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

**LAB-07**

**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: 19 July, 2025**

**Experiment Name : Coin Row Problem Using Recursion and Dynamic Programming**

# Objective

To implement and compare the Coin Row Problem using Recursive and Dynamic Programming techniques, analyze their efficiency, and understand the trade-offs between time and space complexity.

# Algorithm

Problem Definition:

Given a row of coins, each with a value. You cannot pick two adjacent coins. Find the maximum value you can collect.

Mathematical Formulation:

Let:  
- C[i] be the value of the i-th coin.  
- F[i] be the maximum value that can be obtained from the first i coins.  
Then,  
- F[0] = 0  
- F[1] = C[1]  
- F[i] = max(C[i] + F[i-2], F[i-1]) for i >= 2

# Theoretical Solution

Recursive Approach:

- Simple but inefficient for large inputs due to overlapping subproblems.  
- Time Complexity: O(2^n) (Exponential)

Dynamic Programming Approach:

- Solves each subproblem only once and stores the result.  
- Time Complexity: O(n), Space Complexity: O(n)

# Practical Work

## a. Pseudocode

Recursive Solution:

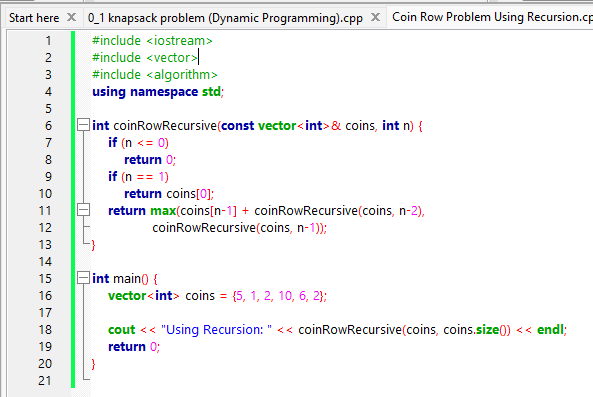
function coinRowRecursive(arr, n):  
 if n <= 0:  
 return 0  
 if n == 1:  
 return arr[0]  
 return max(arr[n-1] + coinRowRecursive(arr, n-2),  
 coinRowRecursive(arr, n-1))

Dynamic Programming Solution:

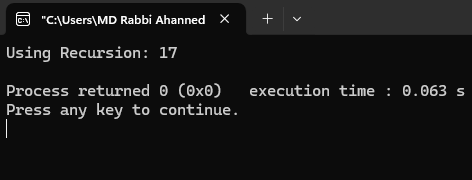
function coinRowDP(arr, n):  
 create dp[n+1]  
 dp[0] = 0  
 dp[1] = arr[0]  
 for i from 2 to n:  
 dp[i] = max(arr[i-1] + dp[i-2], dp[i-1])  
 return dp[n]

## b. Source Code in C++

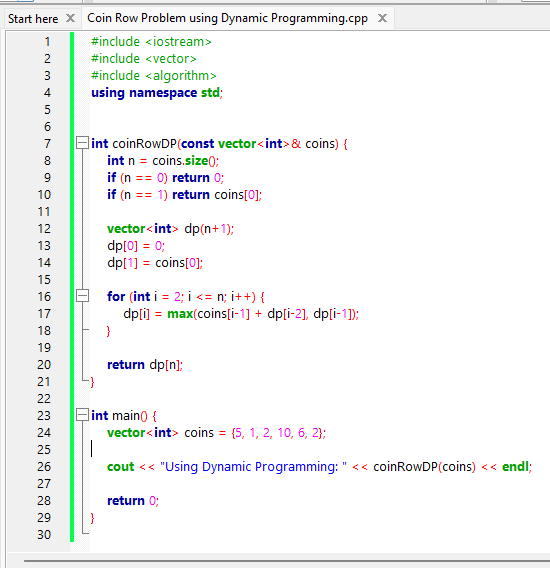
**Recursive way**



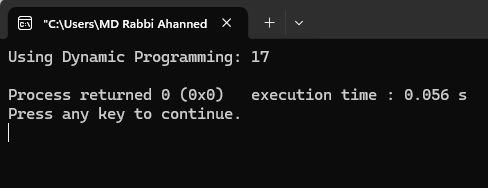
**Output:**

****

**Dynamic Programming**

****

**Output:**

****

# Analysis Table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Method | Best Case | Worst Case | Avg Case | Space |
| Recursive | O(1) (n = 0) | O(2ⁿ) | O(2ⁿ) | O(n) (stack) |
| Dynamic Programming | O(n) | O(n) | O(n) | O(n) |

# Observations

- Recursive solution becomes impractical for large n due to repeated calculations.  
- Dynamic Programming is significantly faster and uses a bottom-up approach to avoid redundant work.  
- DP maintains optimal substructure and overlapping subproblems, ideal for memoization.

# Challenges

- Understanding how subproblems overlap in the recursive solution.  
- Deciding when to use recursion vs dynamic programming.  
- Properly indexing arrays to avoid off-by-one errors.

# Conclusion

The Coin Row problem illustrates the power of Dynamic Programming over brute-force recursive approaches. By storing intermediate results, DP achieves a dramatic improvement in time complexity from exponential to linear, making it suitable for real-world large input sizes.