Dynamic Programming - I & Doubt Clearing Session

Foundation Course on Data Structures & Algorithms - Part II

Love Babbar • Lesson 24 • Aug 18, 2022

Dynamic Programming → (Recursion) → Master

what -> ?

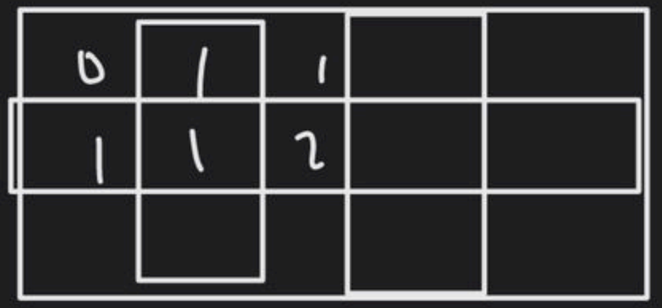
↳ technique

↳ to solve problem

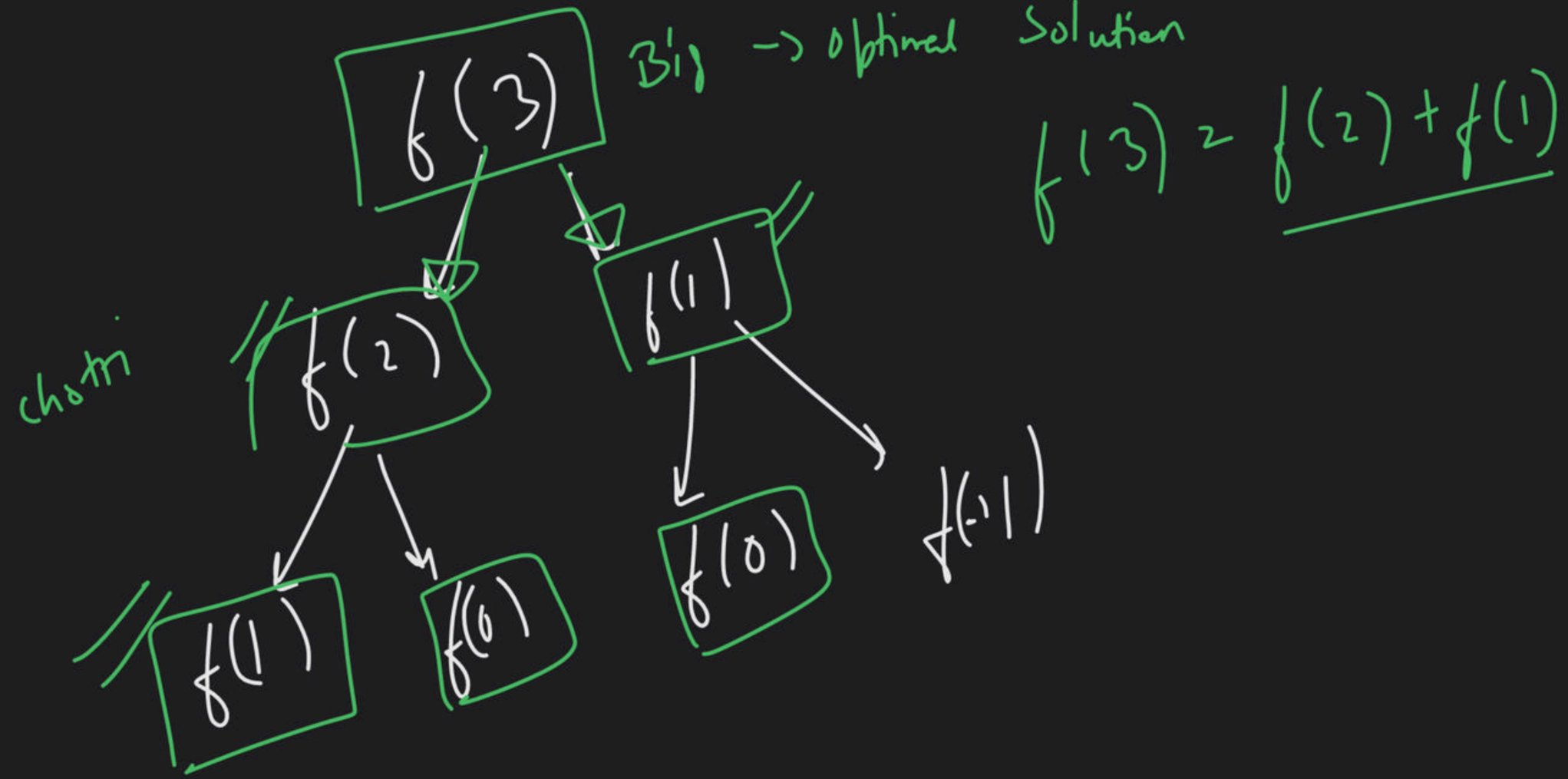
- Overlapping Subproblem
- Optimal Substructure

फिबोनाची

- Top-Down [Rec + Mem]
- Bottom-Up [table]



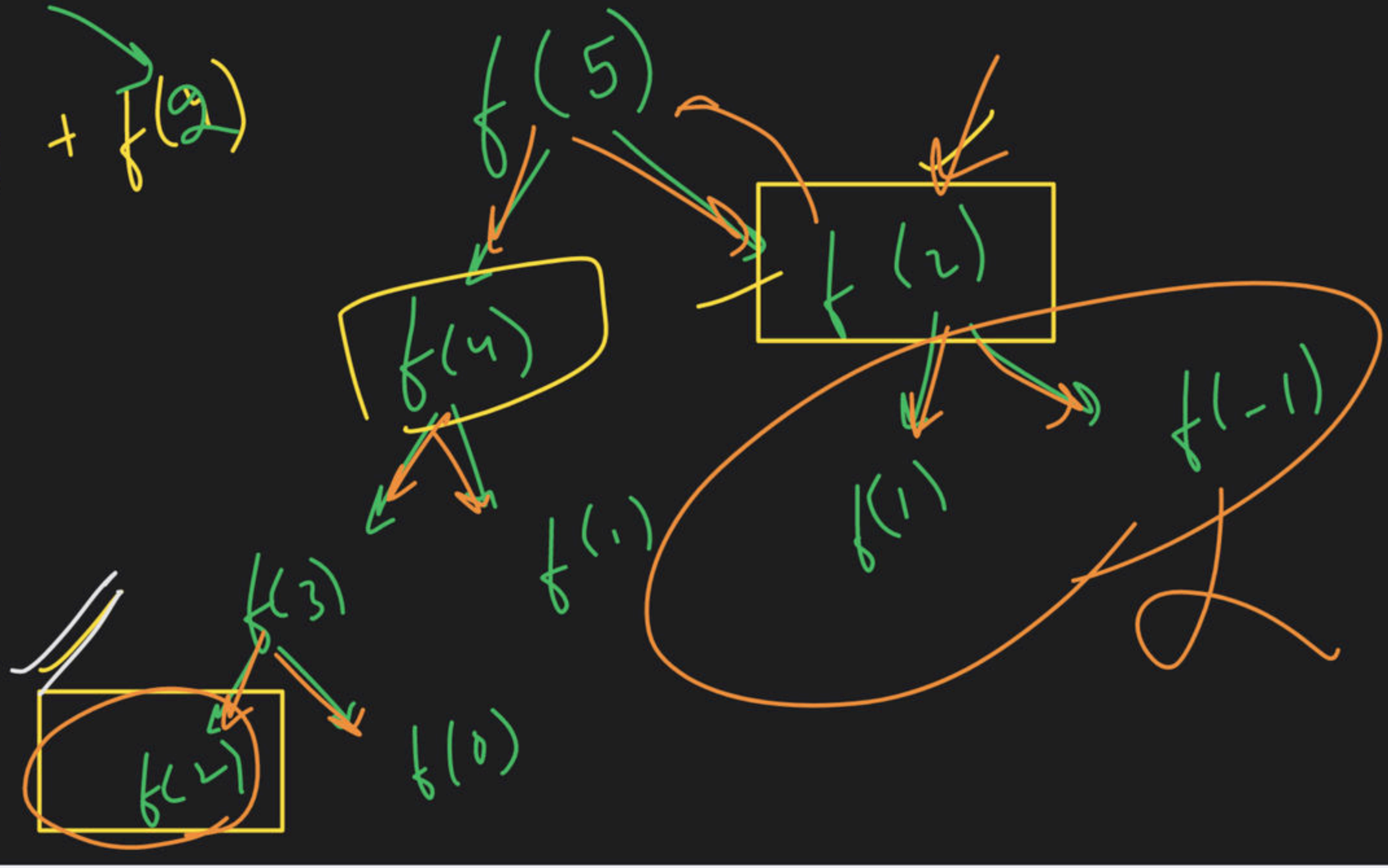
f(3)


f(3)
Big -> optimal Solution
f(3) = f(2) + f(1)
chotti
f(2)
f(1)
f(1)
f(0)
f(0)
f(-1)

R.R ->

$$f(i) = f(i-1) + f(i-3)$$

$$f(5) = f(4) + f(2)$$



DP :-

[those who cannot remember the Past
are condemned to repeat it.]

Types :-

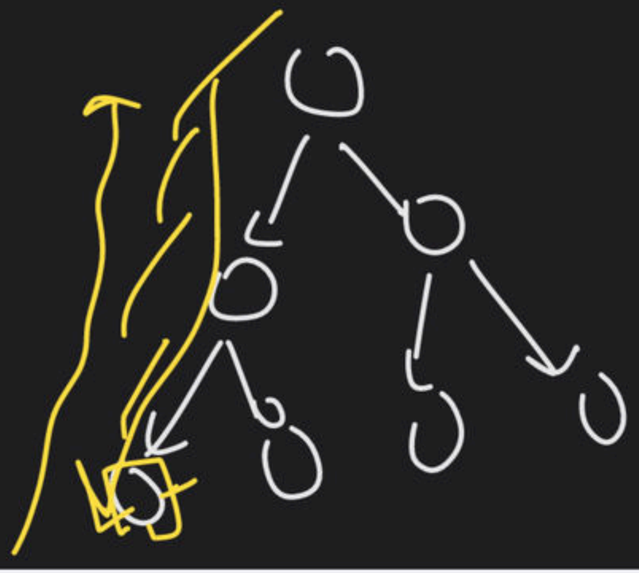
i -> 0 ---> n

-> Top-Down Approach

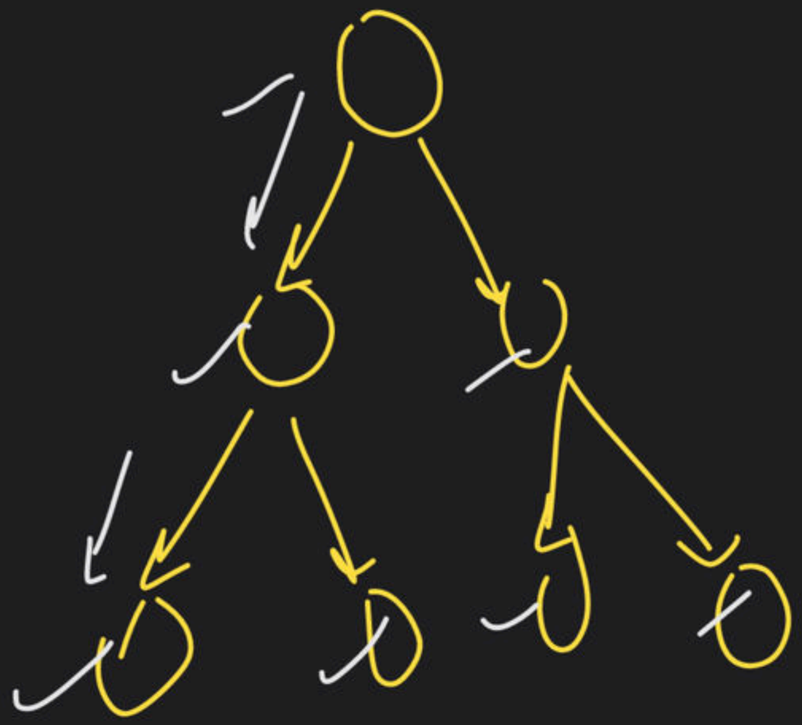
[Recursion + Memoisation]

-> Bottom-Up Approach ---> [Tabulation method]

i -> n ---> 0




Rec
max Sum
Root → Leaf



→ DP Problems:-

n

CodeHelp

↓

DP

Series

① Recursion

Can we apply DP → 1's

1D / 2D / 3D ?

② Recursion + mem

step

step

step

③ Bottom Up Sol^n

step

step

step

④ Space Optimisation

dp arr → dp curr

2D →

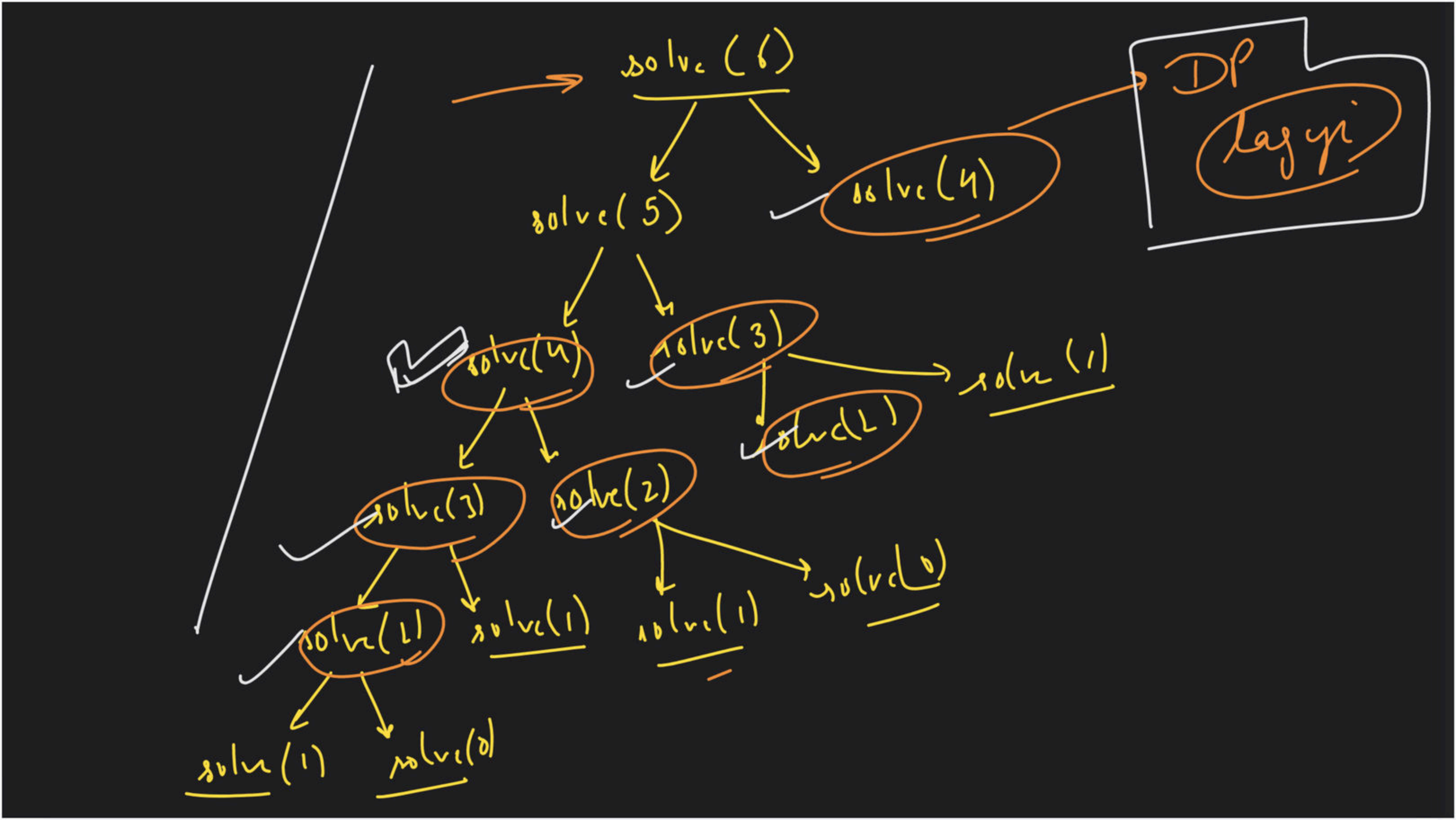
1D DP →

① Fibonacci Series :-

$$f(n) = f(n-1) + f(n-2)$$

R.R

T.C → exponential


solve(6)
solve(5)
solve(4)
DP
lagi
solve(4)
solve(3)
solve(1)
solve(2)
solve(3)
solve(2)
solve(1)
solve(1)
solve(0)
solve(1)
solve(1)
solve(0)

vector<int> dp(n+1, -1)

John Mcan

dp(n)

Recursion → How

+ Mem

(1) create dp array + pass function

1D → vector<int> dp(n, -1)

2D

3D

q

memp

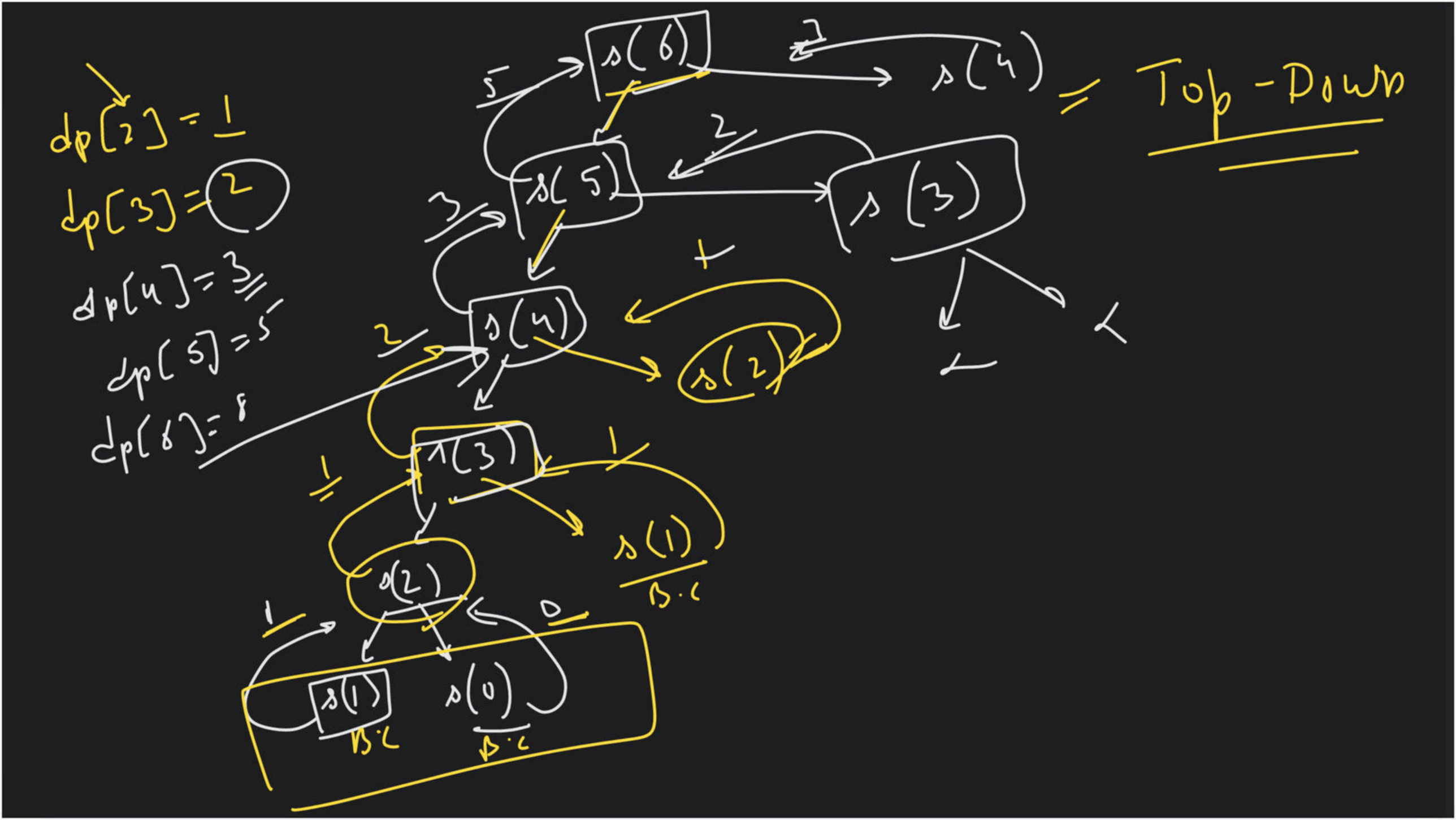
(2) return ans;

return dp(n) = ans;

(3) Base case → Khelna

if (dp(n) != -1)

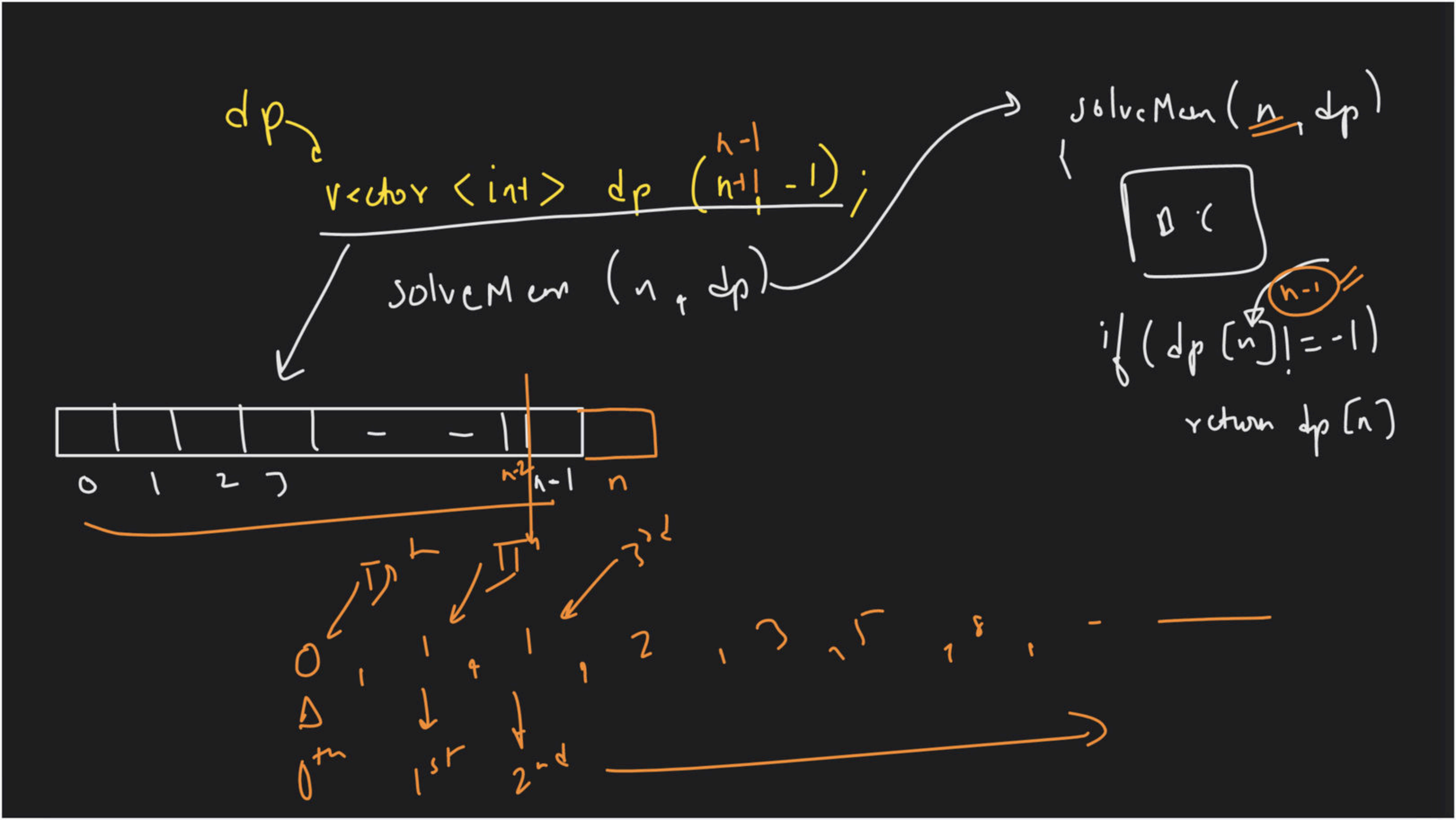
return dp(n);


dp[2] = 1
dp[3] = 2
dp[4] = 3
dp[5] = 5
dp[6] = 8
s(6)
s(4)
s(5)
s(3)
s(4)
s(2)
s(3)
s(1)
s(2)
s(1)
s(0)
B.C
B.C
B.C
= Top-Down

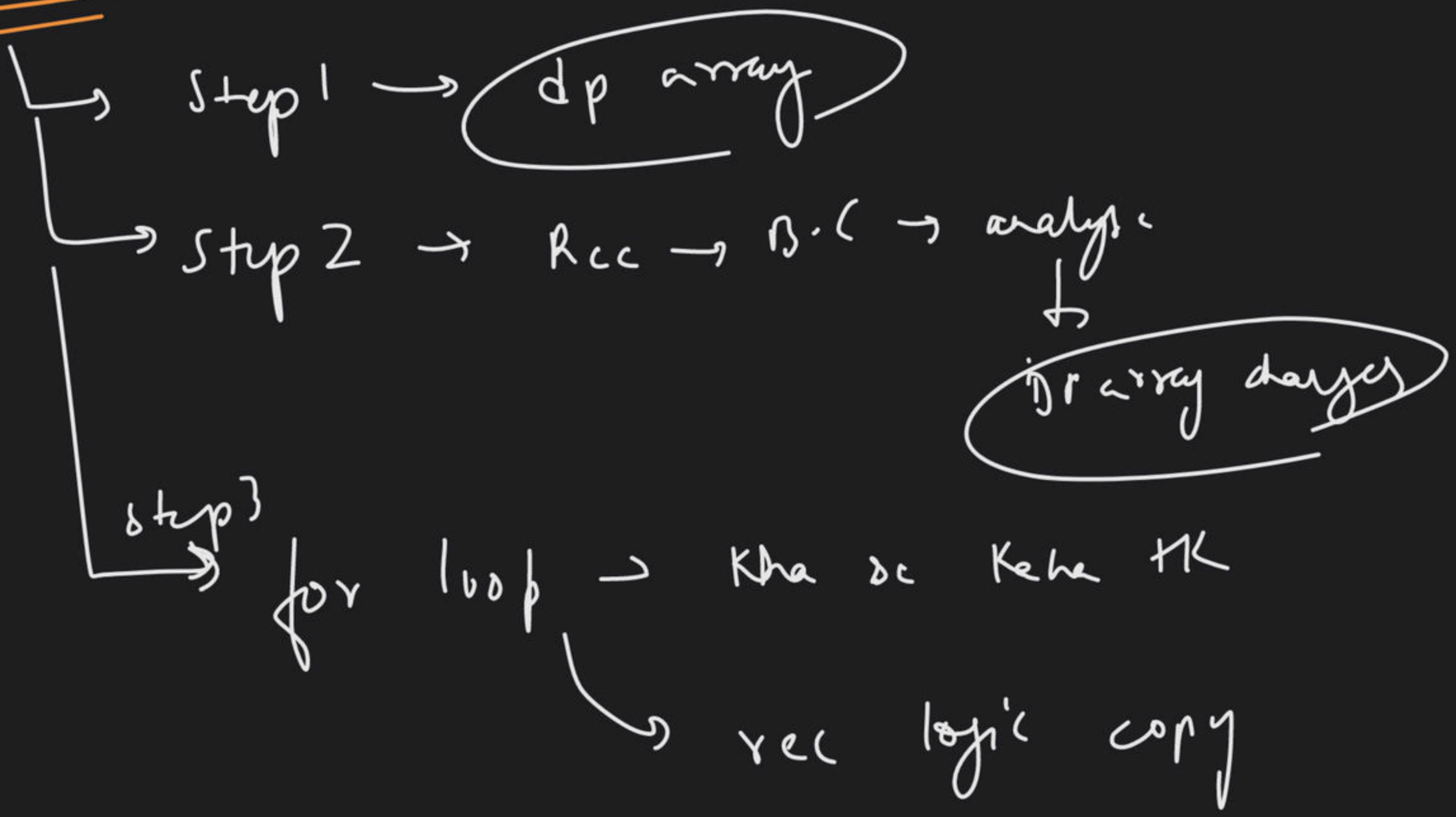
Recursion

```
int solve ( int i ) dp[]
{
    B.C <
    if ( dp[i] != -1)
        return dp[i];
    R.R  dp, dp <
    return ans;   dp[i] = ans;
}
```

1D
R+M

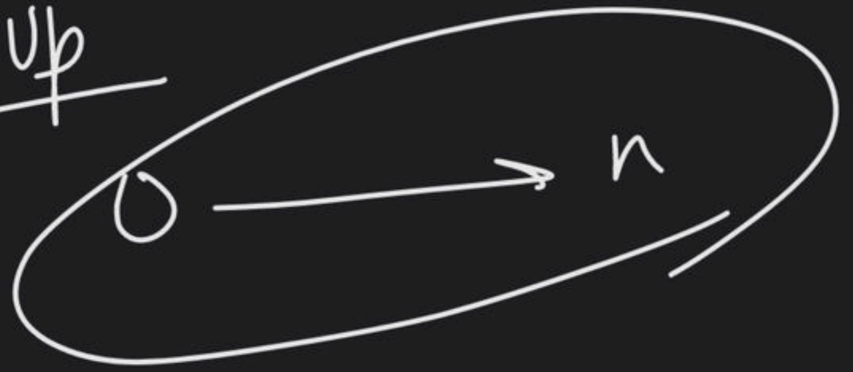

dp
vector<int> dp (n+1, -1);
solveMem(n, dp)
solveMem(n, dp)
{
base case
if(dp[n] != -1)
return dp[n]
n-1
0 1 2 3 n-2 n-1 n
0th 1st 2nd
0 1 1 2 3 5 8

③ Bottom - Up


step 1 → dp array
step 2 → Rec → B.C → analysis
dp array changes
step 3
for loop → Kha se Kaha tk
rec logic copy

Top - Down

Bottom - Up



R → Expo

↓

R+M → O(n) =

→ S.C → O(n) + O(n)

↑ dp ↑ Recursion

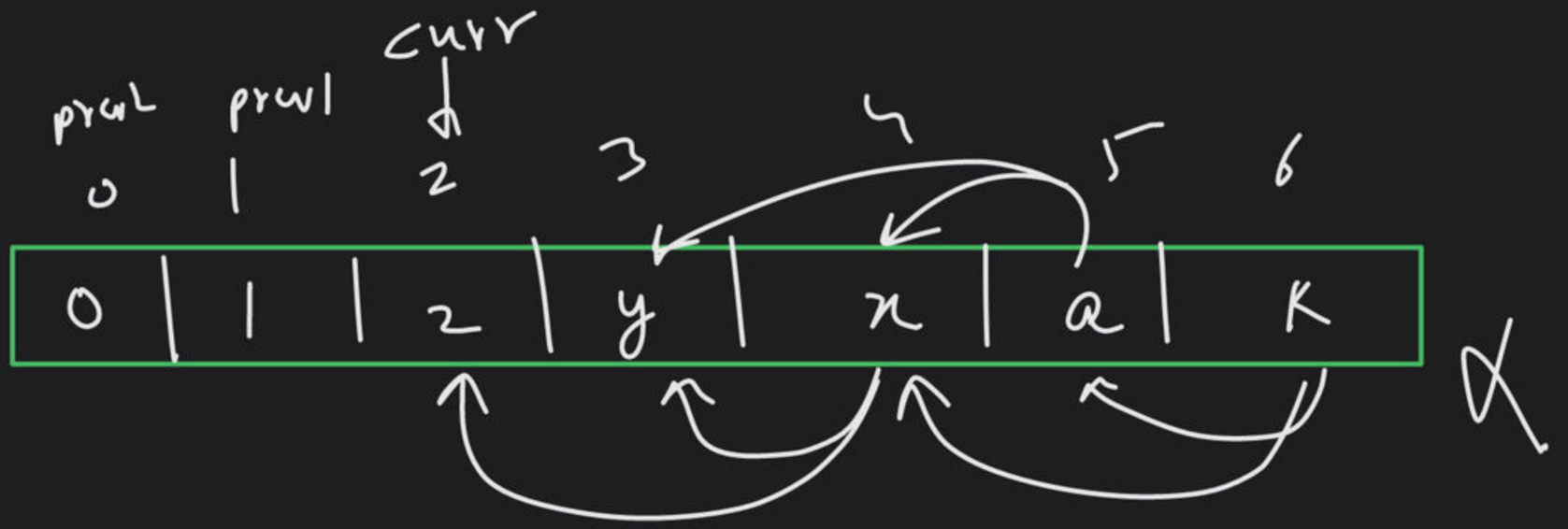
B.U → T.C → O(n) /

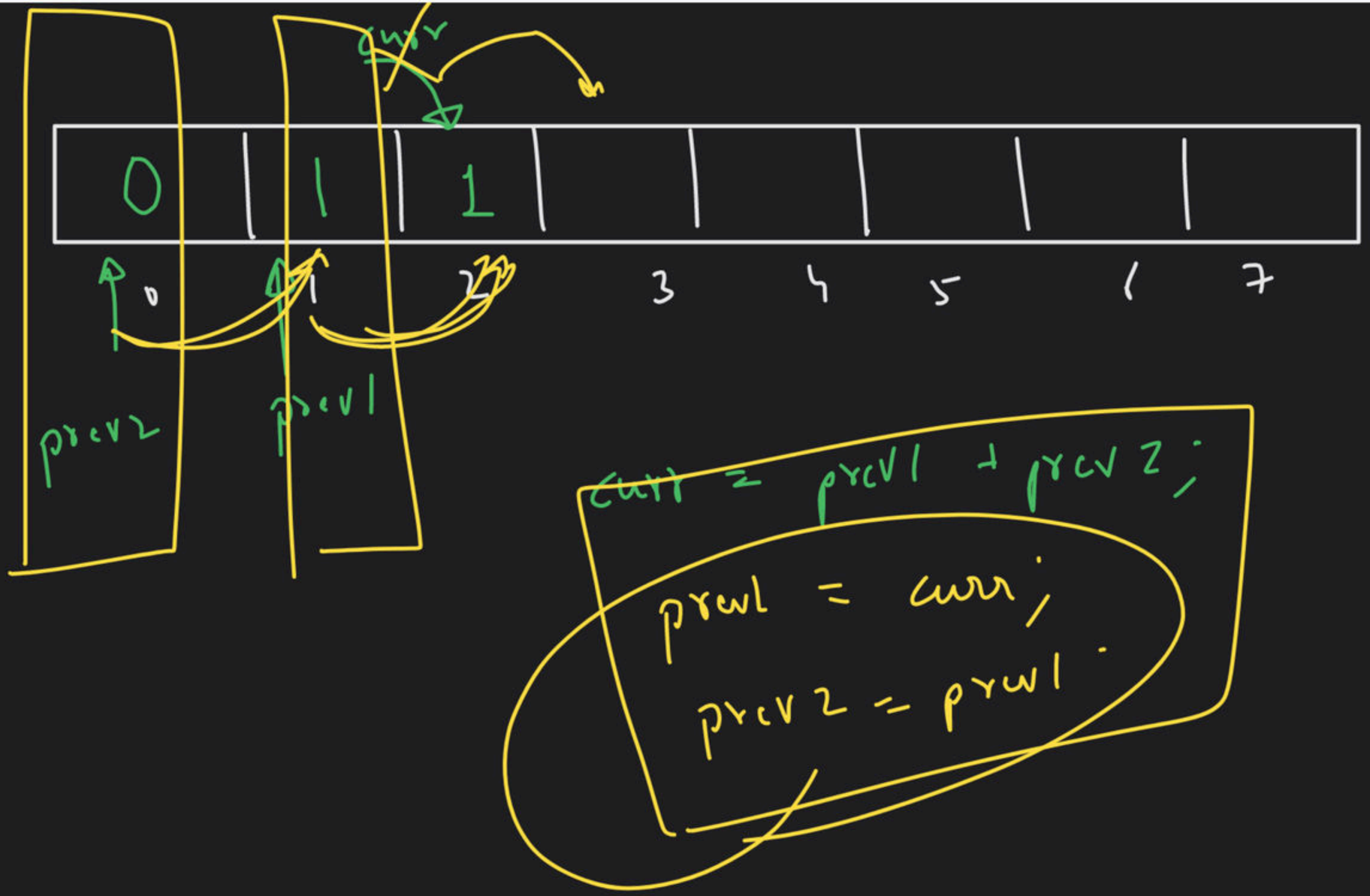
S.C → O(n)

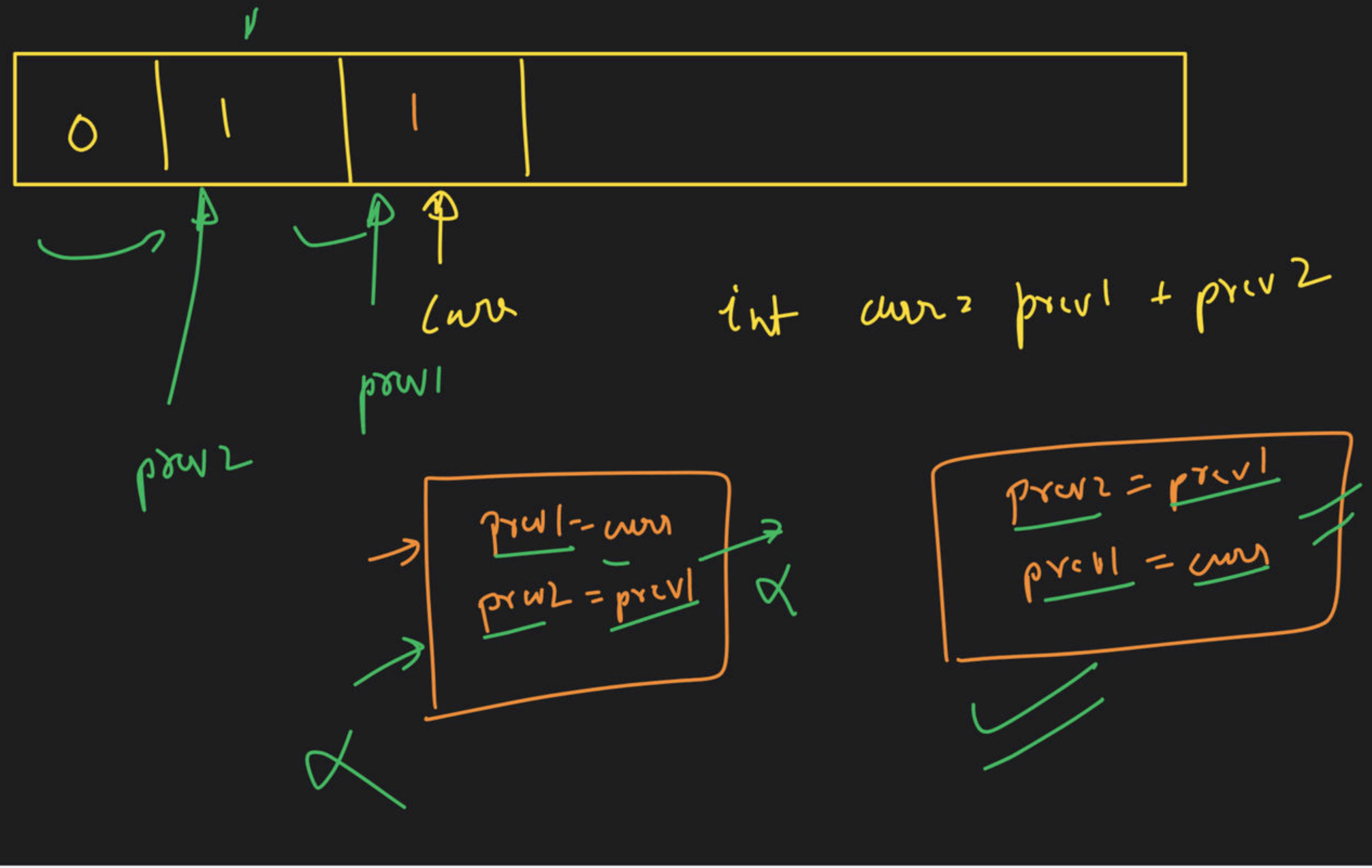
→

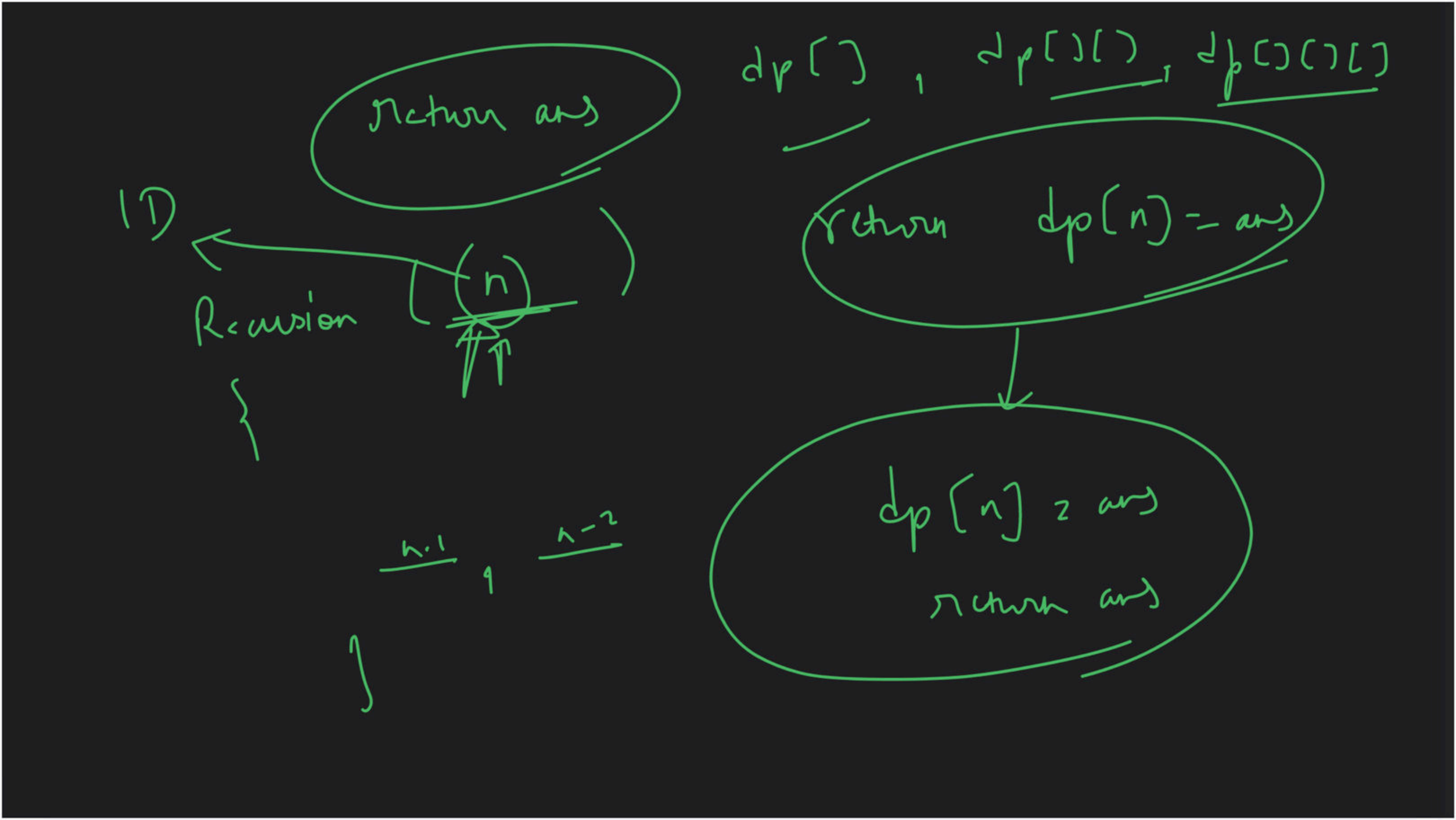
↓

S.O

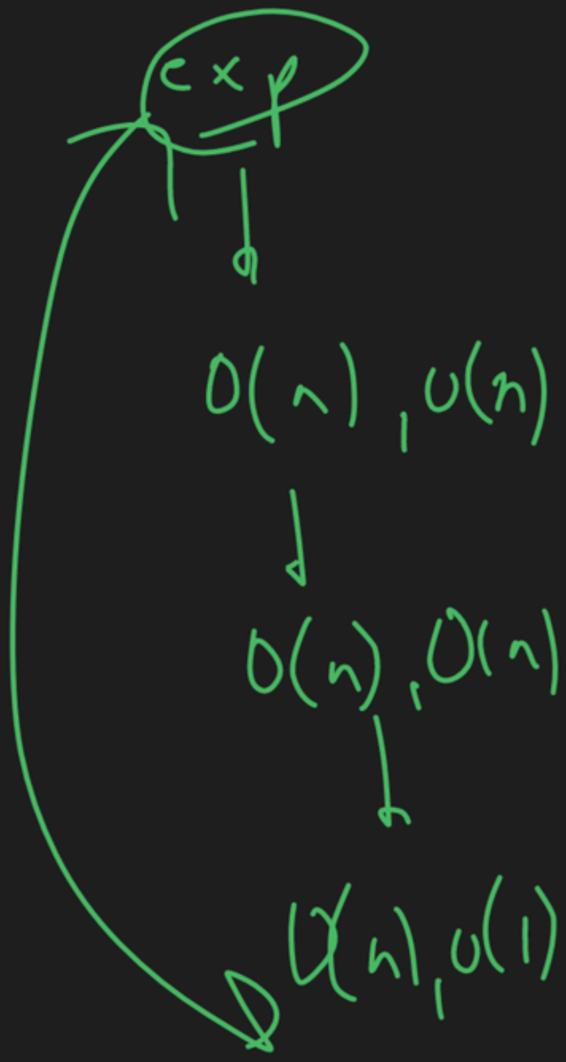

curr
prev2
prev1
0
1
2
3
4
5
6
0
1
2
y
x
a
k

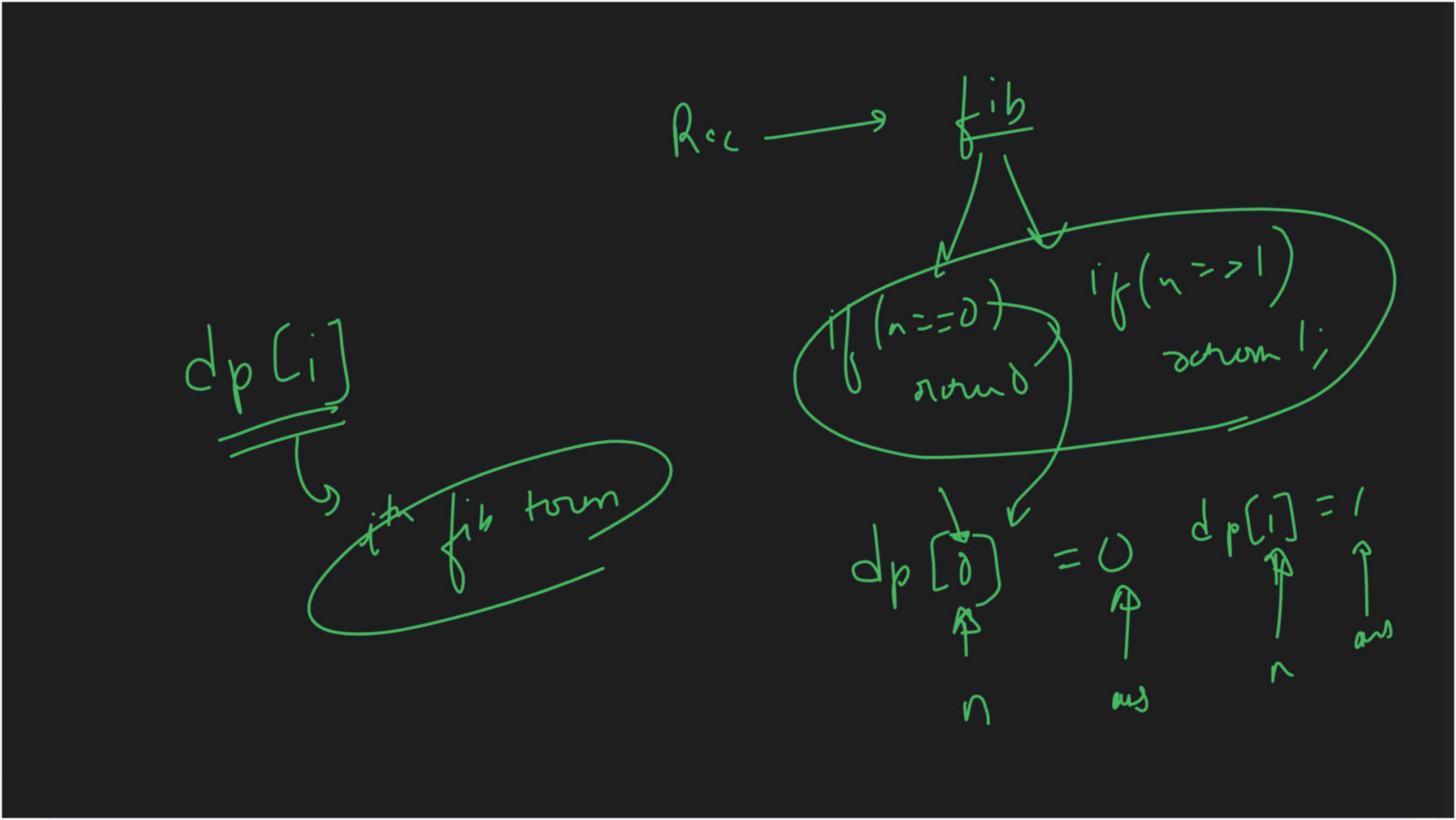

curr
0
1
1
0
1
2
3
4
5
6
7
prev2
prev1
curr = prev1 + prev2;
prev1 = curr;
prev2 = prev1;

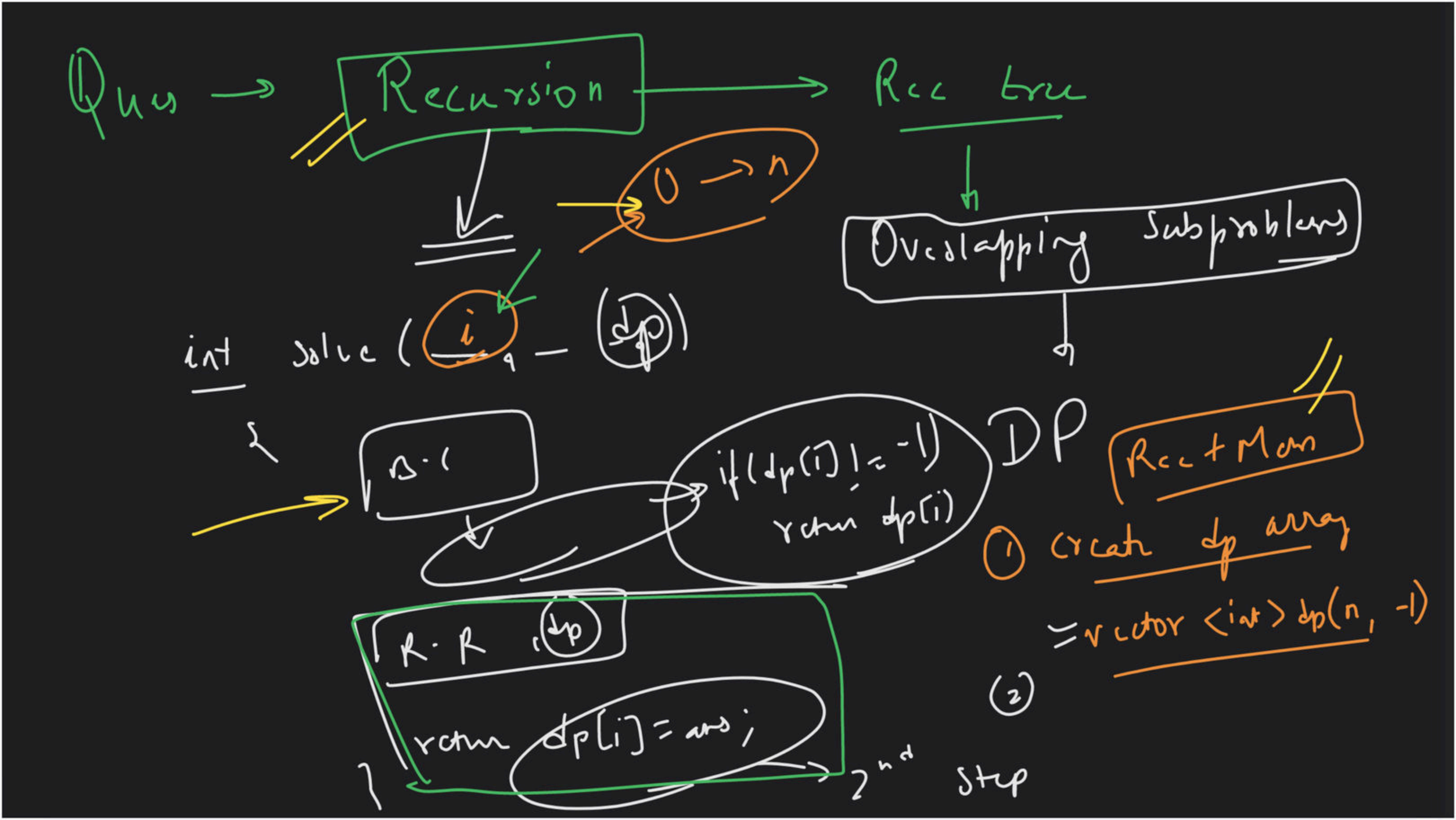

v
0
1
1
curr
prev1
prev2
int curr = prev1 + prev2
prev1 = curr
prev2 = prev1
prev2 = prev1
prev1 = curr

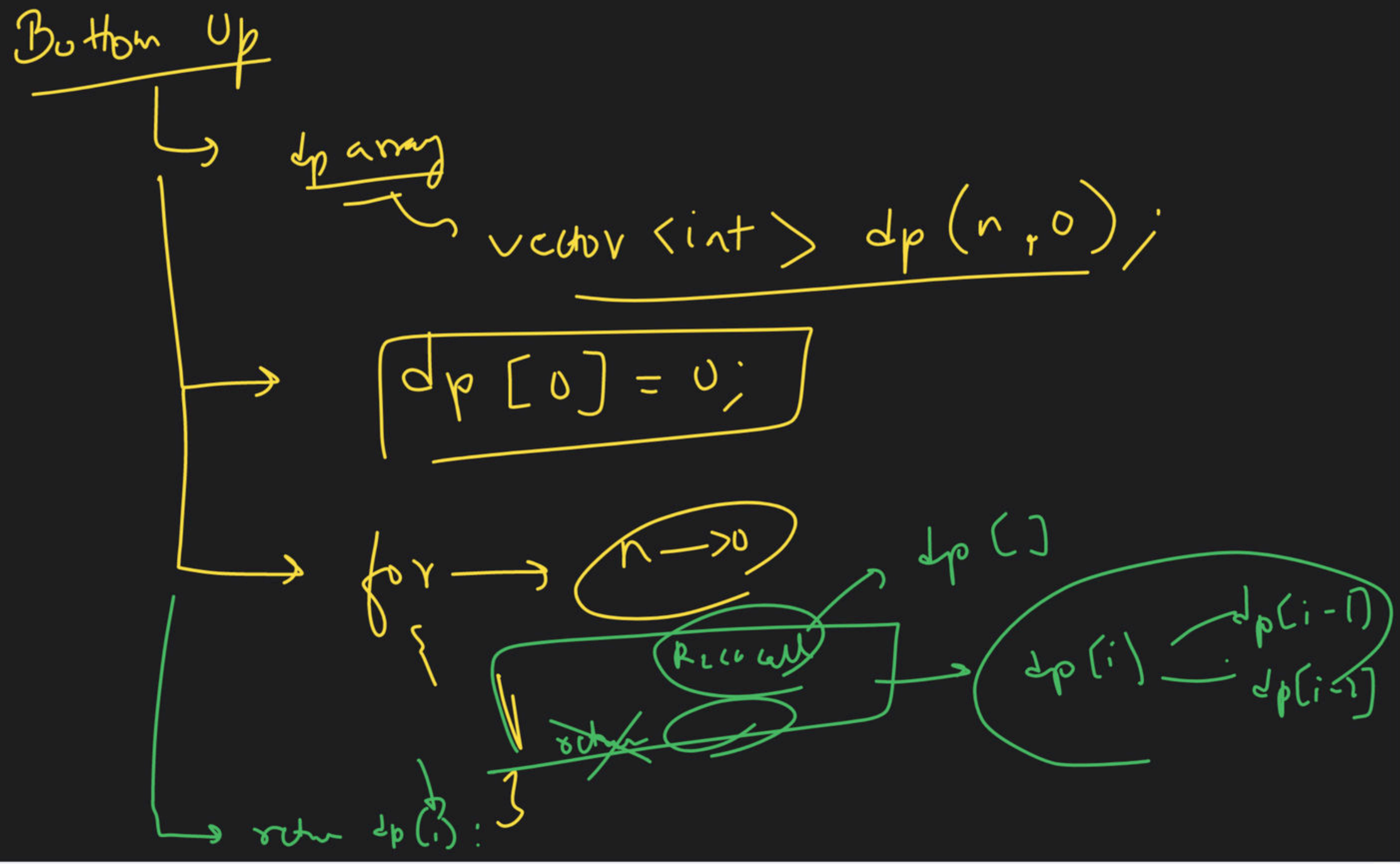

return ans
1D
Recursion
(n)
n-1
n-2
dp[]
dp[][]
dp[][][]
return dp(n) = ans
dp[n] = ans
return ans

Question :-

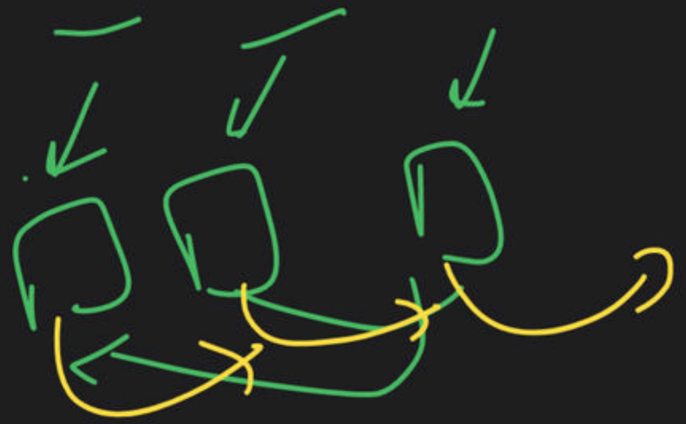



Rec
fib
if(n==0)
return 0
if(n==1)
return 1;
dp[i]
ith fib term
dp[0] = 0
dp[1] = 1
n
ans
n
ans


Ques →
Recursion
→ Rec tree
0 → n
i
int solve (i , dp)
Overlapping Subproblems
DP
Rec + Mem
{
B.C
if(dp[i] != -1)
return dp[i]
① Create dp array
= vector<int> dp(n, -1)
R.R (dp)
return dp[i] = ans;
}
②
2nd Step


Button Up
dp array
vector<int> dp(n,0);
dp[0] = 0;
for
{
n->0
dp[]
Recursive call
dp[i]
dp[i-1]
dp[i-2]
return dp[i];
}

S.O



① dp array

int prev1 =

int prev2 =

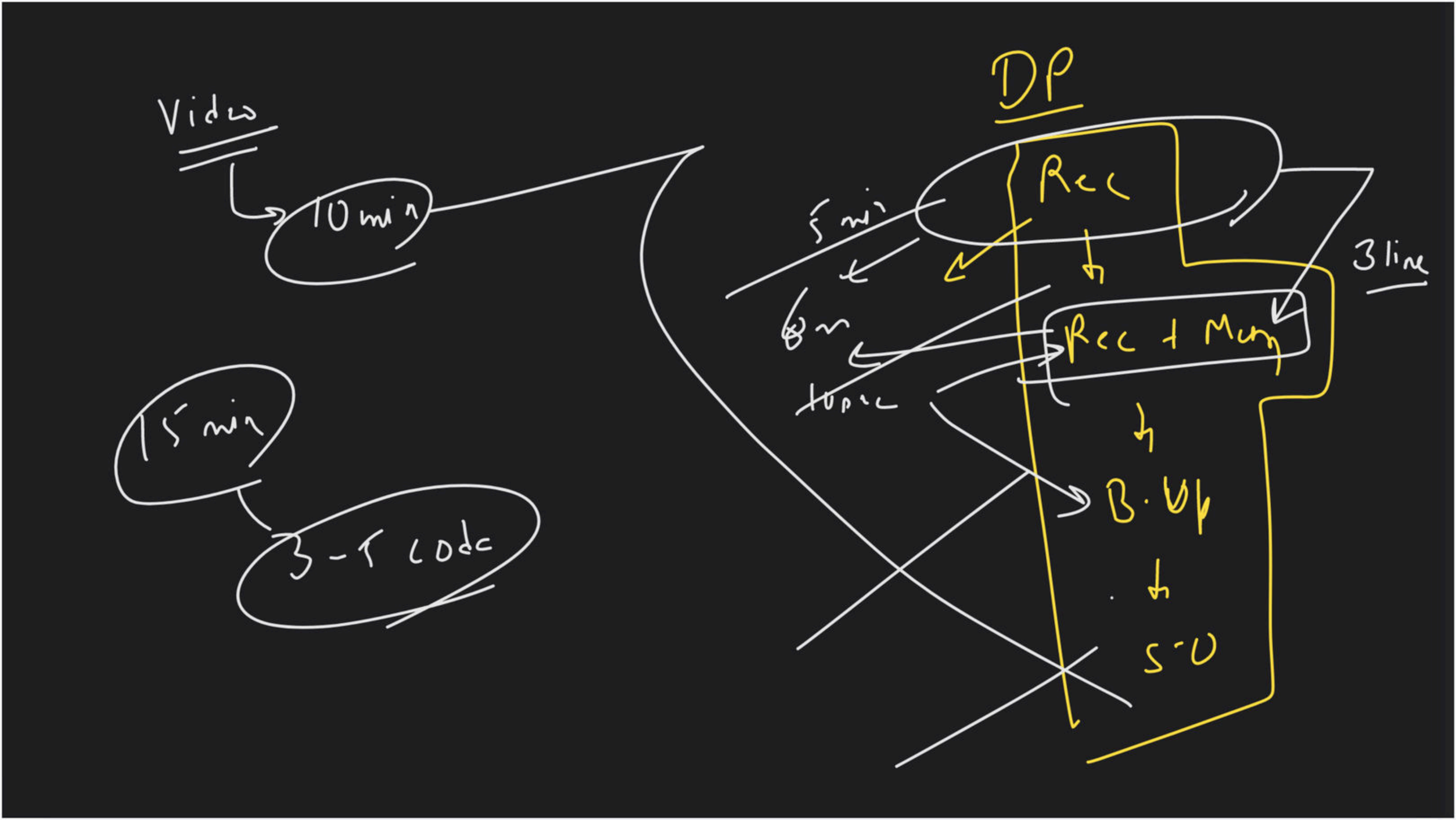
curr

② for ⟶ dp(), dp()

curr, prev1, prev2

③ movement ⟶ 

ry return prev1;


Video
10 min
15 min
3-4 Code
DP
Rec
5 min
3 line
Rec + Mem
10 min
B.Up
S.U

Rec:-

```
int solve(vector<int> nums, int n, vector<int> &dp)
{
    if (n==0)
        return 0;
    if (n<0)
        return INT_MAX;

    if (dp[n] != -1)
        return dp[n];

    int mini = INT_MAX;
    for (int i=0; i<nums.size(); i++)
    {
        int ans = solve(nums, n-nums[i], dp);
        if (ans != INT_MAX)
            mini = min(mini, ans);
    }
    return dp[n] = mini;
}
```

1D or 2D

n → 0

vector<int> dp(n+1, -1)

-1 · P

n →

n → 0

D. U

0 → n

D. C

Wildcard Pattern Matching

H/W → Dr

int solveTab (___)

{

① vector<int> dp(n+1, INT_MAX);

② dp[0] = 0;

③ for (int i = 1; i <= n; i++)

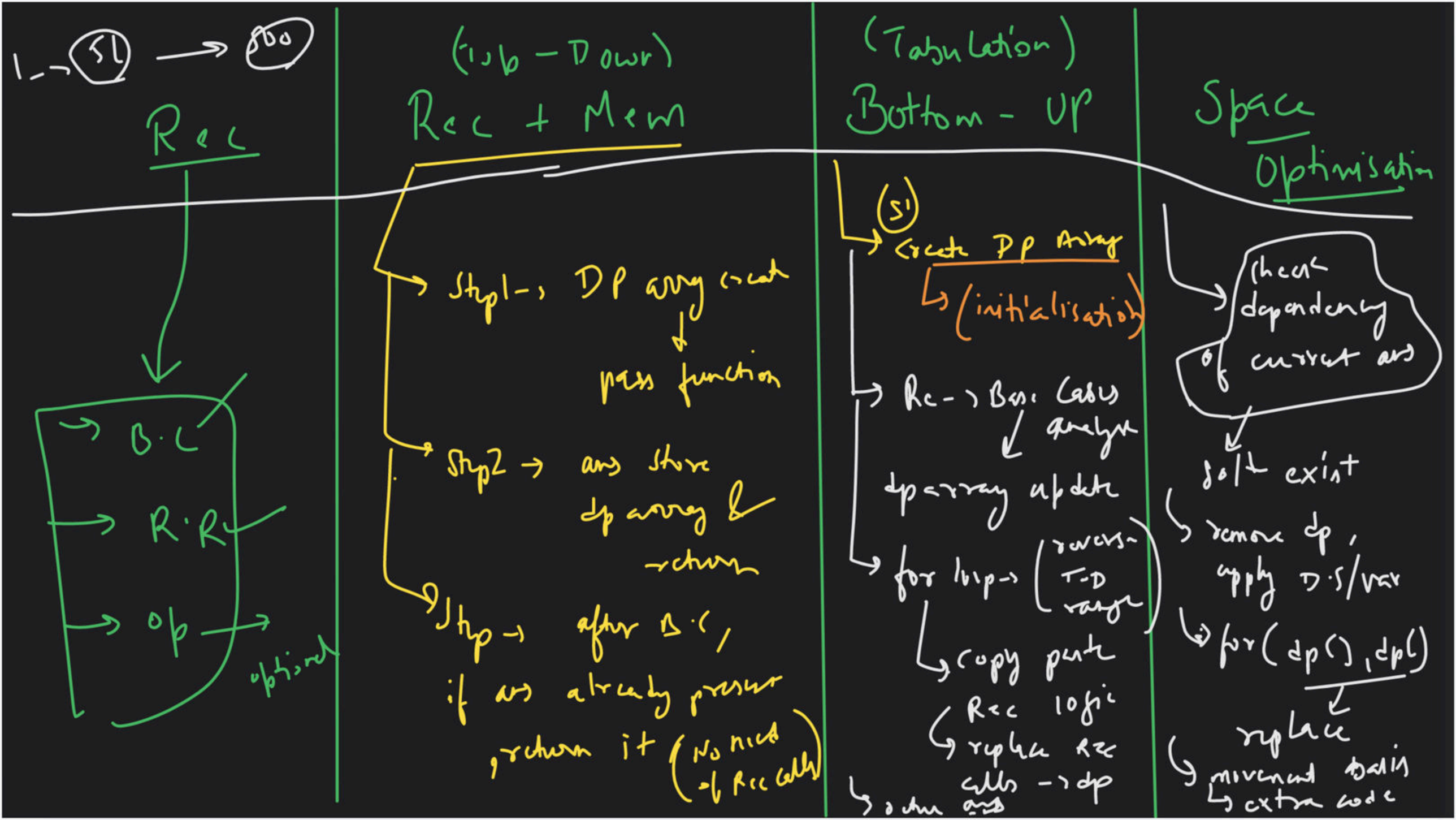
{

paste solve(nums, n-nums[i], dp)

dp[n-nums[i]]

}

④ return dp[n];


Rec
(Top-Down)
Rec + Mem
(Tabulation)
Bottom - UP
Space Optimisation
B.C
R.R
Op
optional
Step1 -> DP array create
pass function
Step2 -> ans store
dp array &
return
Step3 -> after B.C,
if ans already present
return it (No need of Rec calls)
Create DP Array
(initialisation)
Rec -> Base Cases
analyse
dp array update
for loop -> (reverse T-D range)
copy paste
Rec logic
replace Rec calls -> dp
check dependency of current ans
sol exist
remove dp,
apply B.S/var
for(dp(), dp())
replace
movement codes
extra code