|  |  |
| --- | --- |
|  | Department of Computer Science and Engineering  Chandpur Science and Technology University |

**LAB-06**

**Course Title**: Algorithm Design and Analysis Sessional

**Course Code**:CSE 2202

**Submitted To-**

**Mostafiz Ahammed**

*Lecturer****Department of Computer Science and Engineering****Chandpur Science and Technology University*

**Submitted By-**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sl** | Class Roll | Name | Batch | Signature |
| 01 | B210101022 | Md. Rabbi Ahammed | CSE-21 | rabbi |

**Date of Submission: July, 2025**

**Experiment 01: Implementation of Fibonacci Series using Recursion and Dynamic Programming.**

**Objective**

To implement and compare the Fibonacci series using recursion and dynamicprogramming techniques in C++, focusing on their time and space complexities.

### ****Algorithm****

#### a) Recursive Approach:

* If n <= 1, return n.
* Else, return fib(n - 1) + fib(n - 2).

#### b) Dynamic Programming (Bottom-Up):

* Create an array dp[n+1].
* Initialize: dp[0] = 0, dp[1] = 1.
* Loop from 2 to n, compute dp[i] = dp[i - 1] + dp[i - 2].
* Return dp[n].

### *****Theoretical Solution*****

The Fibonacci series is a sequence of numbers where each term is the sum of the two preceding ones:

F(0) = 0

F(1) = 1

F(n) = F(n-1) + F(n-2) for n ≥ 2

Recursive solutions are elegant but inefficient. Dynamic programming avoids redundant calculations and improves performance.

### *****Practical Work*****

#### a) ****Pseudocode****

**Recursive:**

function fib(n):

if n <= 1:

return n

else:

return fib(n-1) + fib(n-2)

**Dynamic Programming:**

function fib\_dp(n):

initialize dp[0] = 0, dp[1] = 1

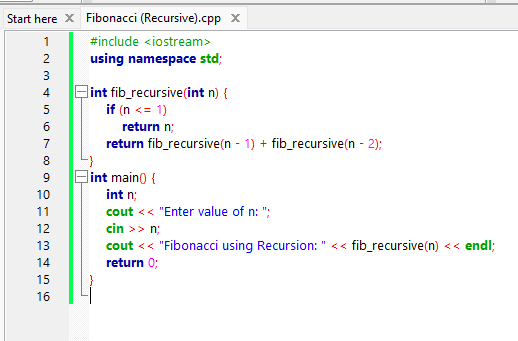
for i = 2 to n:

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

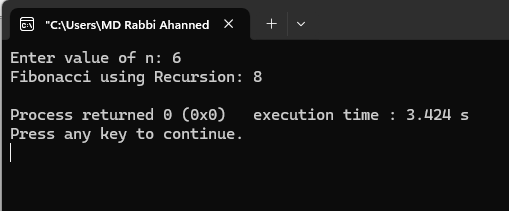
return dp[n]

*b)* ***Source Code in C++***

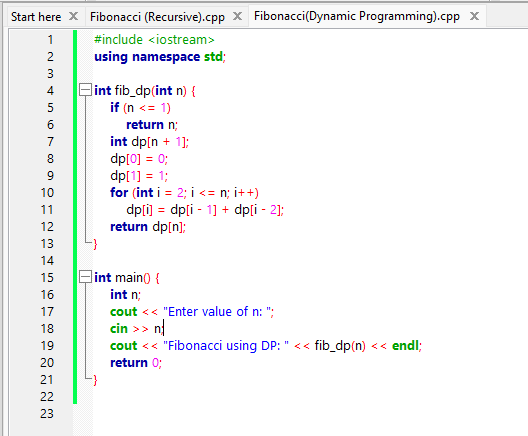
**Recursive way:**



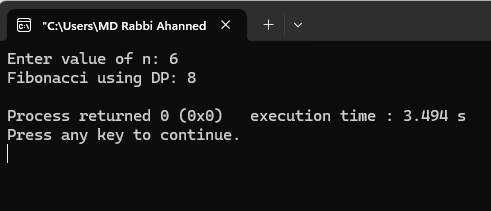
**Output:**



**Dynamic Programming:**

****

**Output:**

****

## Analysis Table

| **Algorithm** | **Best Case** | **Worst Case** | **Avg Case** | **Space Complexity** |
| --- | --- | --- | --- | --- |
| Recursive | O(1) | O(2^n) | O(2^n) | O(n) (stack) |
| Dynamic Programming | O(n) | O(n) | O(n) | O(n) |

## Observations

- Recursive method is elegant but inefficient for large n.  
- Dynamic Programming performs better by eliminating redundant calls.  
- For very large n, recursion leads to stack overflow or long delays.

## Challenges

- Recursive function crashes for n > 45 due to stack overflow.  
- Students often forget to initialize base cases in DP.  
- Understanding how memoization saves time requires a deeper understanding of recursion trees.

## Conclusion

This experiment shows how dynamic programming is significantly more efficient than recursion for computing Fibonacci numbers. It emphasizes the need for optimizing recursive algorithms to handle large inputs effectively.