

## WORKSHEET-7

**StudentName:**Mayank **UID:** 22BCS10612

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

**Semester:**6th **DateofPerformance:**16/3/25

**SubjectName:AP-2 SubjectCode:**22CSP-351

### Aim(i):

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 isthatadjacenthouseshavesecuritysystemsconnectedanditwill 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.

### SourceCode:

classSolution{ public:

introb(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. WelearntaboutDynamicProgramming.
2. WelearnthowtouseVectorclass.
3. Welearnthowtorobhouses.

**Aim(ii)**: 62. There is a robot on an m x n grid.Therobotisinitiallylocatedatthe top-left corner (i.e., grid[0][0]). The robottriestomovetothebottom-rightcorner (i.e., grid[m-1][n-1]).Therobotcanonlymoveeitherdownorrightatanypoint 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.

### SourceCode:

classSolution{ 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(intcol=1;col<n;col++){

currentRow[col]= currentRow[col - 1] + aboveRow[col];

}

aboveRow = currentRow;

}

return aboveRow[n - 1];

}

};

## OUTPUT:

****

### LearningOutcomes

1. WelearnthowtouseGrids.
2. Welearnthowtousecurrentrow.

**Aim(iii):**152**.**Givenanintegerarraynums,findasubarraythathasthelargest product, and return the product.

Thetestcasesaregeneratedsothattheanswerwillfitina32-bitinteger.

### SourceCode:

classSolution{ public:

intmaxProduct(vector<int>&nums) {

intres=\*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;

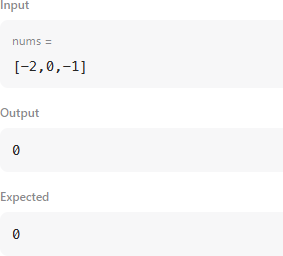
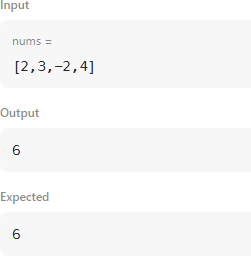
}

};

};

};

**OUTPUT:**

****

# Learning Outcomes

1. Welearntabouttemp.
2. WelearntusageofMax\_element.