08|01|2024

DYNAMIC PROGRAMMING
CLASS - 2

📂 1. House Robber (Leetcode-198)

Ex:01

Nums	4	5	7	1	2
	0	1	2	3	4

Amount of money
Nums[4] = 2

House No = 5
i = 4

FIND Kya Karna Hai?

Return the maximum amount of money

Case:1
Chori Karne wala Ghar

✓ 0th ~~1th~~
✓ 2th ~~3th~~ ~~1th~~
✓ 4th ~~3th~~

amount
4
7
2
Total = 13

Case:2
Chori na Karne wala Ghar

✓ 1th ✗ 0th ✗ 2th
✓ 3th ✗ 2th ✗ 4th

Amount
5
1
Total = 6

$\boxed{\text{max Amount} = 13}$ output

Example : 02

Nums	1	2	3	1
	0	1	2	3

Case-1

Chori Karne wale Ghar			Amount
✓0th	✗1th		1
✓2th	✗1th	✗3th	3
			4

Case-2

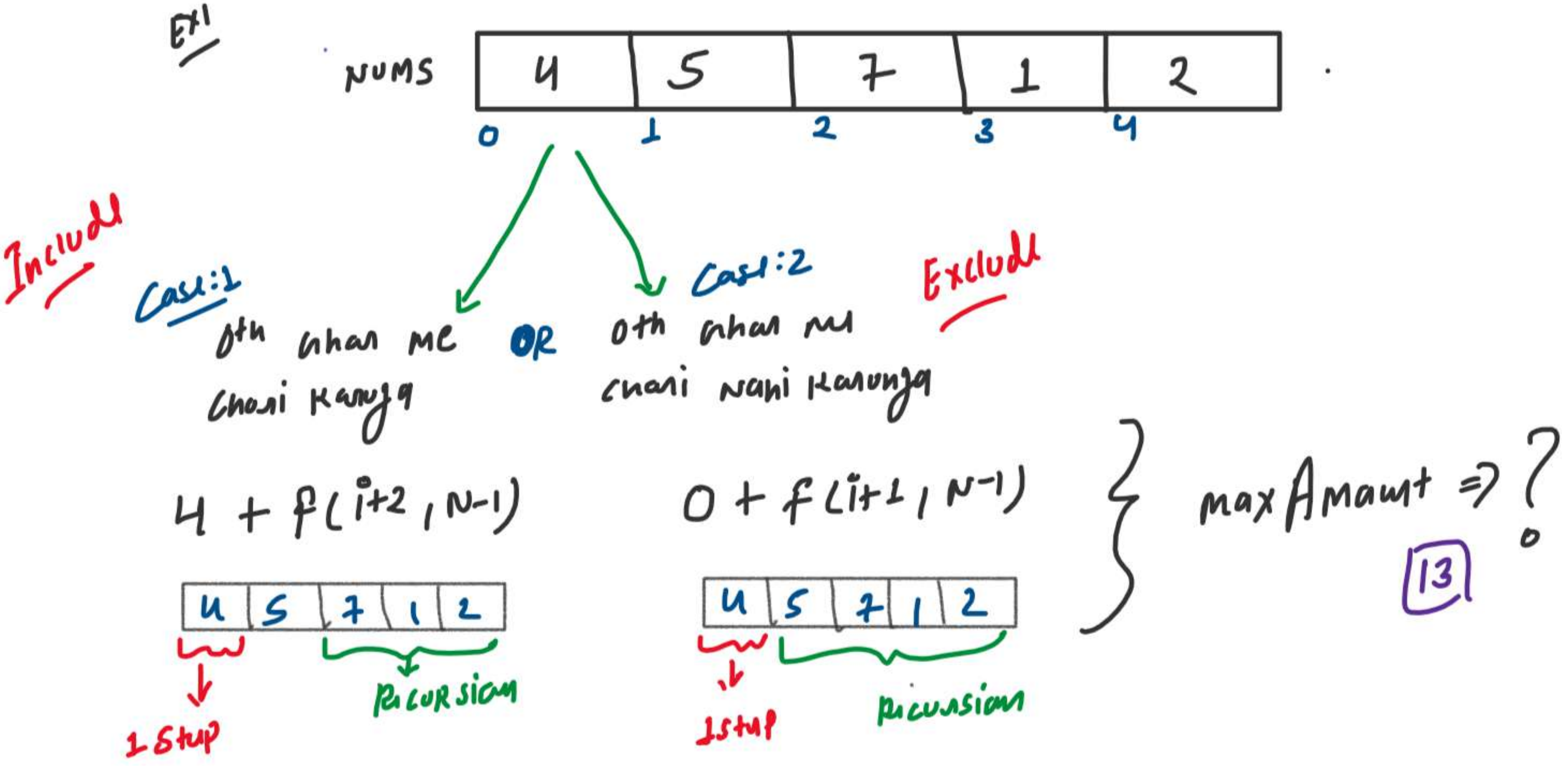
Chori Na Karne wale Ghar			Amount
✓1th	✗0th	✗2th	2
✓3th	✗2th		1
			3

maxAmount = 4

output

Approach 1: Recursion

Inclusive and Exclusive Pattern



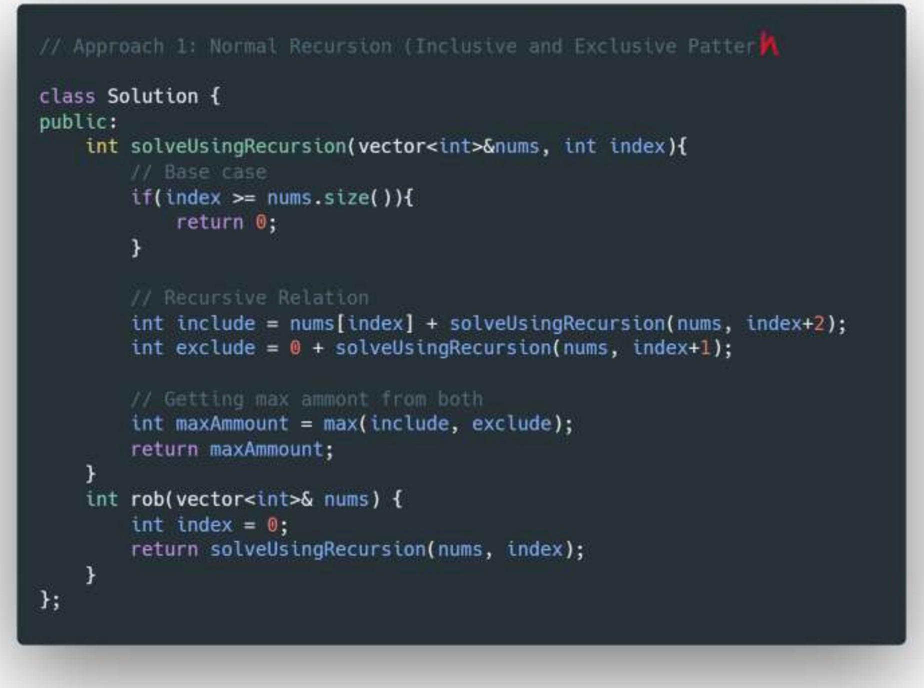
Case:1 — 0th ghar me chori karunga

Case:2 — 0th ghar me chori nahi karunga

$4 + f(i+2, N-1)$

$0 + f(i+1, N-1)$

maxAmount ⇒ ?



```
// Approach 1: Normal Recursion (Inclusive and Exclusive Pattern

class Solution {
public:
    int solveUsingRecursion(vector<int>&nums, int index){
        // Base case
        if(index >= nums.size()){
            return 0;
        }

        // Recursive Relation
        int include = nums[index] + solveUsingRecursion(nums, index+2);
        int exclude = 0 + solveUsingRecursion(nums, index+1);

        // Getting max ammont from both
        int maxAmmount = max(include, exclude);
        return maxAmmount;
    }
    int rob(vector<int>& nums) {
        int index = 0;
        return solveUsingRecursion(nums, index);
    }
};
```

DRY RUN ON EX1

nums	4	5	7	1	2
	0	1	2	3	4

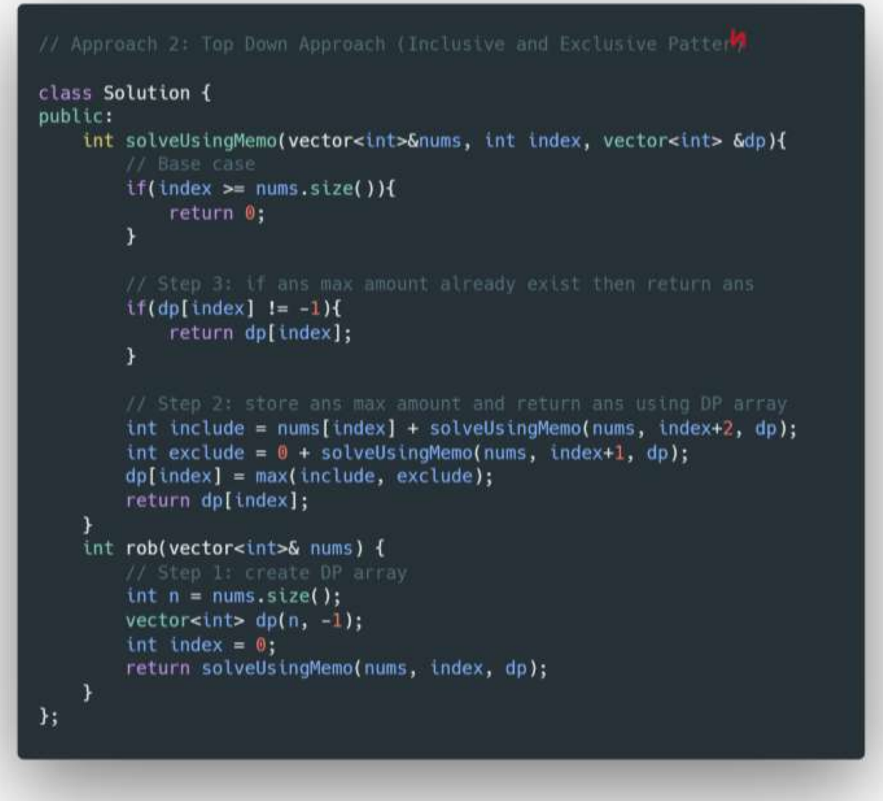
include	exclude	maxamount
4	5	5
4+7=11	5+1=6	11
11+2=13	6	13

↳ Final Ans

Approach 2: Top Down
Inclusive and Exclusive Pattern

N
↑
O

TOP DOWN: Traverse from 0 to N



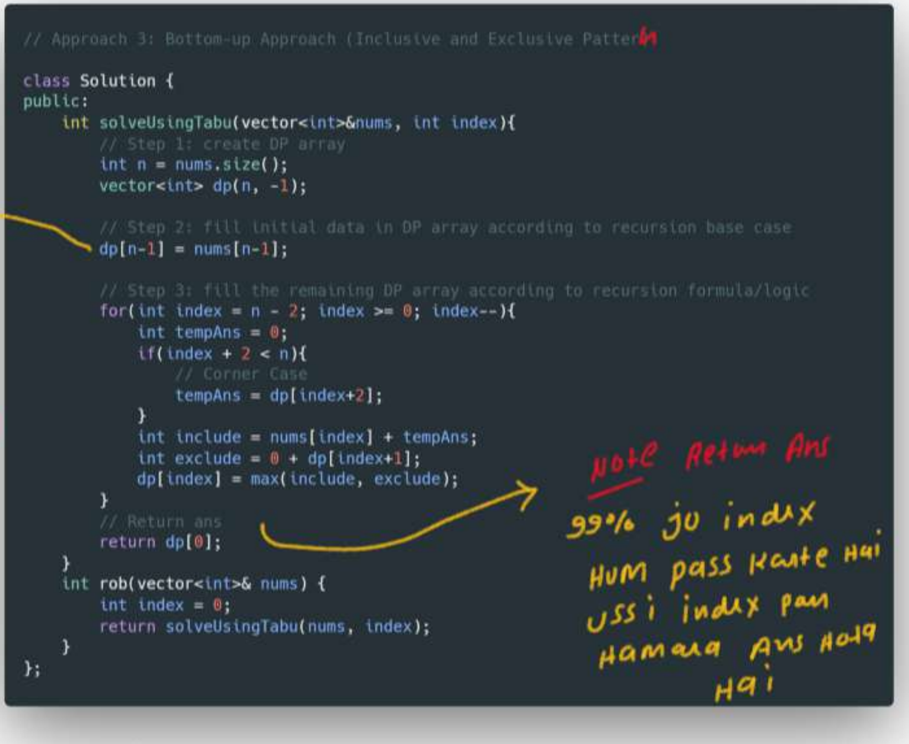
```cpp
// Approach 2: Top Down Approach (Inclusive and Exclusive Pattern

class Solution {
public:
    int solveUsingMemo(vector<int>&nums, int index, vector<int> &dp){
        // Base case
        if(index >= nums.size()){
            return 0;
        }

        // Step 3: if ans max amount already exist then return ans
        if(dp[index] != -1){
            return dp[index];
        }

        // Step 2: store ans max amount and return ans using DP array
        int include = nums[index] + solveUsingMemo(nums, index+2, dp);
        int exclude = 0 + solveUsingMemo(nums, index+1, dp);
        dp[index] = max(include, exclude);
        return dp[index];
    }
    int rob(vector<int>& nums) {
        // Step 1: create DP array
        int n = nums.size();
        vector<int> dp(n, -1);
        int index = 0;
        return solveUsingMemo(nums, index, dp);
    }
};
```

Approach 3: Bottom UP
Inclusive and Exclusive Pattern

N
↓
O

BOTTOM UP: Traverse from N to 0

Main whole ARRAY ka ans Nahi Bata Skta HUN To main Manually Last Index ka Amount ASS main store kar diya Hai



```cpp
// Approach 3: Bottom-up Approach (Inclusive and Exclusive Pattern

class Solution {
public:
    int solveUsingTabu(vector<int>&nums, int index){
        // Step 1: create DP array
        int n = nums.size();
        vector<int> dp(n, -1);

        // Step 2: fill initial data in DP array according to recursion base case
        dp[n-1] = nums[n-1];

        // Step 3: fill the remaining DP array according to recursion formula/logic
        for(int index = n - 2; index >= 0; index--){
            int tempAns = 0;
            if(index + 2 < n){
                // Corner Case
                tempAns = dp[index+2];
            }
            int include = nums[index] + tempAns;
            int exclude = 0 + dp[index+1];
            dp[index] = max(include, exclude);
        }
        // Return ans
        return dp[0];
    }
    int rob(vector<int>& nums) {
        int index = 0;
        return solveUsingTabu(nums, index);
    }
};
```

Note Return Ans

99% jo index Hum pass karte Hai ussi index par Hamara Ans Hota Hai

Bottom up DRY RUN

NUMS	4	5	7	1	2
	0	1	2	3	4

STEP1

DP	13	9	9	2	2
	0	1	2	3	4

n = 5

STEP2

DP[5-1] = NUM[5-1]

DP[4] = 2

Initialy

(Jab Last me Chori karunga To sirf meri pass utna Hi Amount Hoga jitna uski pass Aa KHA Huaa Hai)

STEP3

index	Include	Exclude	DP[Index] max Amount
3	1 + 0 = 1	0 + 2 = 2	2
2	7 + 2 = 9	0 + 2 = 2	9
1	5 + 2 = 9	0 + 9 = 9	9
0	4 + 9 = 13	0 + 9 = 9	$\boxed{13}$

→ Return DP[0]

Approach 4: Space Optimization

Inclusive and Exclusive Pattern

Ex

0	1	2	3	4
4	5	7	1	2

nums

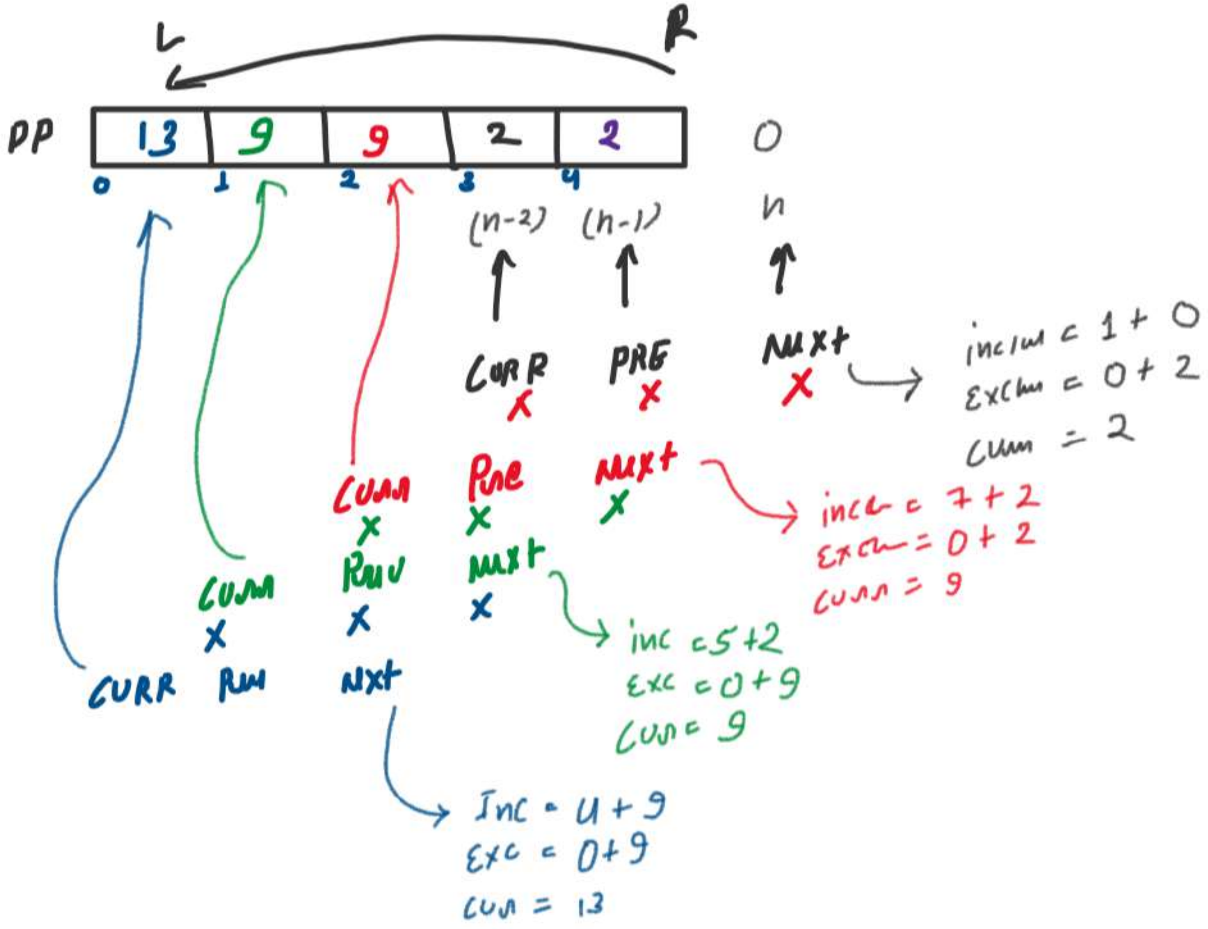
CURR → maxAmount

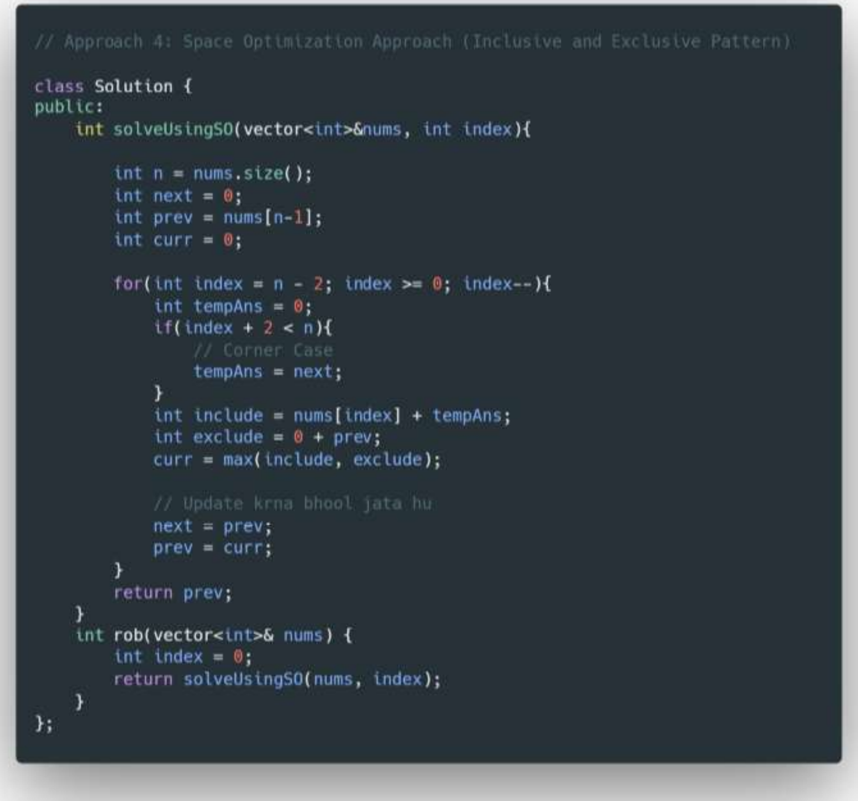
Prev → nums[n-1]

Next → 0

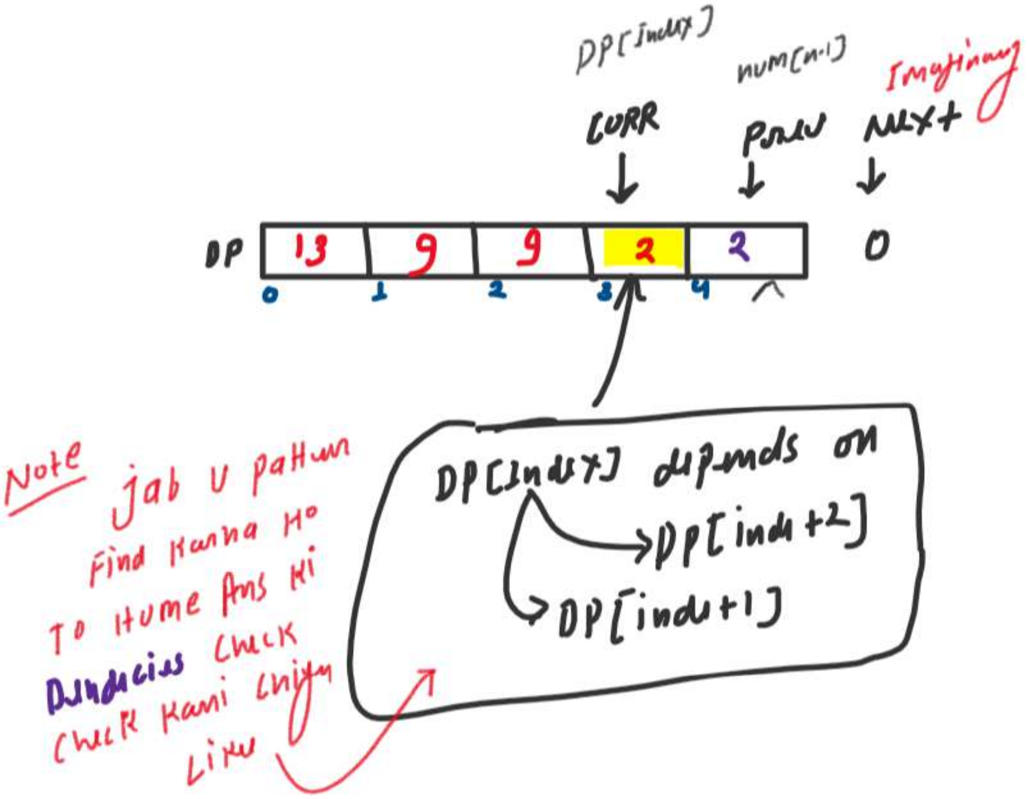
T.C. = O(N)

S.C. = O(1)





```cpp
// Approach 4: Space Optimization Approach (Inclusive and Exclusive Pattern)

class Solution {
public:
    int solveUsingSO(vector<int>&nums, int index){

        int n = nums.size();
        int next = 0;
        int prev = nums[n-1];
        int curr = 0;

        for(int index = n - 2; index >= 0; index--){
            int tempAns = 0;
            if(index + 2 < n){
                // Corner Case
                tempAns = next;
            }
            int include = nums[index] + tempAns;
            int exclude = 0 + prev;
            curr = max(include, exclude);

            // Update krna bhool jata hu
            next = prev;
            prev = curr;
        }
        return prev;
    }
    int rob(vector<int>& nums) {
        int index = 0;
        return solveUsingSO(nums, index);
    }
};
```

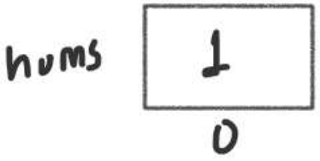


Note

Jab U pattern Find Karna Ho To Hume Ans Ki Dependencies Check Check Karni Chahiye Like

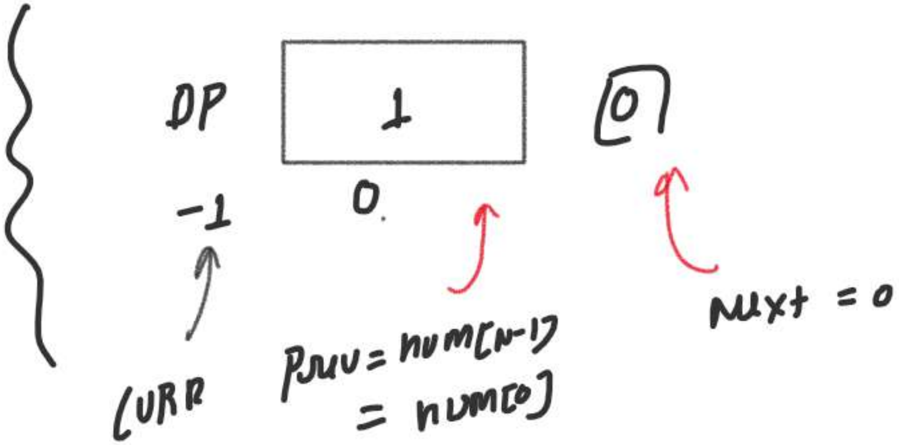
Why return prev

Ex



size = n = 1

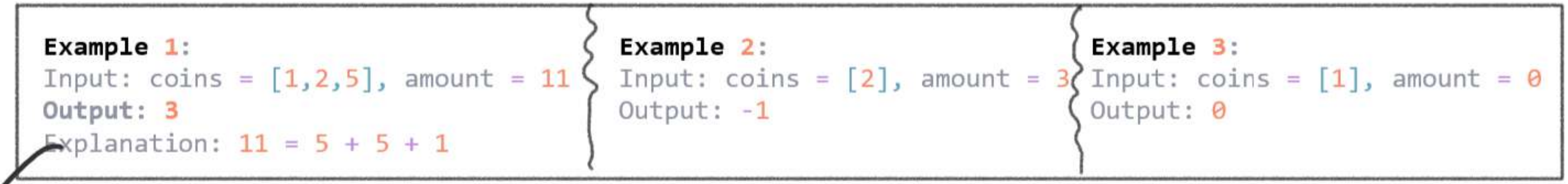
Output => $\boxed{1}$



prev = num[n-1]
= num[0]
= 1

return $\boxed{\text{prev}}$

2. Coin Change (Leetcode-322)



```
Example 1:
Input: coins = [1,2,5], amount = 11
Output: 3
Explanation: 11 = 5 + 5 + 1
```

```
Example 2:
Input: coins = [2], amount = 3
Output: -1
```

```
Example 3:
Input: coins = [1], amount = 0
Output: 0
```

way 1

Coin = 1 Takes 11 times ⇒ 11 × 1 = 11   11 coins

Coin = 2 Takes 5 times & Coin = 1 takes 1 time ⇒ (5 × 2) + 1 = 11   6 coins

Coin = 5 takes 2 times & Coin = 1 takes 1 time ⇒ (2 × 5) + 1 = 11   3 coins

min(11 , 6 , 3) ⇒ 3 coins

Approach 1: Recursion

Explore All Possible Ways Pattern

way 2

Amount = 11

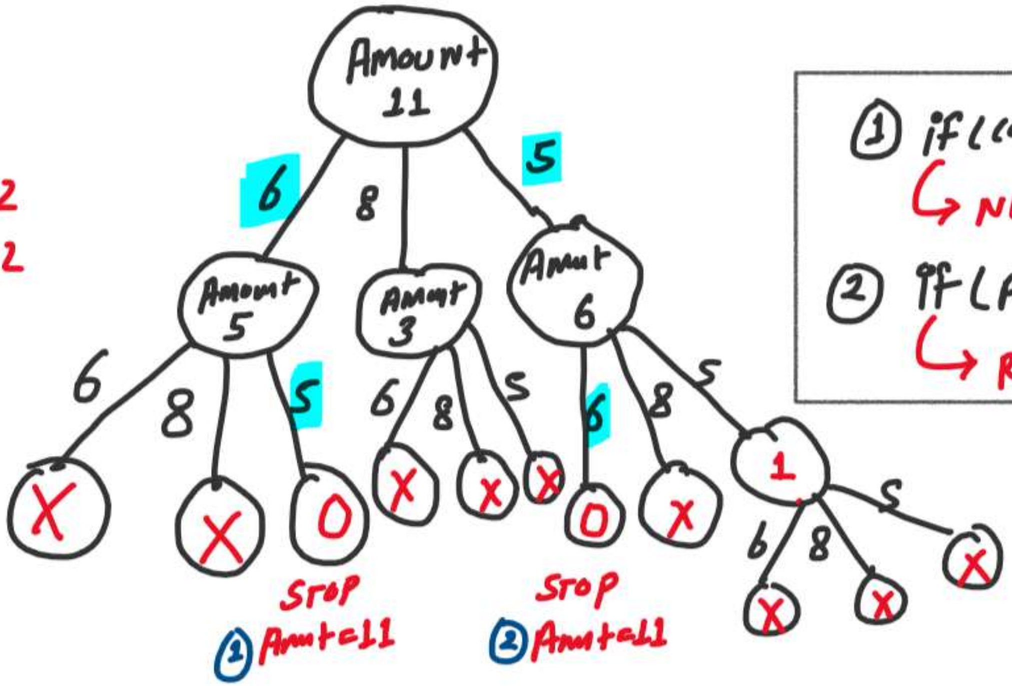
Coins = 6, 8, 5

call ① 6 + 5 = 11 } coins = 2

② 5 + 6 = 11 } coins = 2

min(2,2) = 2

Output



① if (coin > Amount)
↳ No Recursion call

② if (Amount == 0)
↳ Return 0 coin

↳ Base case

Way:3 Amount = N = 11

Coins [6,8,5]

$$\left.\begin{array}{l} f(N) = f(11) \xrightarrow{\text{6 coin}} f(5) \Rightarrow \boxed{\overset{\text{1 STEP}}{\boxed{1}} + f(N-6)} \\[2em] f(N) = f(11) \xrightarrow{\text{8 coin}} f(3) \Rightarrow \boxed{1 + f(N-8)} \\[2em] f(N) = f(11) \xrightarrow{\text{5 coin}} f(6) \Rightarrow \boxed{1 + f(N-5)} \end{array}\right\} \text{MIN}$$

{ But AISA zaroori
Nahi Hai Ki Har Bar
3 Types Ka Coin Hi
Ho USSE Jada
Bhi Ho Skata
Hai To
Har Ek Coin Ko
Check Karne Ke
Liye Loop Ka
Use Karenge. KI
(coin<=Amount)
Hai Ya Nahi }

DRY RUN

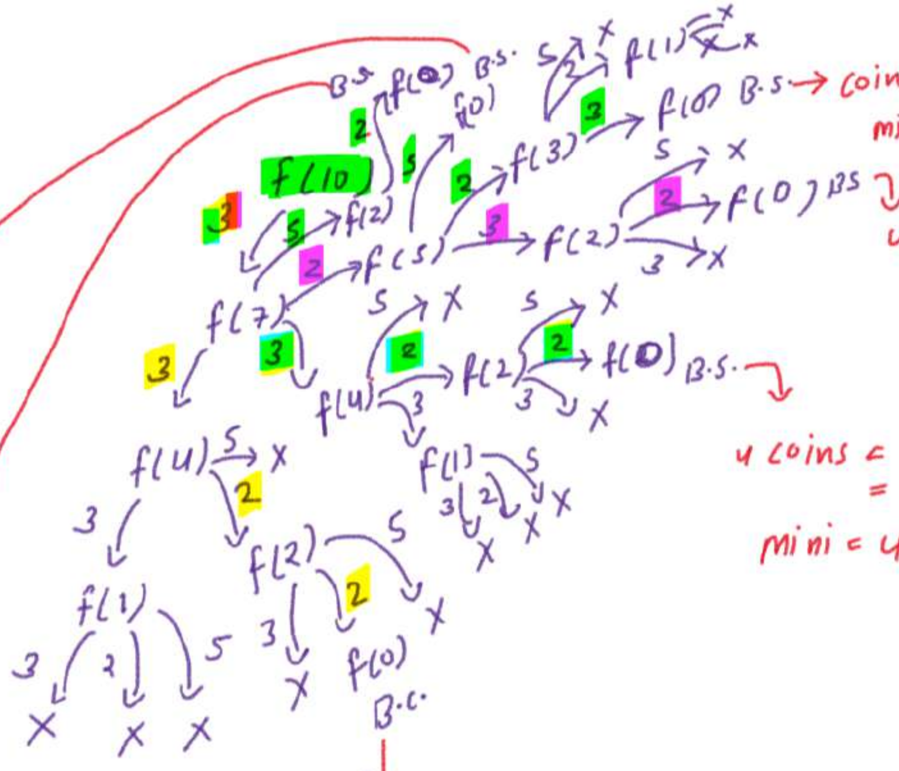
Ex

Coins = [3 , 2 , 5]
0 1 2

Amount = 10

index = 0

mini = 3
Coins = 3+5+2



Coins = 3 + 2 + 5
= 10
mini = 3

coins = 3 + 5 + 2
= 10
mini = 3

coins = 3+2 + 2+3
= 10
mini = 4

4 coins = 3+2 + 3+2
= 10
mini = 4

4 coins = 3 +3 +2 +2
= 10
mini = 4

4 coins = 3+3 + 2+2
= 10
mini = 4

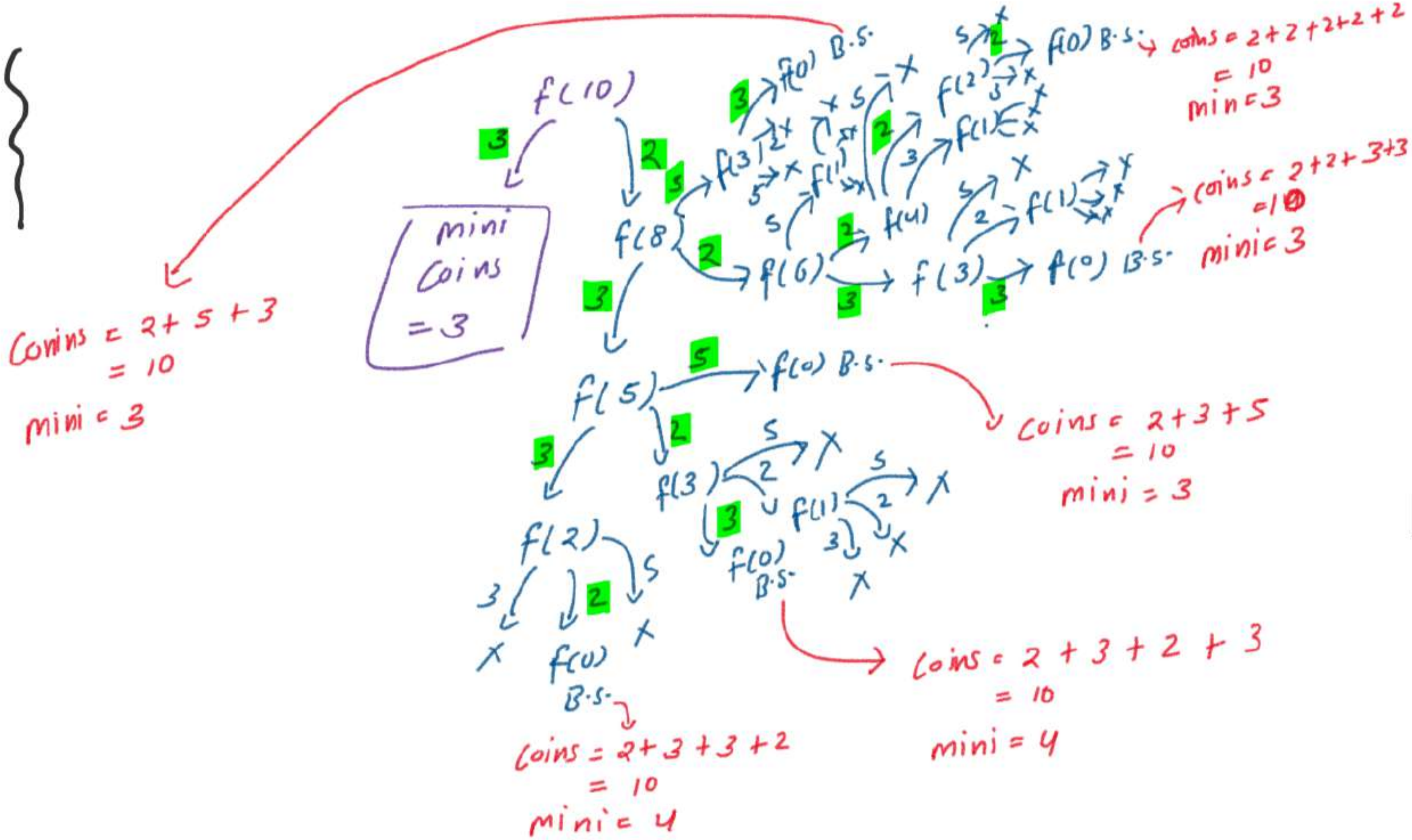
Ex

Coins = [3 , 2 , 5]
0 1 2

Amount = 10

index = 1

mini = 3
Coins = 2+5+3



Ex

Coins = [3 , 2 , 5]
         0   1   2

Amount = 10

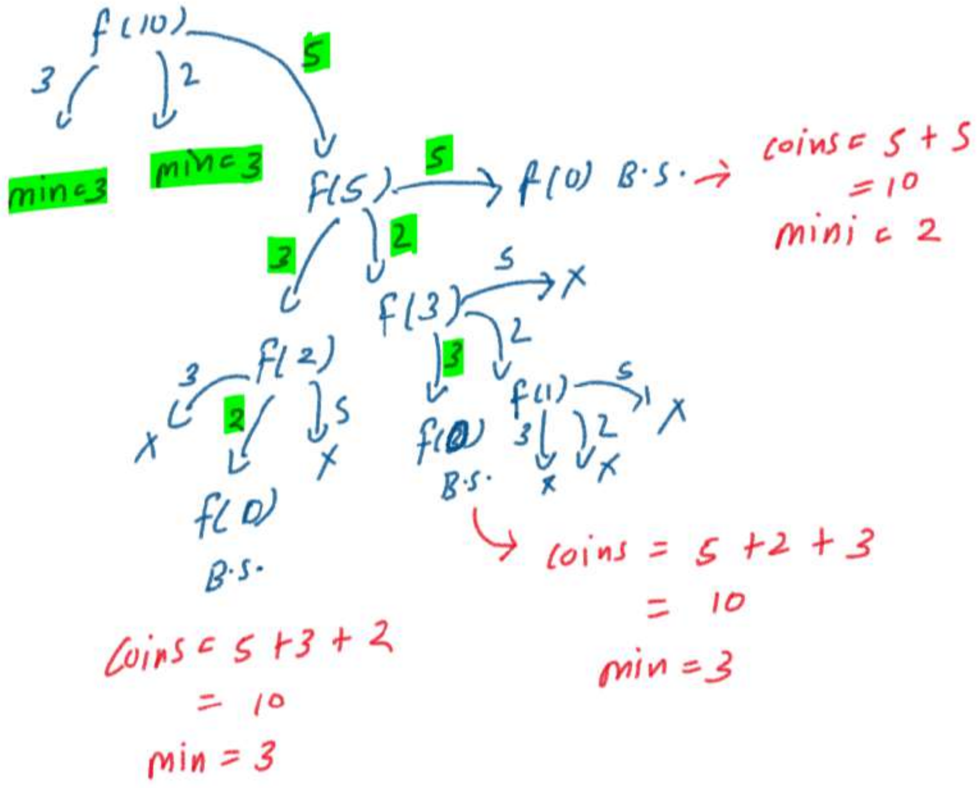
index = 2

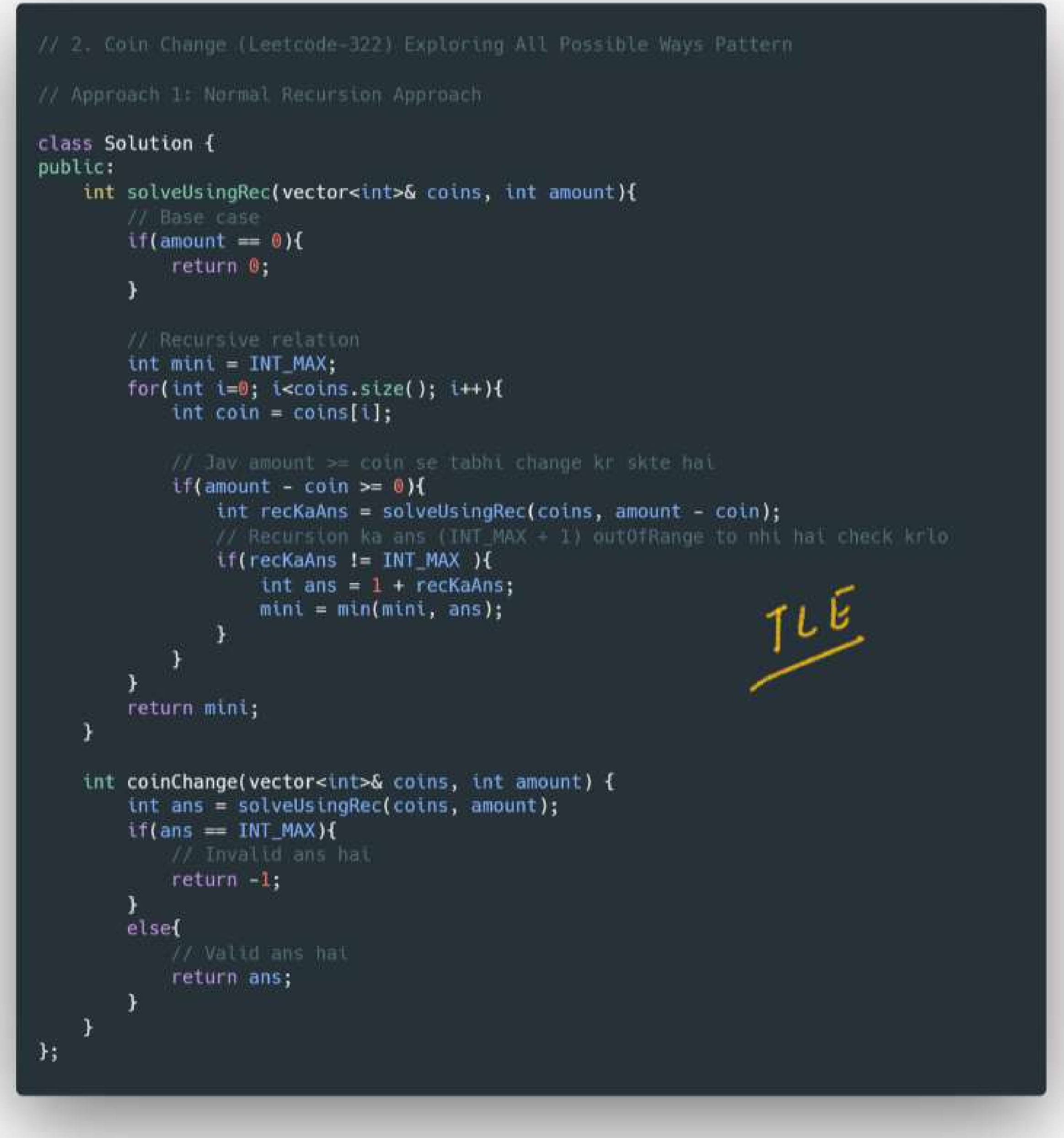
mini = 2

Coins = 5+5

Final output

→ 2





```
// 2. Coin Change (Leetcode-322) Exploring All Possible Ways Pattern

// Approach 1: Normal Recursion Approach

class Solution {
public:
    int solveUsingRec(vector<int>& coins, int amount){
        // Base case
        if(amount == 0){
            return 0;
        }

        // Recursive relation
        int mini = INT_MAX;
        for(int i=0; i<coins.size(); i++){
            int coin = coins[i];

            // Jav amount >= coin se tabhi change kr skte hai
            if(amount - coin >= 0){
                int recKaAns = solveUsingRec(coins, amount - coin);
                // Recursion ka ans (INT_MAX + 1) outOfRange to nhi hai check krlo
                if(recKaAns != INT_MAX ){
                    int ans = 1 + recKaAns;
                    mini = min(mini, ans);
                }
            }
        }
        return mini;
    }

    int coinChange(vector<int>& coins, int amount) {
        int ans = solveUsingRec(coins, amount);
        if(ans == INT_MAX){
            // Invalid ans hai
            return -1;
        }
        else{
            // Valid ans hai
            return ans;
        }
    }
};
```

TLE

Approach 2: Top Down
Explore All Possible Ways Pattern

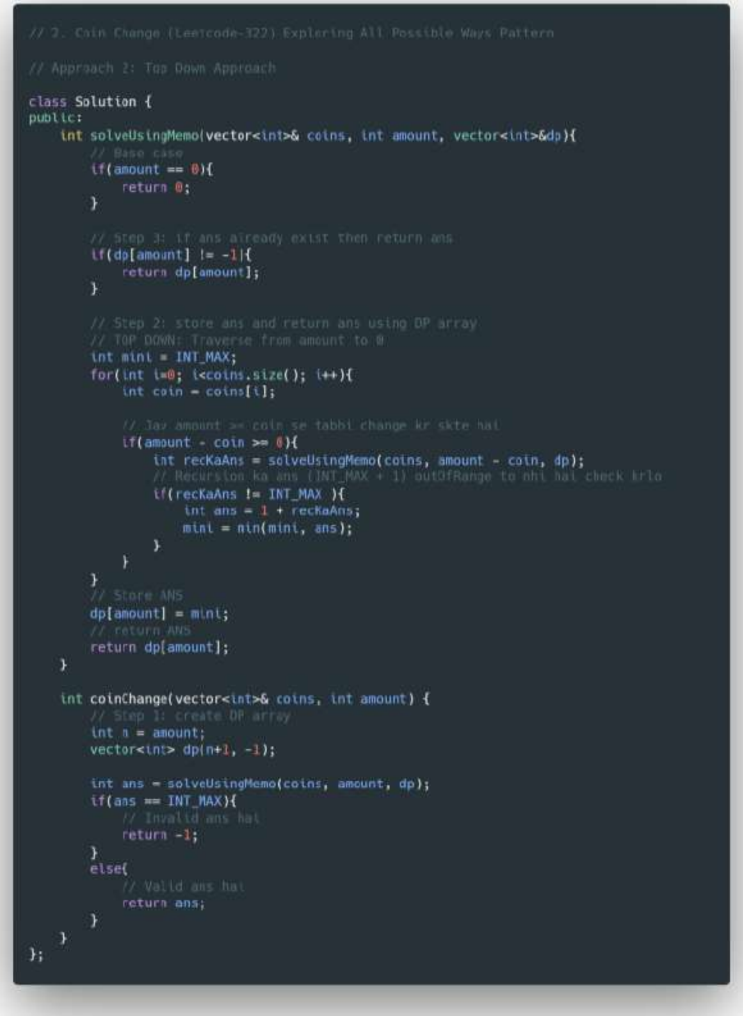
TOP DOWN: Traverse from amount to 0

STEP1,2 (size =Am+1)

DP	mini	mini	mini	. . .	mini	mini
	AM0	AM1	AM2	. . . .	AM	Am+1

Kitna Amount
↳ Kam se kam coin Exchange karega

STEP3 Agar kisi Amount ke minimum coins Ko DP me store kar diya Hoi To Dobara store nahi karugi. [Return dp[AM]]



```
// 2. Coin Change (Leetcode-322) Exploring All Possible Ways Pattern

// Approach 2: Top Down Approach

class Solution {
public:
    int solveUsingMemo(vector<int>& coins, int amount, vector<int>&dp){
        // Base case
        if(amount == 0){
            return 0;
        }

        // Step 3: if ans already exist then return ans
        if(dp[amount] != -1){
            return dp[amount];
        }

        // Step 2: store ans and return ans using DP array
        // TOP DOWN: Traverse from amount to 0
        int mini = INT_MAX;
        for(int i=0; i<coins.size(); i++){
            int coin = coins[i];

            // Jav amount >= coin se tabhi change kr skte hai
            if(amount - coin >= 0){
                int recKaAns = solveUsingMemo(coins, amount - coin, dp);
                // Recursion ka ans (INT_MAX + 1) outOfRange to nhi hai check krlo
                if(recKaAns != INT_MAX ){
                    int ans = 1 + recKaAns;
                    mini = min(mini, ans);
                }
            }
        }
        // Store ANS
        dp[amount] = mini;
        // return ANS
        return dp[amount];
    }

    int coinChange(vector<int>& coins, int amount) {
        // Step 1: create DP array
        int n = amount;
        vector<int> dp(n+1, -1);

        int ans = solveUsingMemo(coins, amount, dp);
        if(ans == INT_MAX){
            // Invalid ans hai
            return -1;
        }
        else{
            // Valid ans hai
            return ans;
        }
    }
};
```

Approach 3: Bottom Up

Explore All Possible Ways Pattern

BOTTOM UP: Traverse from 0 to amount

Ex
Coins = [3 , 2 , 5]
0 1 2
Amount = 10

STEP1

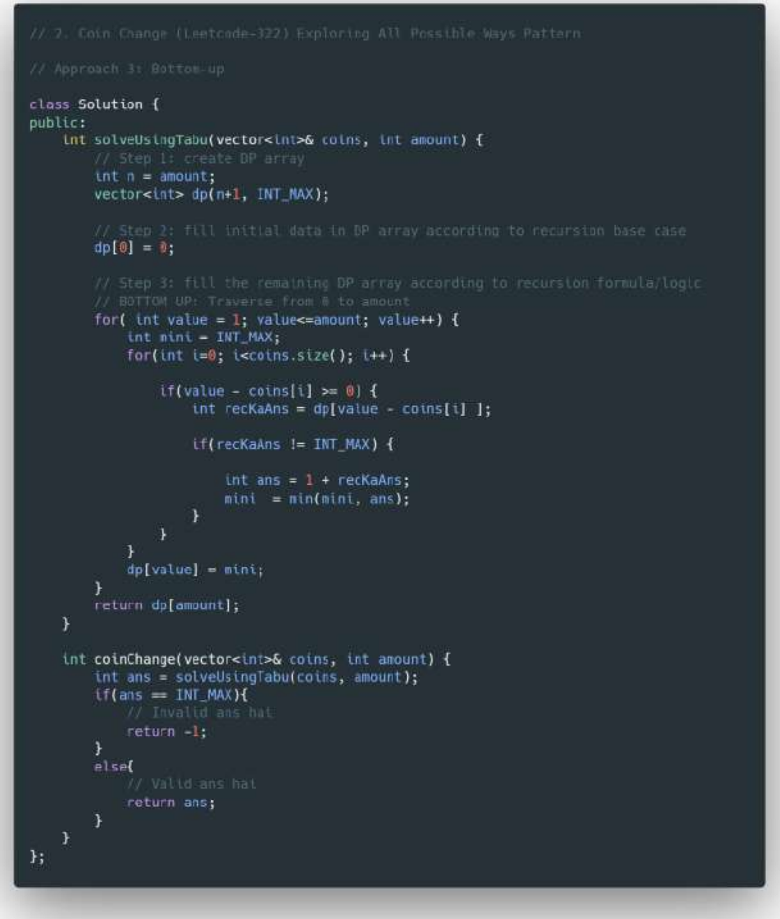
DP	~~IM~~ 0	IM	IM	IM	IM	IM	IM	IM	IM	IM	IM
Amount →	0	1	2	3	4	5	6	7	8	9.	10

STEP2 Analys- Base case and store initial data to DP

Amount == 0
return 0

DP[amount] = 0 , when Amount = 0

STEP3 fill remaining ans based on Recursive Relation



```
// 2. Coin Change (Leetcode-322) Exploring All Possible Ways Pattern

// Approach 3: Bottom-up

class Solution {
public:
    int solveUsingTabu(vector<int>& coins, int amount) {
        // Step 1: create DP array
        int n = amount;
        vector<int> dp(n+1, INT_MAX);

        // Step 2: fill initial data in DP array according to recursion base case
        dp[0] = 0;

        // Step 3: fill the remaining DP array according to recursion formula/logic
        // BOTTOM UP: Traverse from 0 to amount
        for( int value = 1; value<=amount; value++) {
            int mini = INT_MAX;
            for(int i=0; i<coins.size(); i++) {

                if(value - coins[i] >= 0) {
                    int recKaAns = dp[value - coins[i] ];

                    if(recKaAns != INT_MAX) {

                        int ans = 1 + recKaAns;
                        mini  = min(mini, ans);
                    }
                }
            }
            dp[value] = mini;
        }
        return dp[amount];
    }

    int coinChange(vector<int>& coins, int amount) {
        int ans = solveUsingTabu(coins, amount);
        if(ans == INT_MAX){
            // Invalid ans hai
            return -1;
        }
        else{
            // Valid ans hai
            return ans;
        }
    }
};
```

STEP3

2 वाले के लिए only 1 coin ki zaroorat Hai.

DP (value)	0 ~~IM~~	IM ~~IM~~	1 ~~IM~~	1 IM	2 IM	1 IM	2 IM	2 IM	2 IM	3 IM	2 IM
Amount →	0	1	2	3	4	5	6	7	8	9.	10
			(2)	(3)	(2+2)	(5)	(3+3)	(5+2)	(5+3)	(3+3+3)	(5+5)

Ex
Coins = [3, 2, 5]
         0 1 2
Amount = 10

value	mini	i	value - coins[i] >= 0	recAns = DP[-]	recAns != IM	ans = 1 + recAns	mini	i++	DP[value] = mini	value++
1	IM	0	1 - 3 ✗	—	—	—	—	1		
	IM	1	1 - 2 ✗	—	—	—	—	2		
	IM	2	1 - 5 ✗	—	—	—	—	3		
	IM	3 stop	—	—	—	—	—	—	IM	2
2	IM	0	2-3 ✗	—	—	—	—	1		
	IM	1	2-2 ✓	0	✓	1	1	2		
	1	2	2-5 ✗	—	—	—	—	3		
	1	3 stop	—	—	—	—	—	—	1	3

value	mini	i	value - coins[i] >= 0	recAns = DP[—]	recAns != IM	ans = 1 + recAns	mini	i++	DP[value] = mini	value++
1	IM	0	1-3 ✗	—	—	—	—	1		
	IM	1	1-2 ✗	—	—	—	—	2		
	IM	2	1-5 ✗	—	—	—	—	3		
	IM	3 stop	—	—	—	—	—	—	IM	2
2	IM	0	2-3 ✗	—	—	—	—	1		
	IM	1	2-2 ✓	0	✓	1	1	2		
	1	2	2-5 ✗	—	—	—	—	3		
	1	3 stop	—	—	—	—	—	—	1	3
3	IM	0	3-3 ✓	0	✓	1	1	1		
	1	1	3-2 ✓	1	✓	2	1	2		
	1	2	3-5 ✗	—	—	—	—	3		
	1	3 stop	—	—	—	—	—	—	1	4

DRY RUN AS IT

Approach 4: Optimal Solution

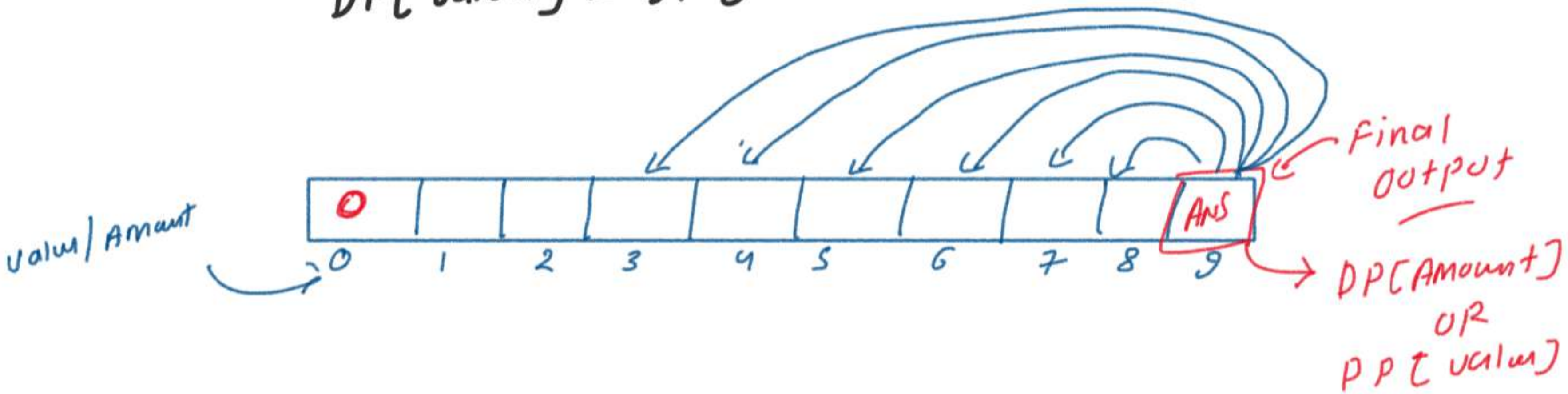
Explore All Possible Ways Pattern

(This Approach is not possible)

why ?

output
DP[Amount] Depends on whole Array
so Yanha koi pattern Ban nahi pa Rha Hai

$$DP[value] = DP[\ value - coins[i]\ ]$$


value/Amount
0
ANS
0 1 2 3 4 5 6 7 8 9
Final output
DP[Amount]
OR
DP[value]