Discover. Learn. Empower. ENGINEERING

WORKSHEET-7

Student Name: Nishant thakur **UID:** 22BCS15878

Branch: CSE **Section/Group:** NTPP-603-B

Semester: 6th **Date of Performance:** 13/3/25

Subject Name: AP-2 **Subject Code:** 22CSP-351

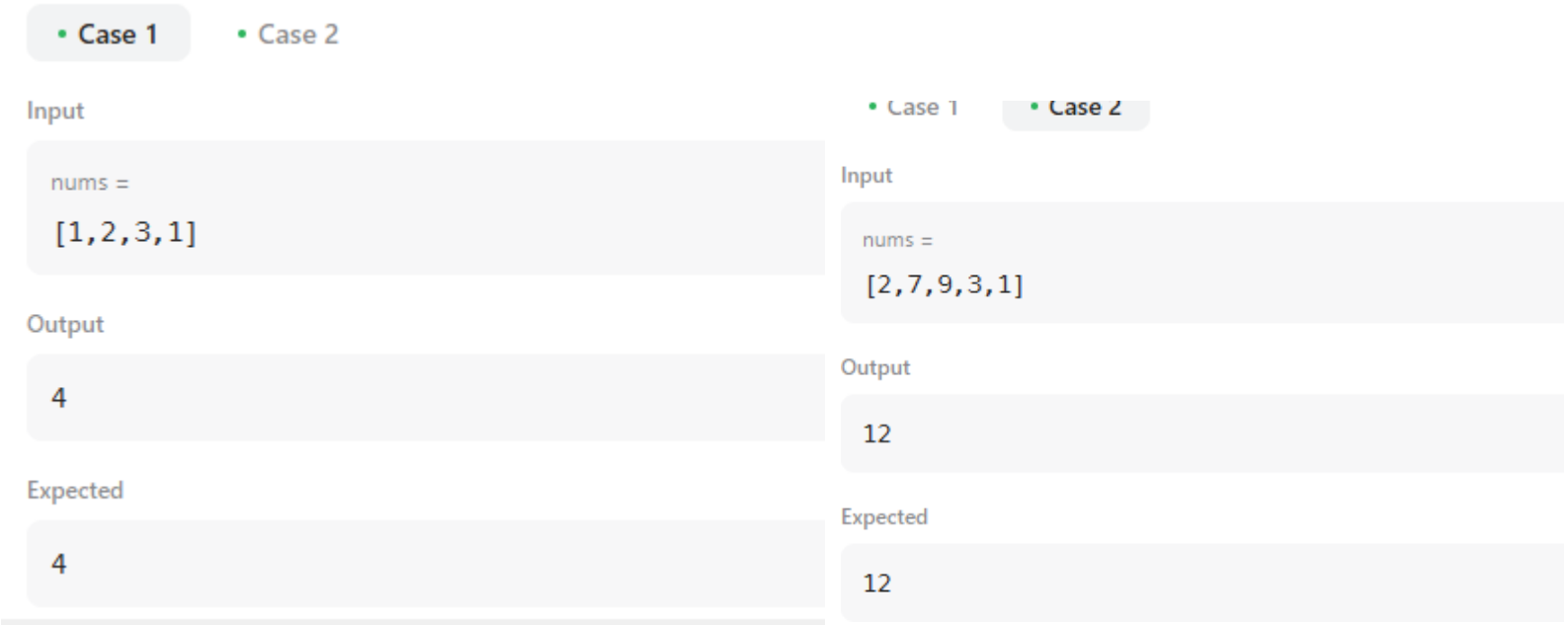
Aim(i): 198. You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent houses have security systems connected and it will automatically contact the police if two adjacent houses were broken into on the same night. Given an integer array nums representing the amount of money of each house, return the maximum amount of money you can rob tonight without alerting the police.

Source Code:

```
class Solution {
public:
    int rob(vector<int>& nums) {
        int n = nums.size();

        if (n == 1) {
            return nums[0];
        }
     vector<int> dp(n, 0);

        dp[0] = nums[0];
        dp[1] = max(nums[0], nums[1]);
        for (int i = 2; i < n; i++) {
            dp[i] = max(dp[i - 1], nums[i] + dp[i - 2]);
        }
        return dp[n - 1];
    }
```

```
};
```

OUTPUT:



LEARNING OUTCOME:

1. We learnt about Dynamic Programming.
2. We learnt how to use Vector class.
3. We learnt how to rob houses.

Aim(ii): 62. There is a robot on an m x n grid. The robot is initially located at the top-left corner (i.e., grid[0][0]). The robot tries to move to the bottom-right corner (i.e., grid[m - 1][n - 1]). The robot can only move either down or right at any point in time.

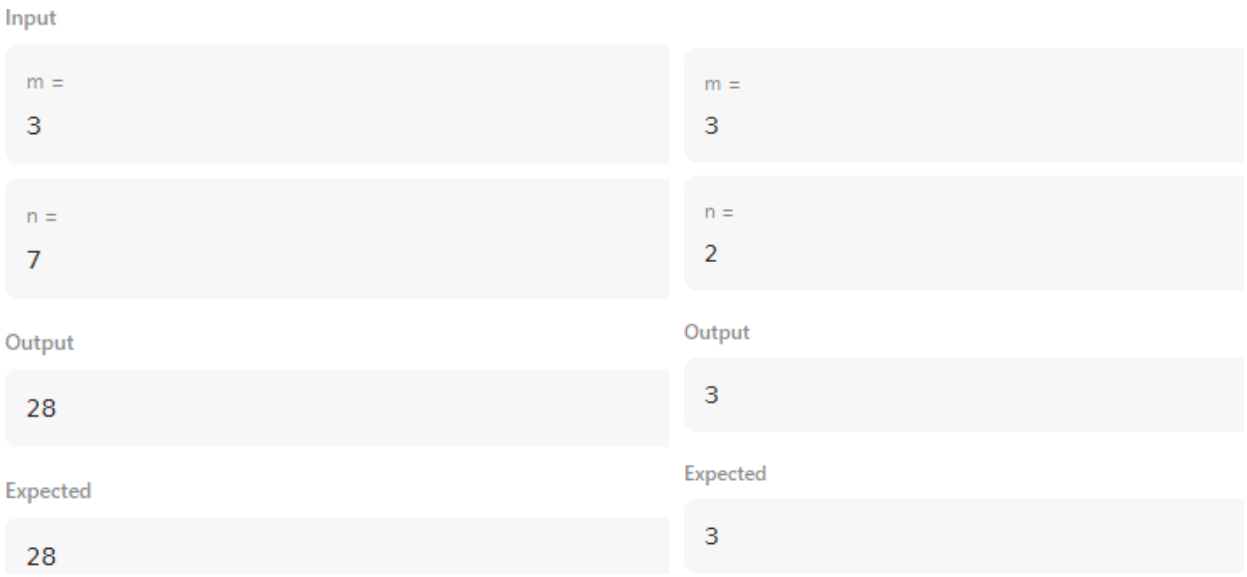
Given the two integers m and n, return the number of possible unique paths that the robot can take to reach the bottom-right corner.

The test cases are generated so that the answer will be less than or equal to 2 * 109.

Source 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];
    }
};
```

OUTPUT:



Learning Outcomes

1. We learnt how to use Grids.
2. We learnt how to use current row.

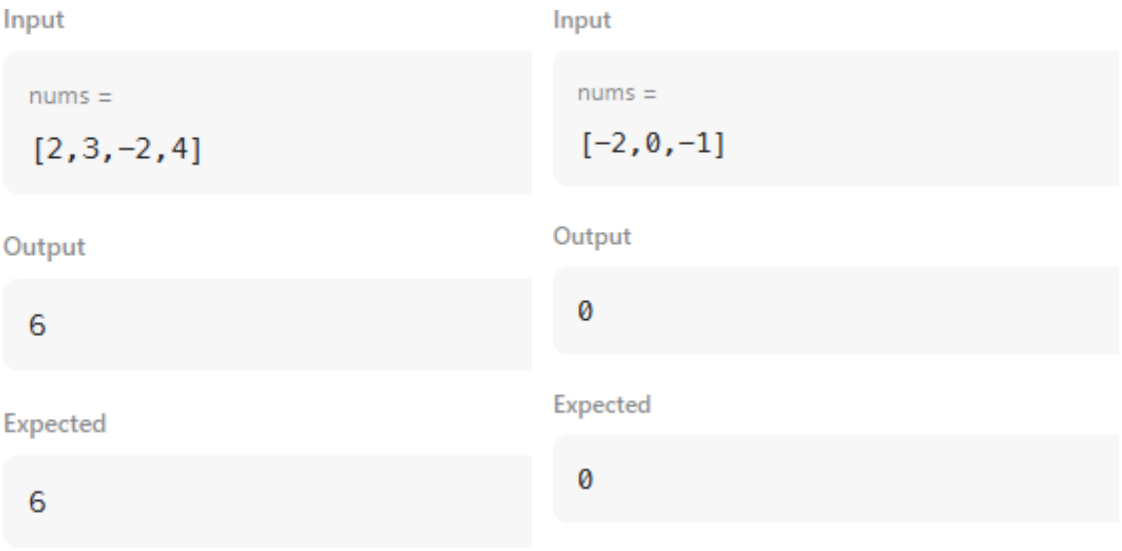
Aim(iii): 152**.** Given an integer array nums, find a subarray that has the largest product, and return the product.

The test cases are generated so that the answer will fit in a 32-bit integer.

Source Code:

```
class Solution {
public:
    int maxProduct(vector<int>& nums) {
        int res = *max_element(nums.begin(), nums.end());
        int curMax = 1, curMin = 1;

        for (int n : nums) {
            int temp = curMax * n;
            curMax = max({temp, curMin * n, n});
            curMin = min({temp, curMin * n, n});

            res = max(res, curMax);
        }

        return res;
    }
};


};



};
```

OUTPUT:



Learning Outcomes

1. We learnt about temp.
2. We learnt usage of Max_element.