1 ----- Leetcode 70: Climbing Stairs

You are climbing a staircase. It takes n steps to reach the top.

Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

Example 1:

Input: n = 2

Output: 2

Explanation: There are two ways to climb to the top.

1. 1 step + 1 step

2. 2 steps

Example 2:

Input: n = 3

Output: 3

Explanation: There are three ways to climb to the top.

1. 1 step + 1 step + 1 step

2. 1 step + 2 steps

3. 2 steps + 1 step

Constraints:

1 <= n <= 45

2 ----- GFG: Geek Jump

Geek wants to climb from the 0th stair to the (n-1)th stair. At a time the Geek can climb either one or two steps. A height[N] array is also given. Whenever the geek jumps from stair i to stair j, the energy consumed in the jump is abs(height[i]- height[j]), where abs() means the absolute difference. return the minimum energy that can be used by the Geek to jump from stair 0 to stair N-1.

**Example:**

**Input:**

n = 4

height = {10 20 30 10}

**Output:**

20

**Explanation:**

Geek jump from 1st to 2nd stair(|20-10| = 10 energy lost).

Then a jump from the 2nd to the last stair(|10-20| = 10 energy lost).

so, total energy lost is 20 which is the minimum.

**Your Task:**  
You don't need to read input or print anything. Your task is to complete the function **MinimumEnergy()**which takes the array **height**, and integer **n,** and returns the minimum energy that is lost.

**Expected Time Complexity:** O(n)  
**Expected Space Complexity:** O(n)

**Constraint:**  
1<=n<=100000  
1<=height[i]<=1000

### 3 ----- GFG: Minimal Cost

There is an array **arr** of heights of stone and Geek is standing at the first stone and can jump to one of the following: Stone i+1, i+2, ... i+k stone, where k is the maximum number of steps that can be jumped and cost will be |hi-hj| is incurred, where j is the stone to land on. Find the minimum possible total cost incurred before the Geek reaches the last stone.

**Example:**

**Input:** k = 3, arr[]= [10, 30, 40, 50, 20]  
**Output:** 30  
**Explanation:** Geek will follow the path 1->2->5, the total cost would be | 10-30| + |30-20| = 30, which is minimum

**Input:** k = 1, arr[]= [10, 20, 10]

**Output:** 20

**Explanation:** Geek will follow the path 1->2->3, the total cost would be |10 - 20| + |20 - 10| = 20.

**Expected Time Complexity**: O(n\*k)  
**Expected Auxilary Space**: O(n)

**Constraint:**1<= arr.size() <=104  
1 <= k <= 100  
1 <= arr[i] <= 104

### 4 ----- LeetCode 198: House Robber

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***.

**Example 1:**

**Input:** nums = [1,2,3,1]

**Output:** 4

**Explanation:** Rob house 1 (money = 1) and then rob house 3 (money = 3).

Total amount you can rob = 1 + 3 = 4.

**Example 2:**

**Input:** nums = [2,7,9,3,1]

**Output:** 12

**Explanation:** Rob house 1 (money = 2), rob house 3 (money = 9) and rob house 5 (money = 1).

Total amount you can rob = 2 + 9 + 1 = 12.

**Constraints:**

* 1 <= nums.length <= 100
* 0 <= nums[i] <= 400

5 ----- Leetcode 213: House Robber II

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed. All houses at this place are **arranged in a circle.** That means the first house is the neighbor of the last one. Meanwhile, adjacent houses have a security system 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***.

**Example 1:**

**Input:** nums = [2,3,2]

**Output:** 3

**Explanation:** You cannot rob house 1 (money = 2) and then rob house 3 (money = 2), because they are adjacent houses.

**Example 2:**

**Input:** nums = [1,2,3,1]

**Output:** 4

**Explanation:** Rob house 1 (money = 1) and then rob house 3 (money = 3).

Total amount you can rob = 1 + 3 = 4.

**Example 3:**

**Input:** nums = [1,2,3]

**Output:** 3

**Constraints:**

* 1 <= nums.length <= 100
* 0 <= nums[i] <= 1000

6 ----- GFG: Geek’s Training

Geek is going for a training program. He can perform any of these activities: Running, Fighting, and Learning Practice. Each activity has some point on each day. As Geek wants to improve all his skills, he can't do the same activity on two consecutive days. Help Geek to maximize his merit points as you are given a 2D array of points **arr,** corresponding to each day and activity.

**Example:**

**Input:** n=3 and arr[]= [[1,2,5],[3,1,1],[3,3,3]]

**Output:**11

**Explanation:**Geek will learn a new move and earn 5 point then on second day he will do running and earn 3 point and on third day he will do fighting and earn 3 points so, maximum point is 11.

**Expected Time Complexity:** O(3\*n)  
**Expected Space Complexity:** O(3\*n)

**Constraint:**  
1 <=  arr.size <= 105  
1 <=  arr[i][j] <= 100

7 ----- Leetcode 62: Unique Paths

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.



**Input:** m = 3, n = 7

**Output:** 28

**Example 2:**

**Input:** m = 3, n = 2

**Output:** 3

**Explanation:** From the top-left corner, there are a total of 3 ways to reach the bottom-right corner:

1. Right -> Down -> Down

2. Down -> Down -> Right

3. Down -> Right -> Down

**Constraints:**

* 1 <= m, n <= 100

8 ----- Leetcode 63: Unique Paths II

You are given an m x n integer array grid. There is a robot 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.

An obstacle and space are marked as 1 or 0 respectively in grid. A path that the robot takes cannot include **any** square that is an obstacle.

Return *the number of possible unique paths that the robot can take to reach the bottom-right corner*.

The testcases are generated so that the answer will be less than or equal to 2 \* 109



**Input:** obstacleGrid = [[0,0,0],[0,1,0],[0,0,0]]

**Output:** 2

**Explanation:** There is one obstacle in the middle of the 3x3 grid above.

There are two ways to reach the bottom-right corner:

1. Right -> Right -> Down -> Down

2. Down -> Down -> Right -> Right



**Input:** obstacleGrid = [[0,1],[0,0]]

**Output:** 1

**Constraints:**

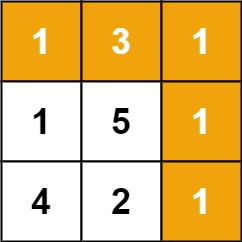
* m == obstacleGrid.length
* n == obstacleGrid[i].length
* 1 <= m, n <= 100
* obstacleGrid[i][j] is 0 or 1.

9 ----- Leetcode 64: Minimum Path Sum

Given a m x n grid filled with non-negative numbers, find a path from top left to bottom right, which minimizes the sum of all numbers along its path.

**Note:** You can only move either down or right at any point in time.

**Example 1:**



**Input:** grid = [[1,3,1],[1,5,1],[4,2,1]]

**Output:** 7

**Explanation:** Because the path 1 → 3 → 1 → 1 → 1 minimizes the sum.

**Example 2:**

**Input:** grid = [[1,2,3],[4,5,6]]

**Output:** 12

**Constraints:**

* m == grid.length
* n == grid[i].length
* 1 <= m, n <= 200
* 0 <= grid[i][j] <= 200

10 ----- Leetcode 120: Triangle

Given a triangle array, return *the minimum path sum from top to bottom*.

For each step, you may move to an adjacent number of the row below. More formally, if you are on index i on the current row, you may move to either index i or index i + 1 on the next row.

**Example 1:**

**Input:** triangle = [[2],[3,4],[6,5,7],[4,1,8,3]]

**Output:** 11

**Explanation:** The triangle looks like:

2

3 4

6 5 7

4 1 8 3

The minimum path sum from top to bottom is 2 + 3 + 5 + 1 = 11 (underlined above).

**Example 2:**

**Input:** triangle = [[-10]]

**Output:** -10

**Constraints:**

* 1 <= triangle.length <= 200
* triangle[0].length == 1
* triangle[i].length == triangle[i - 1].length + 1
* -104 <= triangle[i][j] <= 104

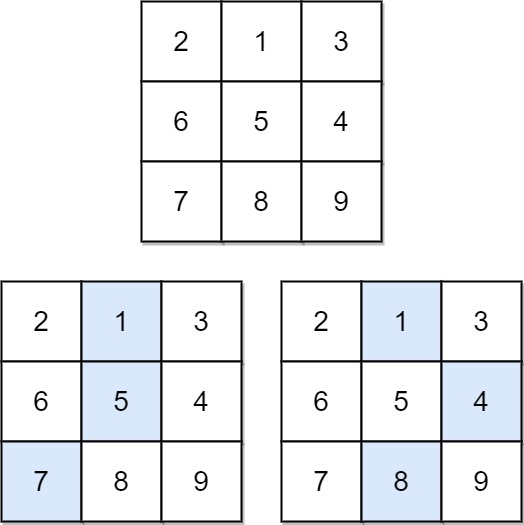
**Follow up:** Could you do this using only O(n) extra space, where n is the total number of rows in the triangle?

11 ----- Leetcode 931: Minimum Falling Path Sum

Given an n x n array of integers matrix, return *the* ***minimum sum*** *of any* ***falling path*** *through* matrix.

A **falling path** starts at any element in the first row and chooses the element in the next row that is either directly below or diagonally left/right. Specifically, the next element from position (row, col) will be (row + 1, col - 1), (row + 1, col), or (row + 1, col + 1)

**Example 1:**

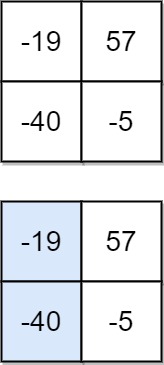


**Input:** matrix = [[2,1,3],[6,5,4],[7,8,9]]

**Output:** 13

**Explanation:** There are two falling paths with a minimum sum as shown.

**Example 2:**



**Input:** matrix = [[-19,57],[-40,-5]]

**Output:** -59

**Explanation:** The falling path with a minimum sum is shown.

**Constraints:**

* n == m
* matrix.length == matrix[i].length
* 1 <= n <= 100
* -100 <= matrix[i][j] <= 100

12 ----- GFG: Chocolates Pickup

You are given an **n** rowsand **m** cols matrix **grid** representing a field of chocolates where grid[i][j] represents the number of chocolates that you can collect from the (i, j) cell.

You have two robots that can collect chocolates for you:

* **Robot #1** is located at the **top-left corner** (0, 0), and
* **Robot #2** is located at the **top-right corner** (0, cols - 1).

Return the maximum number of chocolates collection using both robots by following the rules below:

* From a cell (i, j), robots can move to cell (i + 1, j - 1), (i + 1, j), or (i + 1, j + 1).
* When any robot passes through a cell, It picks up all chocolates, and the cell becomes an empty cell.
* When both robots stay in the same cell, only one takes the chocolates.
* Both robots cannot move outside of the grid at any moment.
* Both robots should reach the bottom row in grid.

**Example:**

**Input:**

n = 4, m = 3

grid = [[3,1,1],[2,5,1],[1,5,5],[2,1,1]]

**Output:**

24

**Explanation:**

Path of robot #1 and #2 are described in color green and blue respectively. Chocolates taken by Robot #1, (3 + 2 + 5 + 2) = 12. Chocolates taken by Robot #2, (1 + 5 + 5 + 1) = 12. Total of Chocolates: 12 + 12 = 24.

**Expected Time Complexity:** O(n \* m \* m)  
**Expected Space Complexity:** O(n \* m \* m)

**Constraint:**  
2 <= n, m < = 70  
0 <= grid[i][j] <= 100