compare their time and space complexities (through graph)
Write algorithms also for each version.

4a. Recursive version
4b Iterative version
4c Dynamic Programming- Top Down Approach
4d Dynamic Programming- Bottom Up Approach

Code:

fibo.c > ...

```c
#include <stdio.h>
#include <string.h>
#include <time.h>
#define MAX 100

// -------------------- (a) Recursive --------------------
int fib_recursive(int n) {
    if (n <= 1) return n;
    return fib_recursive(n-1) + fib_recursive(n-2);
}

// -------------------- (b) Iterative --------------------
int fib_iterative(int n) {
    if (n <= 1) return n;
    int a = 0, b = 1, c;
    for (int i = 2; i <= n; i++) {
        c = a + b;
        a = b;
        b = c;
    }
    return b;
}

// -------------------- (c) DP - Top Down (Memoization) --------------------
int fib_topdown(int n, int memo[]) {
    if (memo[n] != -1) return memo[n];
    memo[n] = fib_topdown(n-1, memo) + fib_topdown(n-2, memo);
    return memo[n];
}

// -------------------- (d) DP - Bottom Up (Tabulation) --------------------
int fib_bottomup(int n) {
    if (n <= 1) return n;
    int dp[MAX];
    dp[0] = 0;
    dp[1] = 1;
    for (int i = 2; i <= n; i++) {
        dp[i] = dp[i-1] + dp[i-2];
    }
    return dp[n];
}

// -------------------- Main Function --------------------
int main() {
    int n;
    printf("Enter n for n-th term of the Fibonacci series: ");
    scanf("%d", &n);

    clock_t start, end;
    double cpu_time_used;

    // Recursive
    start = clock();
    int r1 = fib_recursive(n);
    end = clock();
    cpu_time_used = ((double)(end - start)) / CLOCKS_PER_SEC;
    printf("\n(a) Recursive: %d   | Time: %lf sec\n", r1, cpu_time_used);

    // Iterative
```

```c
int main() {
    // Iterative
    start = clock();
    int r2 = fib_iterative(n);
    end = clock();
    cpu_time_used = ((double)(end - start)) / CLOCKS_PER_SEC;
    printf("(b) Iterative: %d  | Time: %lf sec\n", r2, cpu_time_used);

    // Top-Down DP
    int memo[MAX];
    for (int i = 0; i < MAX; i++) memo[i] = -1;
    memo[0] = 0; memo[1] = 1;

    start = clock();
    int r3 = fib_topdown(n, memo);
    end = clock();
    cpu_time_used = ((double)(end - start)) / CLOCKS_PER_SEC;
    printf("(c) DP Top-Down: %d  | Time: %lf sec\n", r3, cpu_time_used);

    // Bottom-Up DP
    start = clock();
    int r4 = fib_bottomup(n);
    end = clock();
    cpu_time_used = ((double)(end - start)) / CLOCKS_PER_SEC;
    printf("(d) DP Bottom-Up: %d  | Time: %lf sec\n", r4, cpu_time_used);

    return 0;
}
```

OUTPUT:

```
(d) DP Bottom-Up: 2  | Time: 0.000000 sec
PS C:\Users\yadav\OneDrive\Pictures\Desktop\cprog\matrix> cd "c:\Users\yadav\OneDrive\Pictures\Desktop\cprog\matrix\" ; if ($?) { gcc fi
 } ; if ($?) { .\fibo }
Enter n for n-th term of the Fibonacci series: 10

(a) Recursive: 55  | Time: 0.000000 sec
(b) Iterative: 55  | Time: 0.000000 sec
(c) DP Top-Down: 55  | Time: 0.000000 sec
(d) DP Bottom-Up: 55  | Time: 0.000000 sec
PS C:\Users\yadav\OneDrive\Pictures\Desktop\cprog\matrix>
```

PYTHON CODE:

```python
import matplotlib.pyplot as plt

# Example data (replace with your measured times from C program)
ns = [5, 10, 15, 20, 25, 30, 35]

recursive_times = [0.0000, 0.0001, 0.0006, 0.006, 0.07, 0.8, 7.5]
iterative_times = [0.0000, 0.0000, 0.0000, 0.0001, 0.0001, 0.0001, 0.0002]
topdown_times   = [0.0000, 0.0000, 0.0000, 0.0001, 0.0001, 0.0001, 0.0002]
bottomup_times  = [0.0000, 0.0000, 0.0000, 0.0001, 0.0001, 0.0001, 0.0002]

# Plotting
plt.figure(figsize=(8,6))
plt.plot(ns, recursive_times, 'r-o', label="Recursive")
plt.plot(ns, iterative_times, 'g-o', label="Iterative")
plt.plot(ns, topdown_times, 'b-o', label="DP Top-Down")
plt.plot(ns, bottomup_times, 'm-o', label="DP Bottom-Up")

plt.xlabel("n (Fibonacci index)")
plt.ylabel("Execution Time (seconds)")
plt.title("Fibonacci Algorithms: Time Complexity Comparison")
plt.legend()
plt.grid(True)
plt.show()
```

GRAPH: