Checklist

1. Memoization & Tabulation
2. 1D DP Problems
3. 2D DP Grid Problems
4. Knapsack & Unbounded Knapsack Problems
5. DP on Strings
6. MCM problems and DP on Trees

What is Dynamic Programming?

misleading name

Advanced / Optimised Recursion

Prerequisite:

1) Recursion → Basics, Understanding, multiple calls
2) Arrays, 2D arrays → Basics
3) Hashmaps & Trees → Basics

$O(\log n) > O(n) > O(n \log n) > O(n^2) > O(n^2 \log n) > O(n^3) > O(2^n)$

509. Fibonacci Number

Solved

Easy | Topics | Companies

The **Fibonacci numbers**, commonly denoted `F(n)` form a sequence, called the **Fibonacci sequence**, such that each number is the sum of the two preceding ones, starting from `0` and `1`. That is,

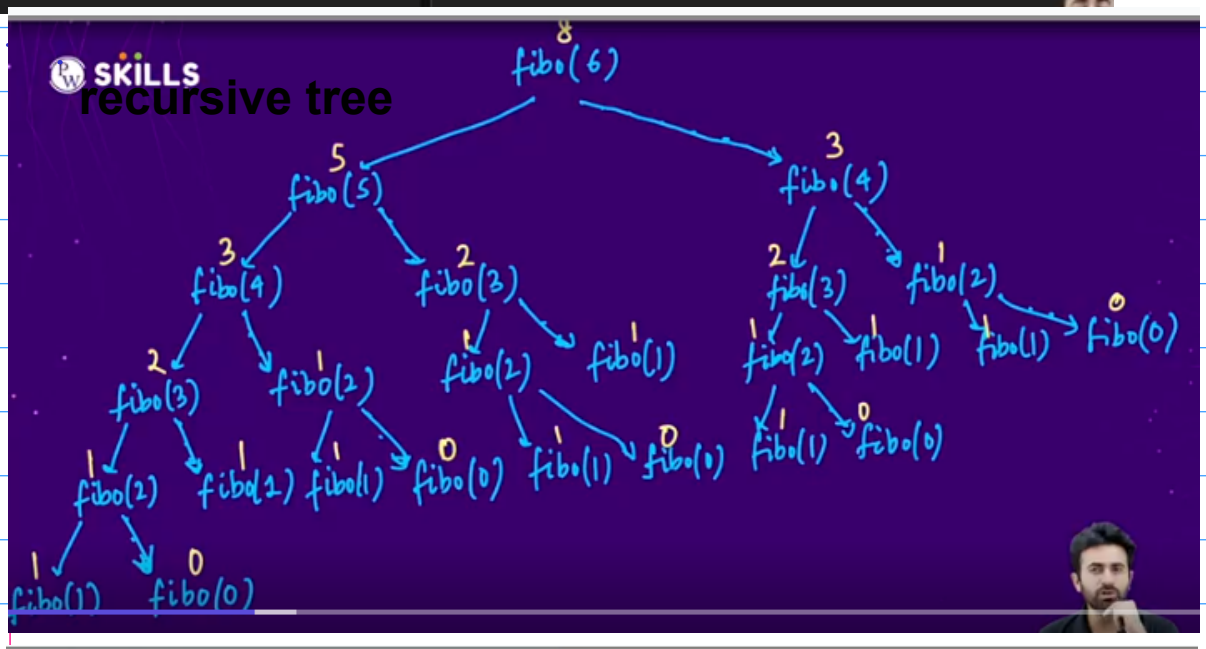
```
F(0) = 0, F(1) = 1
F(n) = F(n - 1) + F(n - 2), for n > 1.
```

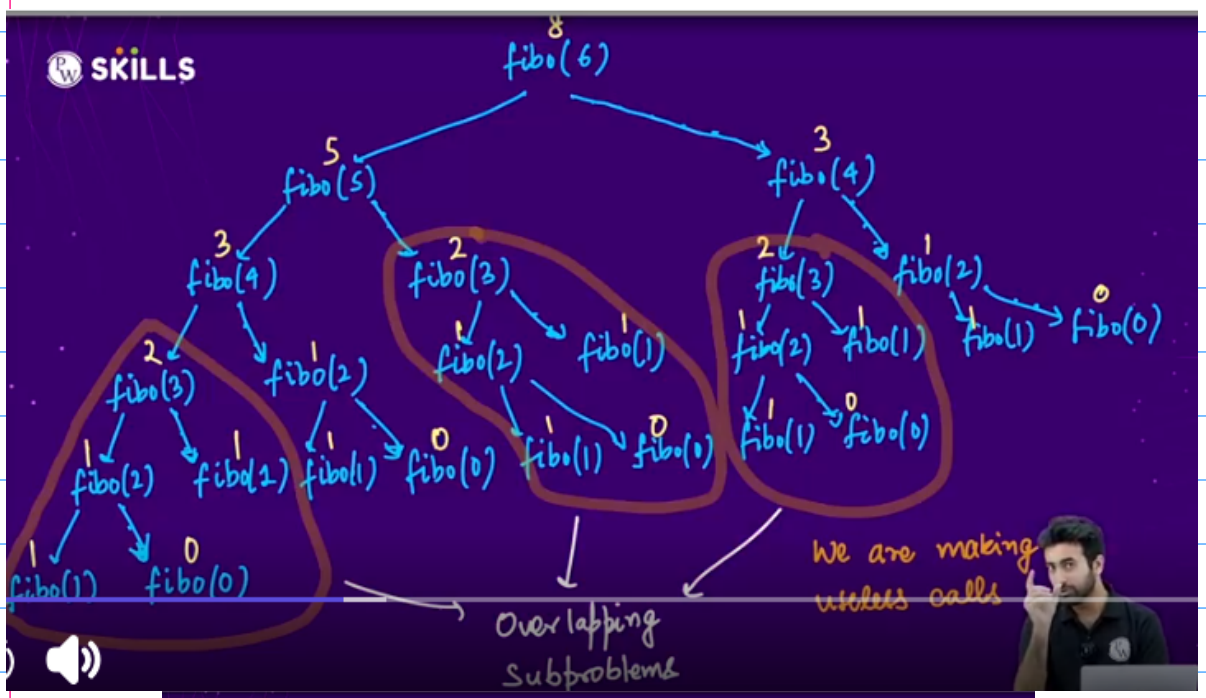
Given `n`, calculate `F(n)`.

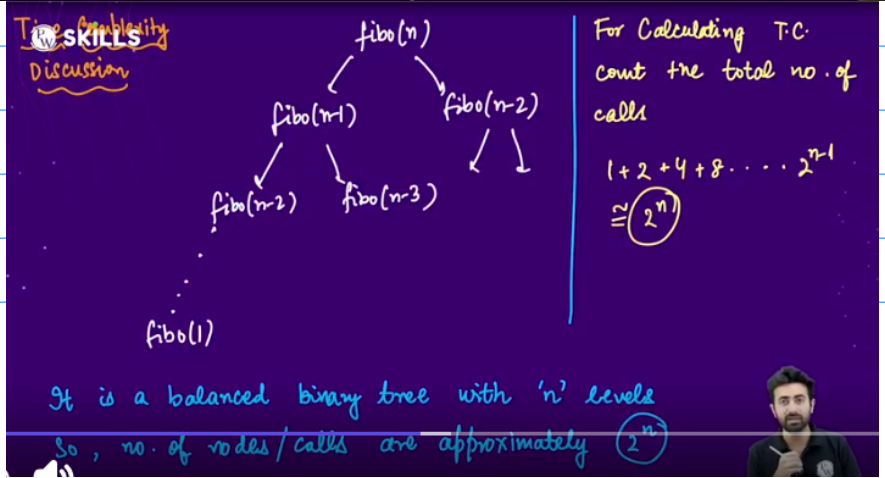
Example 1:

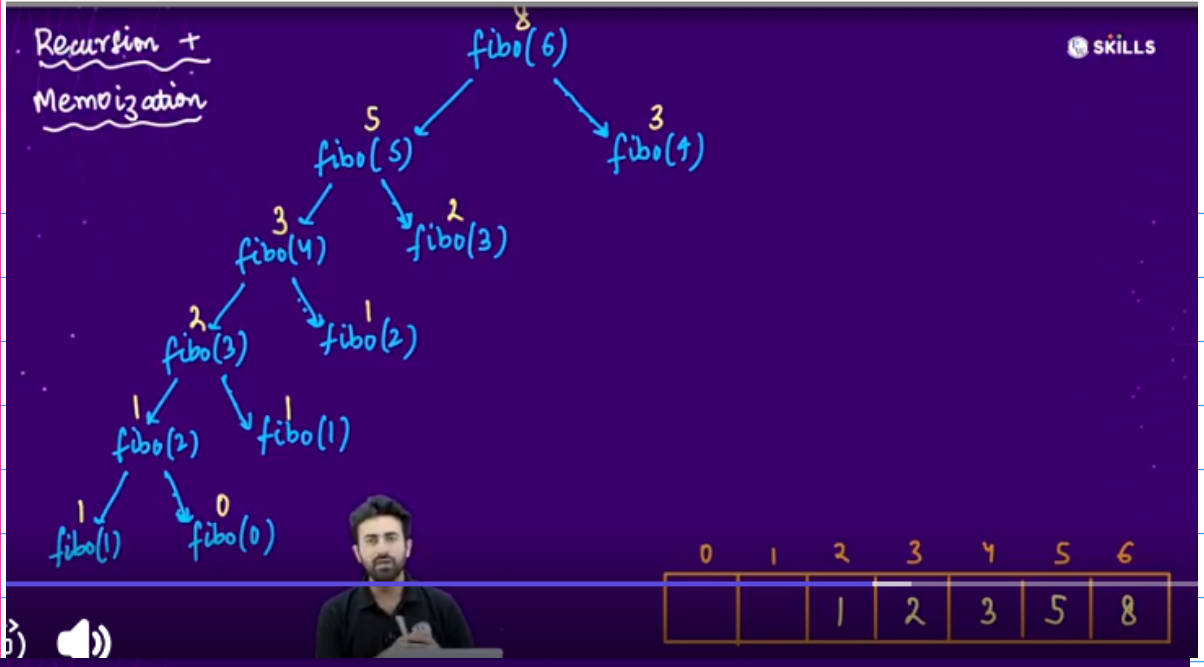
```java
class Solution {
    public int fib(int n) {
        if(n<=1) return n;
        return fib(n-1) + fib(n-2);
    }
}
```

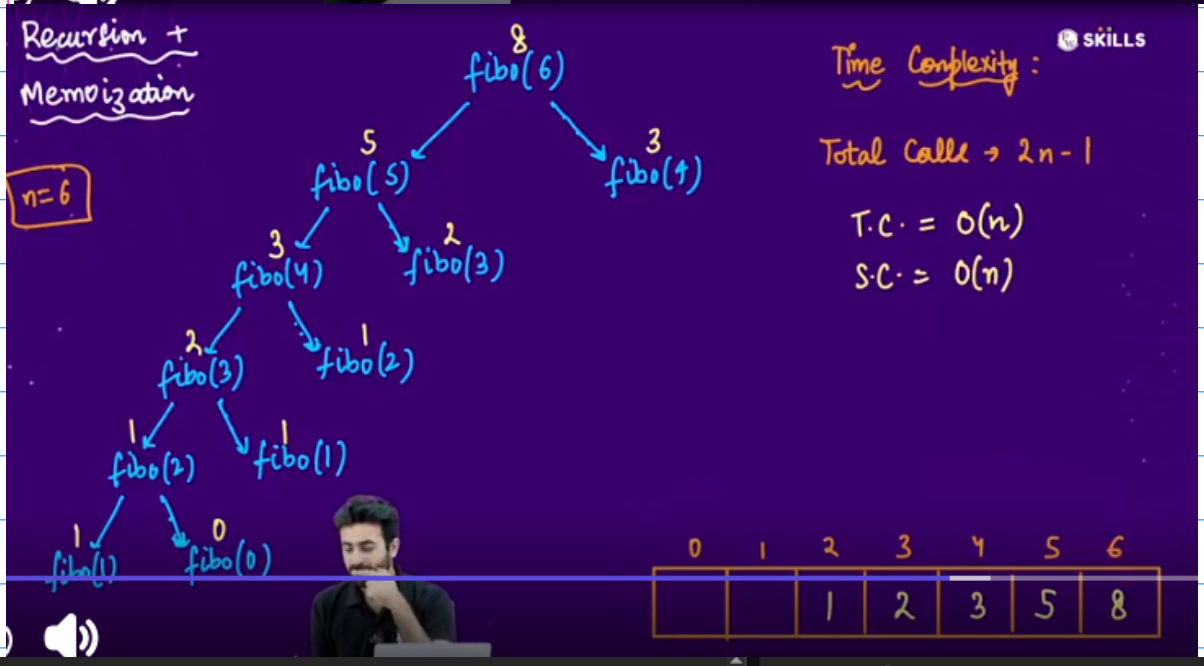
focus on code











509. Fibonacci Number

Solved

Easy | Topics | Companies

The **Fibonacci numbers**, commonly denoted `F(n)` form a sequence, called the **Fibonacci sequence**, such that each number is the sum of the two preceding ones, starting from `0` and `1`. That is,

```
F(0) = 0, F(1) = 1
F(n) = F(n - 1) + F(n - 2), for n > 1.
```

Given `n`, calculate `F(n)`.

dynamicprogramming concept

Example 1:

```
Input: n = 2
Output: 1
```

Java | Auto

```java
class Solution {
    static int[] dp;
    public int fibo(int n){
        if(n<=1) return n;
        if(dp[n]!=0) return dp[n];
        int ans = fibo(n-1)+fibo(n-2);
        dp[n] = ans;
        return ans;
    }
    public int fib(int n) {
        dp = new int[n+1];
     return fibo(n);
    }
}
```

focus on code

509. Fibonacci Number

Solved

Easy | Topics | Companies

The **Fibonacci numbers**, commonly denoted `F(n)` form a sequence, called the **Fibonacci sequence**, such that each number is the sum of the two preceding ones, starting from `0` and `1`. That is,

```
F(0) = 0, F(1) = 1
F(n) = F(n - 1) + F(n - 2), for n > 1.
```

Given `n`, calculate `F(n)`.

Example 1:

```
Input: n = 2
Output: 1
```

Java | Auto

```java
class Solution {
    public int fibo(int n,int[] dp){
        if(n<=1) return n;
        if(dp[n]!=0) return dp[n];
        int ans = fibo(n-1,dp)+fibo(n-2,dp);
        dp[n] = ans;
        return ans;
    }
    public int fib(int n) {
        int[] dp = new int[n+1];
        return fibo(n,dp);
    }
}
```

focus on code

dp(dynamicprogramming) means recursion +memoization

extra space

509. Fibonacci Number

Solved

Easy | Topics | Companies

The **Fibonacci numbers**, commonly denoted `F(n)` form a sequence, called the **Fibonacci sequence**, such that each number is the sum of the two preceding ones, starting from `0` and `1`. That is,

```
F(0) = 0, F(1) = 1
F(n) = F(n - 1) + F(n - 2), for n > 1.
```

Given `n`, calculate `F(n)`.

Java | Auto

```java
class Solution {
    public int fibo(int n,int[] dp){
        if(n<=1) return n;
        if(dp[n]!=0) return dp[n];
        dp[n] =fibo(n-1,dp)+fibo(n-2,dp);
        return dp[n];
    }
    public int fib(int n) {
        int[] dp = new int[n+1];
        return fibo(n,dp);
    }
}
```


509. Fibonacci Number

Solved

Easy | Topics | Companies

The **Fibonacci numbers**, commonly denoted `F(n)` form a sequence, called the **Fibonacci sequence**, such that each number is the sum of the two preceding ones, starting from `0` and `1`. That is,

```
F(0) = 0, F(1) = 1
F(n) = F(n - 1) + F(n - 2), for n > 1.
```

Given `n`, calculate `F(n)`.

Java | Auto

```java
class Solution {
    public int fibo(int n,int[] dp){
        if(n<=1) return n;
        if(dp[n]!=0) return dp[n];
        return dp[n] =fibo(n-1,dp)+fibo(n-2,dp);
        }
    public int fib(int n) {
        int[] dp = new int[n+1];
        return fibo(n,dp);
    }
}
```

Q : Fibonacci Number (Recursion + Memoization)

```java
public int fibo(int n, int[] dp) {
    if(n<=1) return n;
    if(dp[n]!=0) return dp[n];
    return dp[n] = fibo(n-1,dp) + fibo(n-2,dp);
}
public int fib(int n) {
    int[] dp = new int[n+1]; // index from 0 to n
    return fibo(n,dp);
}
```

Recursion + Memoization = Top-Down DP

Q : Fibonacci Number (Tabulation)

n = 8

$dp[i] = dp[i-1] + dp[i-2]$

509. Fibonacci Number

Solved

Easy | Topics | Companies

The **Fibonacci numbers**, commonly denoted `F(n)` form a sequence, called the **Fibonacci sequence**, such that each number is the sum of the two preceding ones, starting from `0` and `1`. That is,

```
F(0) = 0, F(1) = 1
F(n) = F(n - 1) + F(n - 2), for n > 1.
```

Given `n`, calculate `F(n)`.

Java | Auto

```java
class Solution {
    public int fib(int n) {
        if(n<=1) return n;
        int[] dp = new int[n+1];
        dp[0] = 0;
        dp[1] = 1;
        for(int i =2;i<=n;i++) {
          dp[i] = dp[i-1]+dp[i-2];
        }
        return dp[n];
    }
}
```

dp-->dp means using previous result to compute new result

Ques:

Bottom-Up DP = Iterative DP

SKILLS

Q : Fibonacci Number (Tabulation)

n = 8

$dp[i] = dp[i-1] + dp[i-2]$

T.C. = O(n)
S.C. = O(n)

DP = Using previous results to compute new result

Ques:

SKILLS

Q : Min Cost Climbing Stairs

```java
public int minCost(int[] cost, int idx) {
    if(idx==0 || idx==1) return cost[idx];
    return cost[idx] + Math.min(minCost(cost,idx-1),minCost(cost,idx-2));
}
public int minCostClimbingStairs(int[] cost) {
    int n = cost.length;
    return Math.min(minCost(cost,n-1),minCost(cost,n-2));
}
```

T.C. = $O(2^n)$

A.S. = $O(n)$ → Recursive Stack Space

```java
class Solution {
    public int minCost(int[] cost, int idx, int[] dp) {
        if(idx==0 || idx==1) return cost[idx];
        if(dp[idx]!=-1) return dp[idx];
        return dp[idx] = cost[idx] + Math.min(minCost(cost,idx-1,dp),minCost(cost,idx-2,dp));
    }
    public int minCostClimbingStairs(int[] cost) {
        int n = cost.length;
        // n is going from n-1 to 0
        int[] dp = new int[n];
        Arrays.fill(dp,-1);
        return Math.min(minCost(cost,n-1,dp),minCost(cost,n-2,dp));
    }
}
```

746. Min Cost Climbing Stairs

Solved

Easy | Topics | Companies | Hint

You are given an integer array `cost` where `cost[i]` is the cost of `i^th` step on a staircase. Once you pay the cost, you can either climb one or two steps.

You can either start from the step with index `0`, or the step with index `1`.

Return *the minimum cost to reach the top of the floor.*

Example 1:

Input: cost = [10,15,20]
Output: 15

Java | Auto

```java
class Solution {
    public int minCost(int[] cost,int idx,int[] dp){
        if(idx==0||idx==1) return cost[idx];
        if(dp[idx]!=-1) return dp[idx];
        return dp[idx] =cost[idx]+Math.min(minCost(cost,idx-1,dp),minCost(cost,idx-2,dp));

    }
    public int minCostClimbingStairs(int[] cost) {
        int n = cost.length;
        int[] dp = new int[n];
        Arrays.fill(dp,-1);
        return Math.min(minCost(cost,n-1,dp),minCost(cost,n-2,dp));
    }
}
```

Q : Min Cost Climbing Stairs

```java
public int minCost(int[] cost, int idx, int[] dp) {
    if(idx==0 || idx==1) return cost[idx];
    if(dp[idx]!=-1) return dp[idx];
    return dp[idx] = cost[idx] + Math.min(minCost(cost,idx-1,dp),minCost(cost,idx-2,dp));
}
public int minCostClimbingStairs(int[] cost) {
    int n = cost.length;
    // n is going from n-1 to 0
    int[] dp = new int[n];
    Arrays.fill(dp,-1);
    return Math.min(minCost(cost,n-1,dp),minCost(cost,n-2,dp));
}
```

T.C. = $O(n)$

A.S. = $O(n)$

Q : Nth Tribonacci Number

1) Recursion

2) Recursion + Memoization

3) Tabulation

1137. N-th Tribonacci Number

Solved

Easy | Topics | Companies | Hint

The Tribonacci sequence T_n is defined as follows:

$T_0 = 0, T_1 = 1, T_2 = 1$, and $T_{n+3} = T_n + T_{n+1} + T_{n+2}$ for n >= 0.

Given `n`, return the value of T_n.

```java
class Solution {
    public int tribo(int n,int[] dp ){
        if(n<=1) return n;
        if(n==2) return 1;
        if(dp[n]!=0) return dp[n];
        return dp[n] = tribo(n-1,dp)+tribo(n-2,dp)+tribo(n-3,dp);
    }
    public int tribonacci(int n) {
   int[] dp = new int[n+1];
    return tribo(n,dp);
    }
}
```

this is memoization

1137. N-th Tribonacci Number

Solved

Easy | Topics | Companies | Hint

The Tribonacci sequence T_n is defined as follows:

$T_0 = 0, T_1 = 1, T_2 = 1$, and $T_{n+3} = T_n + T_{n+1} + T_{n+2}$ for n >= 0.

Given `n`, return the value of T_n.

Example 1:

```
Input: n = 4
Output: 4
```

```java
class Solution {
    public int tribonacci(int n) {
        if(n==0) return 0;
        if(n==1||n==2) return 1;
       int[] dp= new int[n+1];
       dp[0]  = 0;
       dp[1] = 1;
       dp[2] = 1;
  for(int i = 3;i<=n;i++){
      dp[i] = dp[i-1]+dp[i-2]+dp[i-3];
  }
  return dp[n];
      }
  }
```

this is a tabulation method

1137. N-th Tribonacci Number

Solved

Easy | Topics | Companies | Hint

The Tribonacci sequence T_n is defined as follows:

$T_0 = 0, T_1 = 1, T_2 = 1$, and $T_{n+3} = T_n + T_{n+1} + T_{n+2}$ for n >= 0.

Given `n`, return the value of T_n.

Java | Auto

```java
class Solution {
    public int tribonacci(int n) {
    if(n==0) return 0;
    if(n==1||n==2) return 1;
   int ans =  tribonacci(n-1)+tribonacci(n-2)+tribonacci(n-3);
   return ans;
    }
}
```

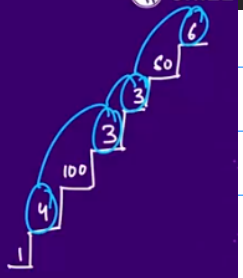
recursion method

Ques:

Q : Min Cost Climbing Stairs (Tabulation)

cost = { 1, 4, 100, 3, 3, 50, 6 }

dp = { 1, 4, 101, 7, 10, 57, 16 }



dp[i] = cost[i] + min(dp[i-1], dp[i-2])

746. Min Cost Climbing Stairs

Solved

Easy | Topics | Companies | Hint

You are given an integer array `cost` where `cost[i]` is the cost of `i^th` step on a staircase. Once you pay the cost, you can either climb one or two steps.

You can either start from the step with index `0`, or the step with index `1`.

Return *the minimum cost to reach the top of the floor.*

```java
class Solution {
    public int minCostClimbingStairs(int[] cost) {
        int n = cost.length;
        int[] dp = new int[n];
        dp[0]=cost[0];
        dp[1]=cost[1];

        for(int i = 2;i<n;i++){
        dp[i] = cost[i]+Math.min(dp[i-2],dp[i-1]);
        }
        return Math.min(dp[n-1],dp[n-2]);
    }
}
```

: Min Cost Climbing Stairs (Tabulation)

```java
public int minCostClimbingStairs(int[] cost) {
    int n = cost.length;
    int[] dp = new int[n];
    dp[0] = cost[0]; dp[1] = cost[1];
    for(int i=2;i<n;i++){
        dp[i] = cost[i] + Math.min(dp[i-2],dp[i-1]);
    }
    return Math.min(dp[n-2],dp[n-1]);
}
```

T.C. = O(n)

A.S. = O(n)

198. House Robber

Solved

Medium | Topics | Companies

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***.

Java | Auto

```java
class Solution {
    public int amount(int[] arr,int i, int[] dp){
        if(i>=arr.length) return 0;
        if(dp[i]!=-1) return dp[i];
int take = arr[i]+amount(arr,i+2,dp);
int skip = amount(arr,i+1,dp);
   return dp[i] =  Math.max(take,skip);
    }
    public int rob(int[] arr) {
        int n = arr.length;
        int[] dp = new int[n];
        Arrays.fill(dp,-1);
      return amount(arr,0,dp);
    }
}
```

Q : House Robber

```java
public int amount(int[] nums, int i, int[] dp) {
    if(i>=nums.length) return 0;
    if(dp[i]!=-1) return dp[i];
    int take = nums[i] + amount(nums,i+2,dp);
    int skip = amount(nums,i+1,dp);
    return dp[i] = Math.max(take,skip);
}
public int rob(int[] nums) {
    // 'i' varies from 0 to n-1
    // dp[i] will store the value of amount(i)
    int[] dp = new int[nums.length]; Arrays.fill(dp,-1);
    return amount(nums,0,dp);
}
```

198. House Robber

Solved

Medium | Topics | Companies

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***.

Java | Auto

```java
class Solution {
    public int rob(int[] nums) {
      int n = nums.length;
      if(n==1) return nums[0];
      int[] dp = new int[n] ;
      dp[0]= nums[0];
      dp[1]=Math.max(nums[0],nums[1]);
      for(int i = 2;i<n;i++){
        dp[i] = Math.max(nums[i]+dp[i-2],dp[i-1]);
      }
      return dp[n-1];
    }
}
```

by tabulation method

: House Robber (Recursion + Memoization)

```java
public int amount(int[] nums, int i, int[] dp) {
    if(i>=nums.length) return 0;
    if(dp[i]!=-1) return dp[i];
    int take = nums[i] + amount(nums,i+2,dp);
    int skip = amount(nums,i+1,dp);
    return dp[i] = Math.max(take,skip);
}
public int rob(int[] nums) {
    // 'i' varies from 0 to n-1
    // dp[i] will store the value of amount(i)
    int[] dp = new int[nums.length]; Arrays.fill(dp,-1);
    return amount(nums,0,dp);
}
```

: House Robber (Tabulation) (Bottom-Up DP)

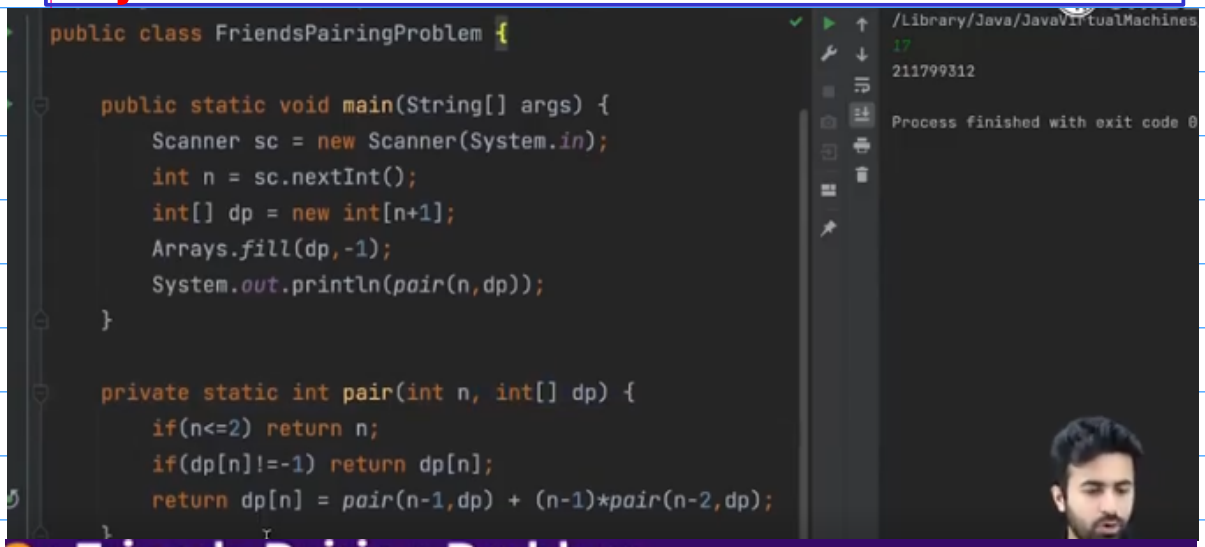
```java
public int rob(int[] arr) {
    int n = arr.length;
    int[] dp = new int[n];
    dp[0] = arr[0];
    if(n>1) dp[1] = Math.max(arr[0],arr[1]);
    for(int i=2;i<n;i++){
        dp[i] = Math.max(arr[i]+dp[i-2], dp[i-1]);
    }
    return dp[n-1];
}
```

Homework:

Q : House Robber II

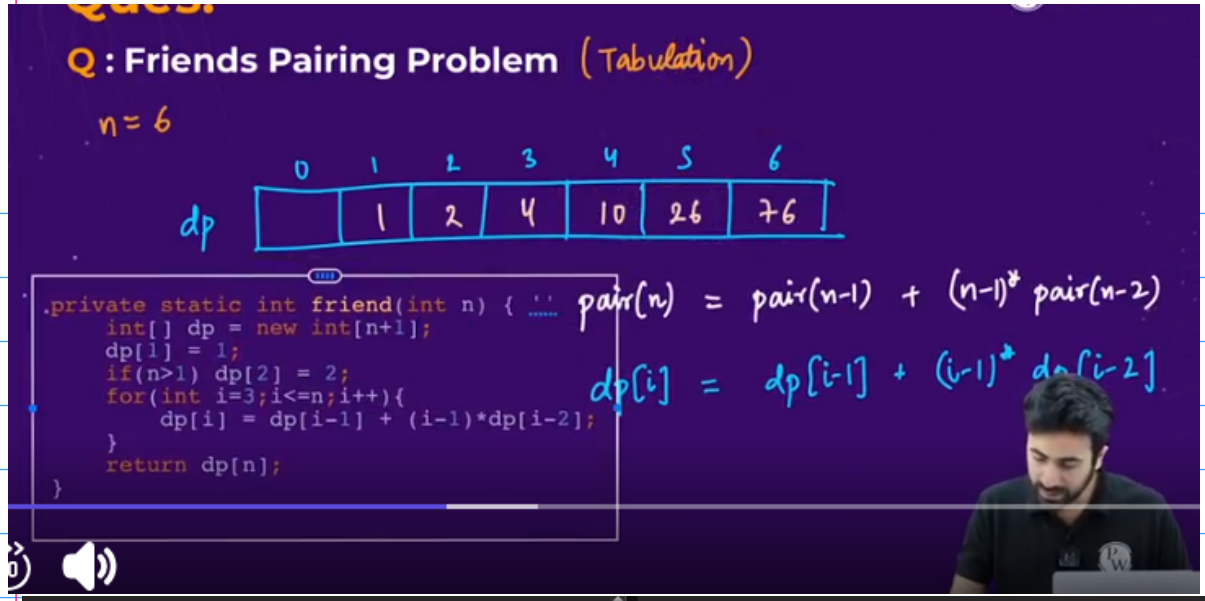
Street is Circular

```java
public class friendsPairingProblem {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("enetr the element:");
        int n = sc.nextInt();
        System.out.println(pair(n));
    }
    private static int pair(int n){
        if(n<=2) return n;
        return pair(n-1)+(n-1)*pair(n-2);

    }
}
```



Q : Friends Pairing Problem

```java
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int n = sc.nextInt();
    int[] dp = new int[n+1];
    Arrays.fill(dp,-1);
    System.out.println(pair(n,dp));
}

private static int pair(int n, int[] dp) {
    if(n<=2) return n;
    if(dp[n]!=-1) return dp[n];
    return dp[n] = pair(n-1,dp) + (n-1)*pair(n-2,dp);
}
```




```
private static int friend(int n) {
    int[] dp = new int[n+1];
    dp[1] = 1;
    if(n>1) dp[2] = 2;
    for(int i=3;i<=n;i++){
        dp[i] = dp[i-1] + (i-1)*dp[i-2];
    }
    return dp[n];
}
```

70. Climbing Stairs

Solved

Easy | Topics | Companies | Hint

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.

Java | Auto

```
class Solution {
    public int climbStairs(int n) {
        if(n==1||n==2) return n;
        int[] dp = new int[n+1];
        dp[1] = 1;
        dp[2] = 2;
        for(int i = 3;i<=n;i++){
            dp[i]=dp[i-1]+dp[i-2];
        }
        return dp[n];

    }
}
```

62. Unique Paths

Solved

Medium | Topics | Companies

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 * 10^9`.

Java | Auto

```
class Solution {
    public int path(int row,int col,int m,int n,int[][] dp){
        if(row>=m||col>=n) return 0;
        if(row==m-1&&col==n-1) return 1;
        if(dp[row][col]!=0) return dp[row][col];
        int rightways = path(row,col+1,m,n,dp);
        int downways = path(row+1,col,m,n,dp);
        return dp[row][col]=rightways+downways;
    }
    public int uniquePaths(int m, int n) {
        int[][] dp = new int[m][n];
       return path(0,0,m,n,dp);


    }
}
```


62. Unique Paths

Solved

Medium | Topics | Companies

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.*

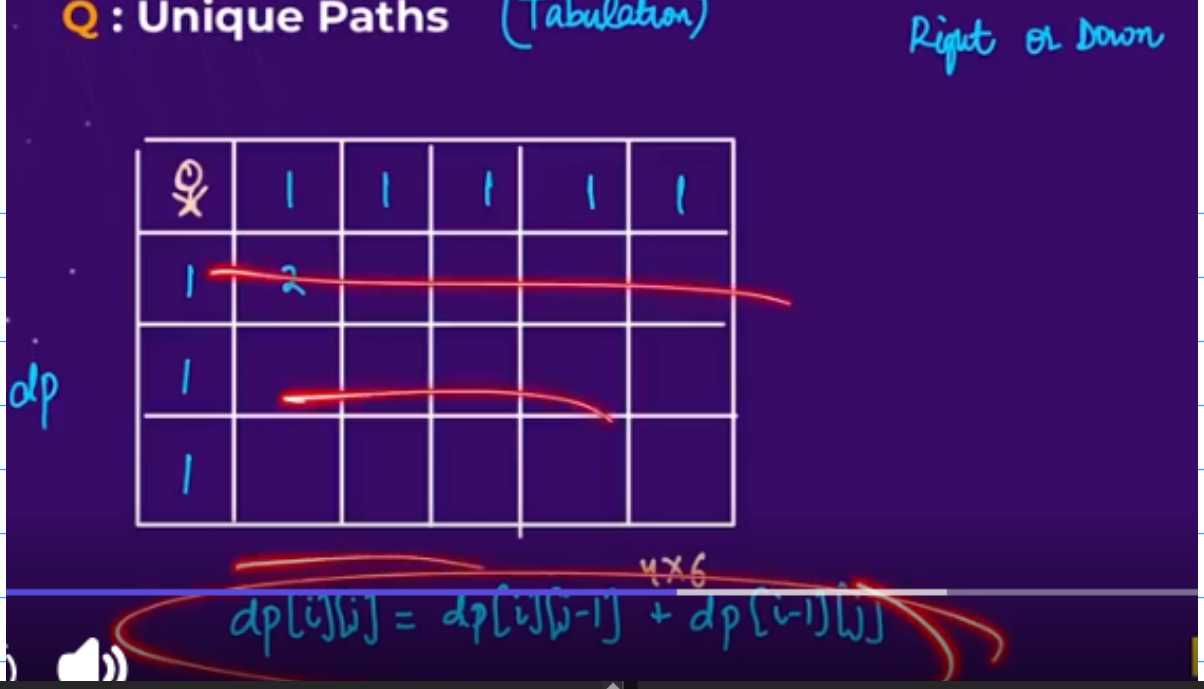
The test cases are generated so that the answer will be less than or equal to `2 * 10^9`.

Java | Auto

```
class Solution {
    public int path(int row,int col,Integer m,Integer n,int[][] dp){
        if(row>=m||col>=n) return 0;
        if(row==m-1&&col==n-1) return 1;
        if(dp[row][col]!=0) return dp[row][col];
        int rightways = path(row,col+1,m,n,dp);
        int downways = path(row+1,col,m,n,dp);
        return dp[row][col]=rightways+downways;
    }
    public int uniquePaths(int m, int n) {
        int[][] dp = new int[m][n];
       return path(0,0,m,n,dp);


    }
}
```

pass by refrence ja rha hai

memoization method use



62. Unique Paths

Solved

Medium | Topics | Companies

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 * 10^9`.

Java | Auto

```java
class Solution {
    public int uniquePaths(int m, int n) {
      int[][] dp = new int[m][n];
      for(int i = 0;i<m;i++){
        for(int j = 0;j<n;j++){
            if(i==0||j==0) dp[i][j] = 1;
            else dp[i][j]=dp[i][j-1]+dp[i-1][j];
        }
      }
      return dp[m-1][n-1];
    }
}
```

tabulation method use

main dynamic programming tabulation hi hota hai

64. Minimum Path Sum

Solved

Medium | Topics | Companies

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.

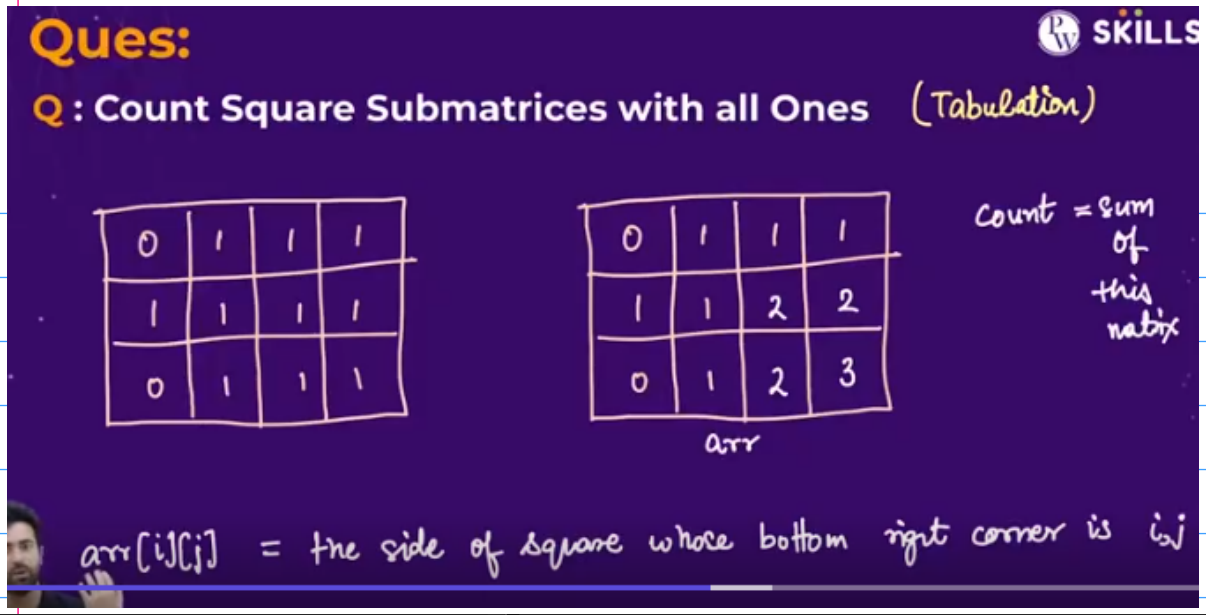
Note: You can only move either down or right at any point in time.

Example 1:

Java | Auto

```java
class Solution {
    public int minPathSum(int[][] arr) {
        int m = arr.length;
        int n = arr[0].length;
      int[][] dp = new int[m][n];
      for(int i = 0;i<m;i++){
        for(int j = 0;j<n;j++){
            if(i==0&&j==0) dp[i][j] = arr[i][j];
            else if(i==0) dp[i][j] = arr[i][j]+dp[i][j-1];
            else if(j==0) dp[i][j] = arr[i][j]+dp[i-1][j];
            else dp[i][j] =arr[i][j]+Math.min(dp[i][j-1],dp[i-1][j]);
        }
      }
      return dp[m-1][n-1];
    }
}
```

tabulation method



1277. Count Square Submatrices with All Ones

Solved

Medium | Topics | Companies | Hint

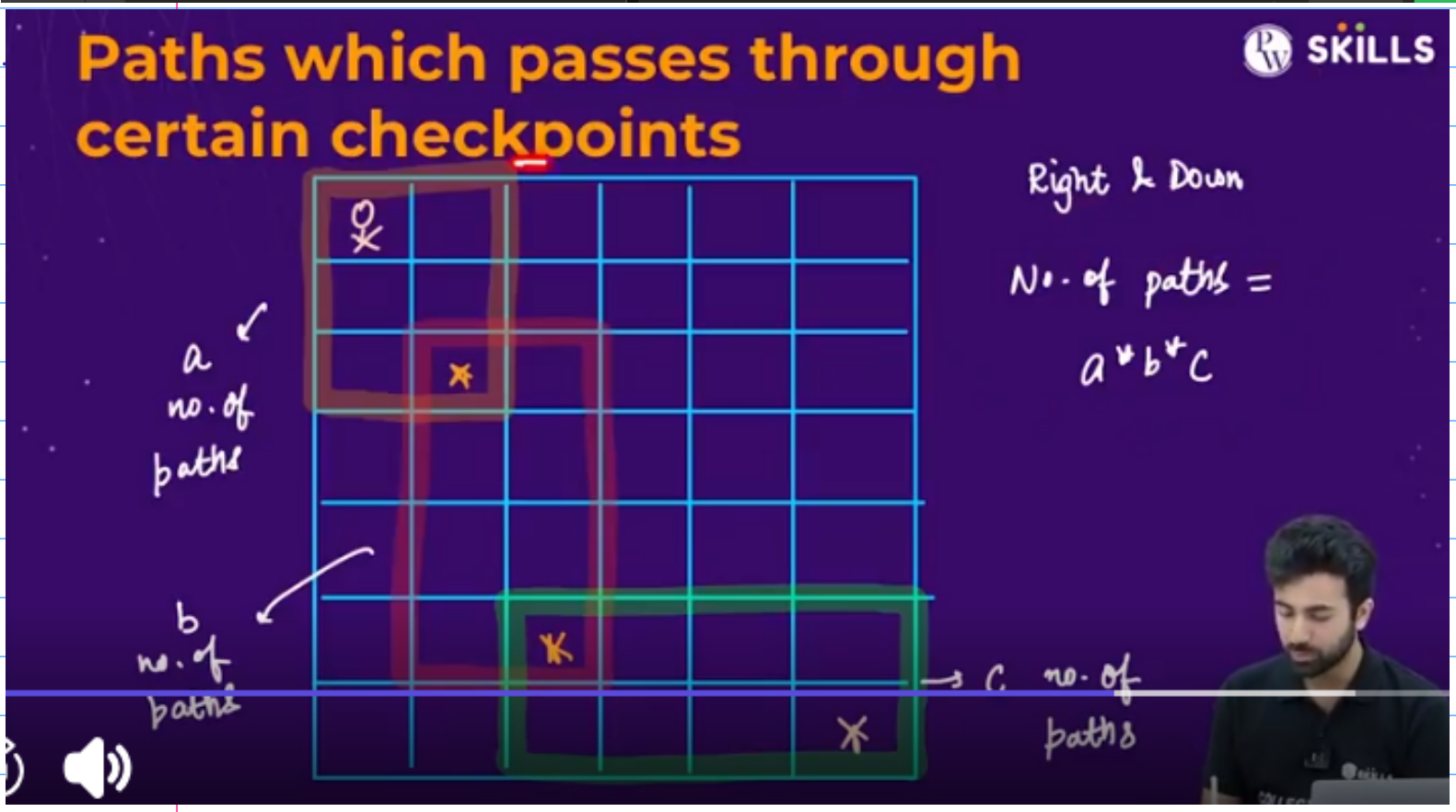
Given a `m * n` matrix of ones and zeros, return how many **square** submatrices have all ones.

Example 1:

```
Input: matrix =
[
  [0,1,1,1],
  [1,1,1,1],
  [0,1,1,1]
]
Output: 15
```

Java | Auto

```java
public int countSquares(int[][] arr) {
 int m = arr.length;
 int n = arr[0].length;
 int count = 0;
 for(int i = 0;i<m;i++){
    for(int j=0;j<n;j++){
        if(arr[i][j]==0) continue;
        if(i>0&&j>0){
            arr[i][j]+=min(arr[i-1][j],arr[i][j-1],arr[i-1][j-1]);
        }
        count+=arr[i][j];
    }
 }
 return count;
}
public int min(int a ,int b,int c){
    return Math.min(a,Math.min(b,c));
}
}
```



Q : Fibonacci Number (Tabulation) (Space Optimization)

n = 8 (21)

	0	1	2
dp	13	21	21

T.C. = O(n)
S.C. = O(1)

dp[2] = dp[0] + dp[1]

dp[0] = dp[1]

dp[1] = dp[2]

509. Fibonacci Number

Solved

Easy | Topics | Companies

The **Fibonacci numbers**, commonly denoted `F(n)` form a sequence, called the **Fibonacci sequence**, such that each number is the sum of the two preceding ones, starting from `0` and `1`. That is,

```
F(0) = 0, F(1) = 1
F(n) = F(n - 1) + F(n - 2), for n > 1.
```

Given `n`, calculate `F(n)`.

space optimization

Java | Auto

```java
class Solution {
    public int fib(int n) {
        if(n<=1) return n;
        int[] dp = new int[3];
        dp[0]=0;dp[1]=1;
        for(int i=1;i<=n-1;i++){
            dp[2] = dp[1]+dp[0];
            dp[0] = dp[1];
            dp[1] = dp[2];
        }
        return dp[2];

    }
}
```


Q : Fibonacci Number (Tabulation) (Space Optimization)

n = 8 (21)

	0	1	2
dp	13	21	21

T.C. = O(n)
S.C. = O(1)

dp[2] = dp[0] + dp[1]

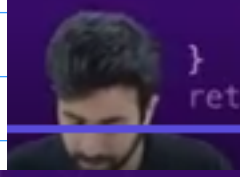
dp[0] = dp[1]

dp[1] = dp[2]

Q : Unique Paths

```java
public int uniquePaths(int m, int n) {
    int[][] dp = new int[2][n];
    for(int j=0;j<n;j++){
        dp[0][j] = 1;
        dp[1][j] = 1;
    }
    for(int i=1;i<=m-1;i++){ // m-1 times
        // DP wala kaam
        for(int j=1;j<n;j++){
            dp[1][j] = dp[1][j-1] + dp[0][j];
        }
        // copy the 1st row to 0th row
        for(int j=1;j<n;j++){
            dp[0][j] = dp[1][j];
        }
    }
    return dp[1][n-1];
```

space optimization



Ques:

Q : Unique Paths

SKILLS

```java
public int uniquePaths(int m, int n) {
    int[][] dp = new int[2][n];
    for(int j=0;j<n;j++){
        dp[0][j] = 1;
        dp[1][j] = 1;
    }
    for(int i=1;i<=m-1;i++){ // m-1 times
        // DP wala kaam
        for(int j=1;j<n;j++){
            dp[1][j] = dp[1][j-1] + dp[0][j];
        }
        // copy the 1st row to 0th row
        for(int j=1;j<n;j++){
            dp[0][j] = dp[1][j];
        }
    }
    return dp[1][n-1];
}
```

```java
public int uniquePaths(int m, int n) {
    int[][] dp = new int[2][n];
    for(int j=0;j<n;j++){
        dp[0][j] = 1;
        dp[1][j] = 1;
    }
    for(int i=1;i<=m-1;i++){ // m-1 times
        if(i%2==1){
            for(int j=1;j<n;j++){
                dp[1][j] = dp[1][j-1] + dp[0][j];
            }
        }
        else{
            for(int j=1;j<n;j++){
                dp[0][j] = dp[0][j-1] + dp[1][j];
            }
        }
    }
    return Math.max(dp[1][n-1],dp[0][n-1]);
}
```

[leetcode 62]

213. House Robber II

Solved

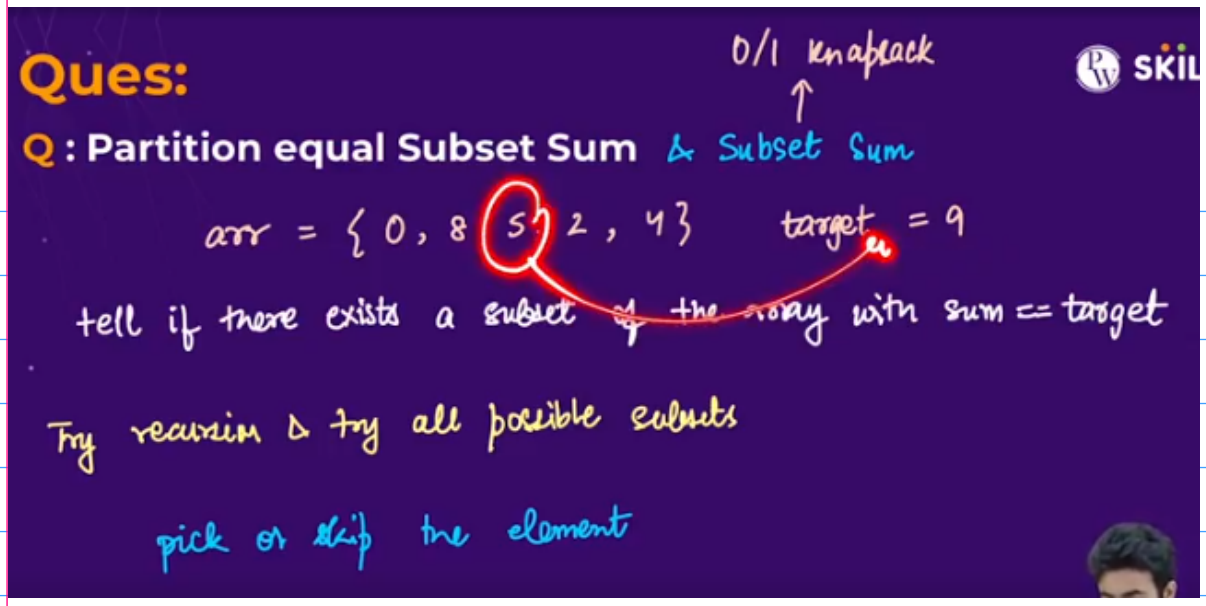
Medium | Topics | Companies | Hint

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***.

```
class Solution {
    public int solve(int[] arr,int l,int r){
        int prev = 0;
        int prevprev = 0;
        for(int i = l;i<=r;i++){
            int skip = prev;
            int take = arr[i]+prevprev;
            int temp = Math.max(skip,take);
            prevprev = prev;
            prev = temp;
        }
        return prev;
    }
    public int rob(int[] arr) {
        int n = arr.length;
   if(n==1) return arr[0];
   if(n==2) return Math.max(arr[0],arr[1]);
   int take_first_house = solve(arr,0,n-2);
   int skip_first_house = solve(arr,1,n-1);
   return Math.max(take_first_house,skip_first_house);
    }
}
```


```
package dynamicprogramming;

public class knapsack {
    public  static int profit(int i ,int[] val,int[] wt,int c){
        if(i==wt.length) return 0;
        int skip = profit(i+1,val,wt,c);
        if(wt[i]>c) return skip;
        int pick = val[i]+profit(i+1,val,wt,c-wt[i]);
        return Math.max(pick,skip);
    }
    public static void main(String[] args) {
        int[] val = {5,3,9,16};
        int[] wt = {1,2,8,10};
        int c = 8;
        System.out.println(profit(0,val,wt,c));

    }
}
```


Ques:
0/1 knapsack
SKIL
Q : Partition equal Subset Sum & Subset Sum
arr = { 0, 8, 5, 2, 4 }
target = 9
tell if there exists a subset of the array with sum == target
Try recursion & try all possible subsets
pick or skip the element

Q : O/1 Knapsack

```
public static int profit(int i, int[] wt, int[] val, int C){
    if(i==wt.length) return 0;
    int skip = profit(i+1,wt,val,C);
    if(wt[i]>C) return skip;
    int pick = val[i] + profit(i+1,wt,val,C-wt[i]);
    return Math.max(pick,skip);
}
```

T.C. = $O(2^n)$

A.S. = $O(n*C)$

```
public  static int profit(int i ,int[] val,int[] wt,int c,int[][] dp){
    if(i==wt.length) return 0;
    if(dp[i][c]!=-1) return dp[i][c];
    int skip = profit(i+1,val,wt,c,dp);
    if(wt[i]>c) return dp[i][c] =skip;
    int pick = val[i]+profit(i+1,val,wt,c-wt[i],dp);
    return dp[i][c] =Math.max(pick,skip);
}
public static void main(String[] args) {
    int[] val = {5,3,9,16};
    int[] wt = {1,2,8,10};
    int c = 8;
    int n = wt.length;
    //i goes 0 to wt.length-1 and c goes c to 0 ;
    int[][] dp = new int[n][c+1];
    for(int i = 0;i<dp.length;i++) {
        for (int j = 0; j < dp[0].length; j++){
            dp[i][j] = -1;
        }
    }
    System.out.println(profit(0,val,wt,c,dp));

}
```

Q : O/1 Knapsack (Recursion + Memoization)

```
public static int profit(int i, int[] wt, int[] val, int C, int[][] dp){
    if(i==wt.length) return 0;
    if(dp[i][C]!=-1) return dp[i][C];
    int skip = profit(i+1,wt,val,C,dp);
    if(wt[i]>C) return dp[i][C] = skip;
    int pick = val[i] + profit(i+1,wt,val,C-wt[i],dp);
    return dp[i][C] = Math.max(pick,skip);
}

public static void main(String[] args) {
    int[] val = {5,3,7,16};
    int[] wt = {1,2,8,10};
    int C = 8;
    int n = wt.length;
    // i = 0 to n-1 | C = C to 0
    int[][] dp = new int[n][C+1];
    for(int i=0;i<dp.length;i++)
        for(int j=0;j<dp[0].length;j++) dp[i][j] = -1;
    System.out.println(profit(0,wt,val,C,dp));
}
```

T.C. = $O(n*C)$

A.S. = $O(n*C)$