Experiment - 7

Name: Yuvraj Tripathi UID: 22BET10140

Branch: IT Section: IOT 703/B

Semester: 6th Date of Performance: 18/03/25

Subject: AP LAB - 2 Subject Code: 22ITP-351

1. Aim:

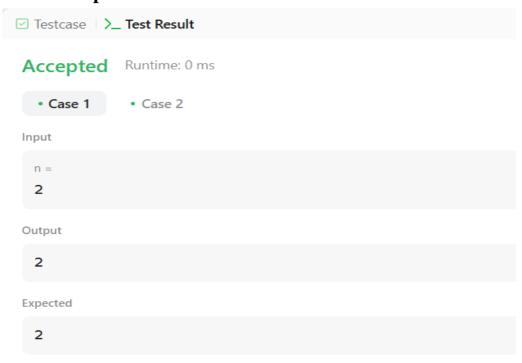
Problem 1.1.1: Climbing Stairs

Problem Statement: Climbing Stairs Find the number of ways to climb n stairs when you can take 1 or 2 steps at a time. This follows the Fibonacci sequence and can be solved using dynamic programming (DP) in O(n) time and O(1) space. Edge cases include n = 1 and n = 2.

2. Objectives: Find the number of ways to climb n stairs using 1 or 2 steps at a time. Solve efficiently using DP in O(n) time and O(1) space, following the Fibonacci sequence while handling edge cases like n = 1 and n = 2.

```
class Solution {
public:
  int solve(int n,vector<int> &dp){
     //base case
     if(n \le 2)
      return n;
     if(dp[n]!=-1)
      return dp[n];
     dp[n]=solve(n-1,dp)+solve(n-2,dp);
     return dp[n];
  int climbStairs(int n) {
     if(n \le 2)
     return n;
     vector<int> dp(n+1,-1);
     return solve(n,dp);
};
```

4. Output:



Problem 1.1.2: Maximum Subarray

Problem Statement: - Maximum Subarray Find the contiguous subarray with the maximum sum using Kadane's Algorithm. This runs in O(n) time and O(1) space. Edge cases include an array of all negative numbers.

```
class Solution {
public:
    int maxSubArray(vector<int>& nums) {
        int res = nums[0];
        int total = 0;

        for (int n : nums) {
            if (total < 0) {
                total = 0;
            }

            total += n;
            res = max(res, total);
        }
}</pre>
```

```
return res;
}
};
```

✓ Testcase \>_ Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

nums = [-2,1,-3,4,-1,2,1,-5,4]

Output

6

Expected

6

1. Problem 1.1.3: House Robber

Problem Statement: You are given an array representing the amount of money in each house on a street. You cannot rob two adjacent houses. Determine the maximum amount you can rob without alerting the police.

2. Implementation of Code:

```
class Solution {
  public:
    int rob(vector<int>& nums) {
       int prevRob = 0;
      int maxRob = 0;

      for (int curValue : nums) {
         int temp = max(maxRob, prevRob + curValue);
         prevRob = maxRob;
         maxRob = temp;
      }

      return maxRob;
    }
}
```

3. Output:

```
Testcase > Test Result

Accepted Runtime: 0 ms

• Case 1
• Case 2

Input

nums = [1,2,3,1]

Output

4

Expected

4
```

Problem 1.1.4: Jump Game

Problem Statement: Given an array where each element represents the maximum jump length from that position, determine if you can reach the last index starting from the first index

Implementation of code:

```
class Solution {
  public:
    bool canJump(vector<int>& nums) {
      int goal = nums.size() - 1;

      for (int i = nums.size() - 2; i >= 0; i--) {
        if (i + nums[i] >= goal) {
            goal = i;
        }
    }

    return goal == 0;
}
```

Output:

```
Testcase > Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

nums = [2,3,1,1,4]

Output

true

Expected

true
```

Problem 1.1.5: Unique Paths

Problem Statement: You are given an m x n grid where a robot starts at the top-left and can only move right or down. Determine the number of unique paths the robot can take to reach the bottom-right corner.

Implementation of code:

```
class Solution {
public:
    int uniquePaths(int m, int n) {
        std::vector<int> aboveRow(n, 1);

        for (int row = 1; row < m; row++) {
            std::vector<int> currentRow(n, 1);
            for (int col = 1; col < n; col++) {
                currentRow[col] = currentRow[col - 1] + aboveRow[col];
            }
            aboveRow = currentRow;
        }

        return aboveRow[n - 1];
    }
};</pre>
```

Output:

```
Testcase > Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

m = 3

n = 7

Output

28

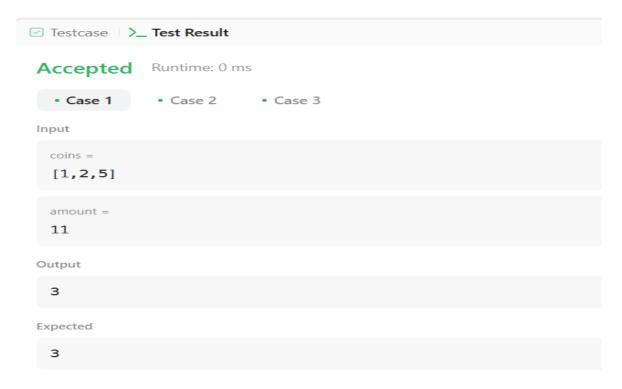
Expected

28
```

Problem 1.1.6: Coin Change

Problem Statement: Given an array of coin denominations and an amount, determine the minimum number of coins needed to make up that amount. If it is impossible, return -1.

```
class Solution {
public:
  int minCoin(vector<int> &coins, int amount) {
    if (amount <= 0) {
       return 0;
     }
    int ans = INT_MAX;
    for (int i = 0; i < coins.size(); i++) {
       int coin = coins[i];
       if (coin <= amount) {
          int recAns = minCoin(coins, amount - coin);
          if (recAns != INT_MAX) {
            recAns += 1;
            ans = min(ans, recAns);
          }
     }
    return ans;
  }
  int coinChange(vector<int>& coins, int amount) {
    int ans = minCoin(coins, amount);
    if (ans == INT\_MAX) {
       return -1;
    }
    return ans;
};
```



Problem 1.1.7: Longest Increasing Subsequence

Problem Statement: Given an array of integers, find the length of the longest subsequence where the elements are strictly increasing.

```
class Solution {
public:
    int LISRec(vector<int> &nums, int prev, int curr) {
        if (curr >= nums.size()) {
            return 0;
        }

    int include = 0;
    if (prev == -1 || nums[prev] < nums[curr]) {
            include = 1 + LISRec(nums, curr, curr + 1);
        }
        int exclude = LISRec(nums, prev, curr + 1);
        return max(include, exclude);
    }
}</pre>
```

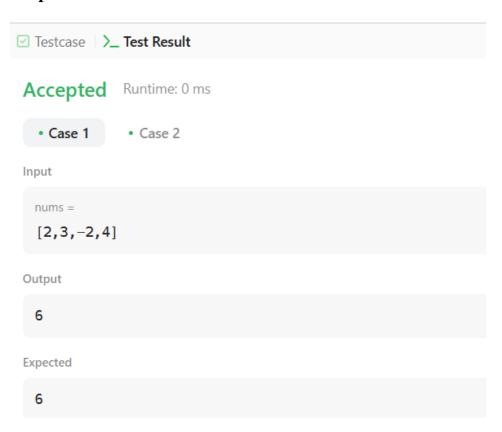
```
int lengthOfLIS(vector<int>& nums) {
  int prev = -1;
  int curr = 0;
  return LISRec(nums, prev, curr);
}
};
```

Problem 1.1.8: Maximum Product Subarray

Problem Statement: Given an array of integers, find the contiguous subarray (of at least one element) that has the largest product and return its product.

```
class Solution {
public:
  int maxProduct(vector<int>& nums) {
    int maxi = INT_MIN;
    int prod=1;
    for(int i=0;i<nums.size();i++)
     prod*=nums[i];
     maxi=max(prod,maxi);
     if(prod==0)
      prod=1;
     }
    prod=1;
    for(int i=nums.size()-1;i>=0;i--)
     prod*=nums[i];
     maxi=max(prod,maxi);
     if(prod==0)
      prod=1;
    return maxi;
};
```





Learning Outcomes:

- Understand how to solve problems using dynamic programming by breaking them into smaller subproblems.
- Learn to optimize time and space complexity using DP techniques.
- Recognize how the Fibonacci sequence applies to real-world problems like climbing stairs.
- Understand how to handle constraints (e.g., not robbing adjacent houses) while maximizing a value.
- Learn to solve problems like the Jump Game using greedy approaches to determine reachability.