Linear Dynamic Programming:

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 = [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.

Grid Dynamic Programming

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.

Example 1:
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

0/1 Knapsack

Given N items where each item has some weight and profit associated with it and also given a bag with capacity W, [i.e., the bag can hold at most W weight in it]. The task is to put the items into the bag such that the sum of profits associated with them is the maximum possible. (Note: The constraint here is we can either put an item completely into the bag or cannot put it at all. It is not possible to put a part of an item into the bag).

Examples:
Input: N = 3, W = 4, profit[] = {1, 2, 3}, weight[] = {4, 5, 1}
Output: 3

Explanation: There are two items which have weight less than or equal to 4. If we select the item with weight 4, the possible profit is 1. And if we select the item with weight 1, the possible profit is 3. So the maximum possible profit is 3. Note that we cannot put both the items with weight 4 and 1 together as the capacity of the bag is 4.

Knapsack Unbounded

Given a knapsack weight W and a set of n items with certain value val_i and weight wt_i, we need to calculate the maximum amount that could make up this quantity exactly. Allowed to use unlimited number of instances of an item. (Note: N is always positive i.e greater than 0)

Example 1:

Input : W = 100
val[] = {1, 30}
wt[] = {1, 50}

Output : 100
There are many ways to fill knapsack: “2 instances of 50 unit weight item”, “100 instances of 1 unit weight item” or “1 instance of 50 unit weight item and 50 instances of 1 unit weight items”.Maximum value with option 2.

Longest Common Subsequence

Given two strings text1 and text2, return the length of their longest common subsequence. If there is no common subsequence, return 0. A subsequence of a string is a new string generated from the original string with some characters (can be none) deleted without changing the relative order of the remaining characters. For example, "ace" is a subsequence of "abcde".

Example 1:

Input: text1 = "abcde", text2 = "ace"
Output: 3
Explanation: The longest common subsequence is "ace" and its length is 3.

Example 2:

Input: text1 = "abc", text2 = "def"
Output: 0
Explanation: There is no such common subsequence, so the result is 0.