**Assignment 4**

**Name: Atharva Salitri**

**Roll No.: 029**

**Branch: TY CSAI-B**

**Batch: B2**

**PRN: 12310120**

**Title:**

Assignment Based on Fibonacci Sequence using Dynamic Programming. (Determine Time and space complexity)

**Theory:**

The Fibonacci sequence is a classic problem in computer science and mathematics, often used to demonstrate the power of dynamic programming. The sequence starts with 0 and 1, and every subsequent number is the sum of the previous two. The challenge lies in computing the *n-th Fibonacci number* efficiently, since the naive recursive solution grows exponentially in time. Dynamic programming solves this problem by storing results of subproblems and reusing them, thus avoiding redundant computations.

**Theory and Explanation**

When solving Fibonacci problems:

* **Naive recursive approach** recalculates the same values multiple times, leading to exponential time complexity O(2n)O(2^n)O(2n).
* **Dynamic programming approach** improves efficiency by storing previously computed results (called memoization or tabulation).
* In tabulation (bottom-up), we start from the base cases (Fib(0) and Fib(1)) and iteratively compute up to Fib(n) using an array. This ensures each Fibonacci number is computed once.

This method drastically reduces the time complexity to O(n)O(n)O(n) and the space complexity to O(n)O(n)O(n), as extra space is used for the array storing Fibonacci values. However, with optimization, we can reduce the space complexity to O(1)O(1)O(1) by only keeping track of the last two computed values.

**Steps Involved**

* Start with base cases: Fib(0) = 0, Fib(1) = 1.
* Use a loop from 2 up to n.
* In each step, compute the current Fibonacci number as the sum of the previous two.
* Store each computed Fibonacci value in an array (for reuse).
* Return Fib(n) as the final answer.

**Pseudocode (Dynamic Programming Approach)**

text

function fibonacci(n):

if n == 0:

return 0

if n == 1:

return 1

create array fib[0...n]

fib[0] = 0

fib[1] = 1

for i from 2 to n:

fib[i] = fib[i-1] + fib[i-2]

return fib[n]

**Example (n = 6)**

* Start with base: fib = 0, fib = 1
* fib = fib + fib = 1 + 0 = 1
* fib = fib + fib = 1 + 1 = 2
* fib = fib + fib = 2 + 1 = 3
* fib = fib + fib = 3 + 2 = 5
* fib = fib + fib = 5 + 3 = 8

So, for n = 6, the Fibonacci number is 8.

-

**Code:**

import java.util.Scanner;

public class Fibonacci {

    public static int[] fibonacci(int n) {

        int[] dp = new int[n];

        if (n > 0) dp[0] = 0;

        if (n > 1) dp[1] = 1;

        for (int i = 2; i < n; i++) {

            dp[i] = dp[i - 1] + dp[i - 2];

        }

        return dp;

    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        System.out.println("Enter the number of Fibonacci terms: ");

        int n = scanner.nextInt();

        int[] result = fibonacci(n);

        System.out.println("Fibonacci sequence:");

        for (int num : result) {

            System.out.println(num);

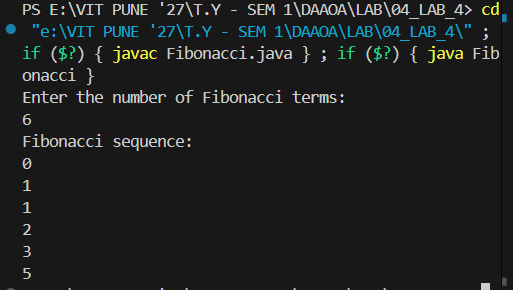
        }

        scanner.close();

    }

}

**OUTPUT:**

****

**Time and Space Complexity Analysis:**

**Time Complexity: O(n)**

* The loop runs from 2 to n, so it makes about n steps in total.
* Each step only does a simple addition, which takes constant time.
* As n grows, the time taken grows directly in proportion to n.

**Space Complexity: O(n)**

* An array is created to store all Fibonacci numbers up to n.
* This means memory usage increases as n increases.
* Storing results avoids recalculating Fibonacci numbers again, saving time but using more memory.

FUNCTION fibonacci(n)

DECLARE array dp of size n // Space: +n for array storage

IF n > 0 // Time: +1

dp[0] ← 0 // Time: +1

IF n > 1 // Time: +1

dp[1] ← 1 // Time: +1

FOR i FROM 2 TO n-1 // Time: + (n-2) ≈ +n

dp[i] ← dp[i-1] + dp[i-2] // Time: +1 per iteration (addition & assignment)

RETURN dp // Time: +1 (return array reference)

ENDFUNCTION

FUNCTION main

DECLARE scanner // Space: +1 for scanner object

PRINT "Enter the number of Fibonacci terms:" // Time: +1

INPUT n // Time: +1

result ← fibonacci(n) // Time: O(n), Space: O(n)

PRINT "Fibonacci sequence:" // Time: +1

FOR each num IN result // Time: +n

PRINT num // Time: +1 per iteration

CLOSE scanner // Time: +1

ENDFUNCTION

**Conclusion**

In this lab exercise, we learned how to create the Fibonacci sequence using Dynamic Programming.