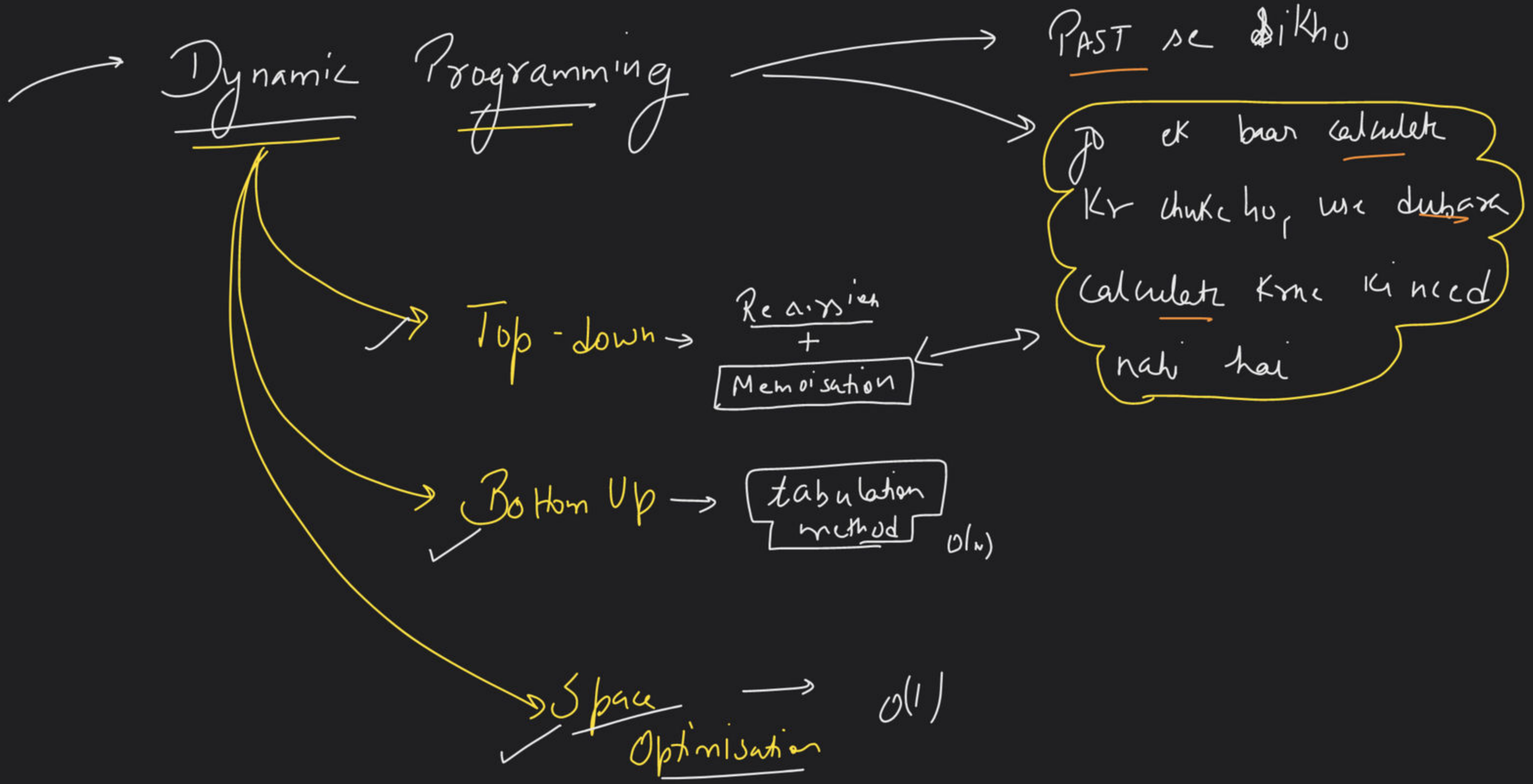
Dynamic Programming Class - 1

Special class

Love Babbar • June 3, 2023


Dynamic Programming
PAST se sikho
Jo ek baar calculate Kr chuke ho, use dubara calculate Krne ki need nahi hai
Top - down → Recursion + Memoisation
Bottom Up → tabulation method O(N)
Space Optimisation → O(1)

When to apply?

Overlapping Sub problems

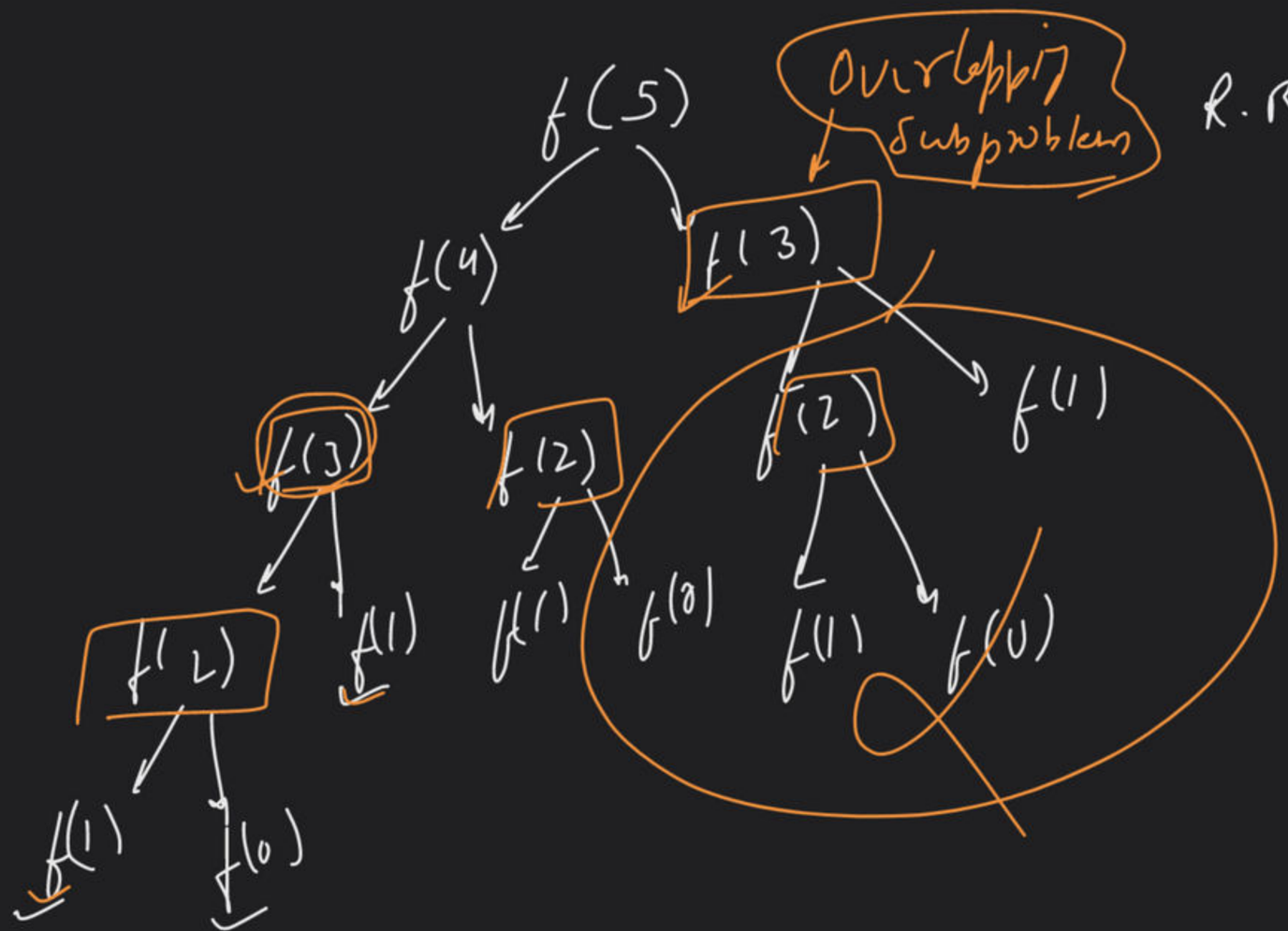
Same problem ko baar baar solve krre ho

Optimal Substructure

Optimal Sol^{n} of Bigger Problem can be calculated using Optimal sol^{n} of smaller problem

Fibonacci :-

0, 1, 1, 2, 3, 5, 8, 13, 21 - -



R.R →

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

$f(1) = 1$

$f(0) = 0$

← Base Case

Rec:-

+ Memoisation

$0 \rightarrow n$ $\quad n \rightarrow 0$ $\quad n+1$

```
int fib ( int n)
{
    // B.C
    if (n == 1 || n == 0)
        return n;
    if (dp[n] != -1)
        return dp[n];
    dp[n] = fib(n-1) + fib(n-2)
    return dp[n];
}
```

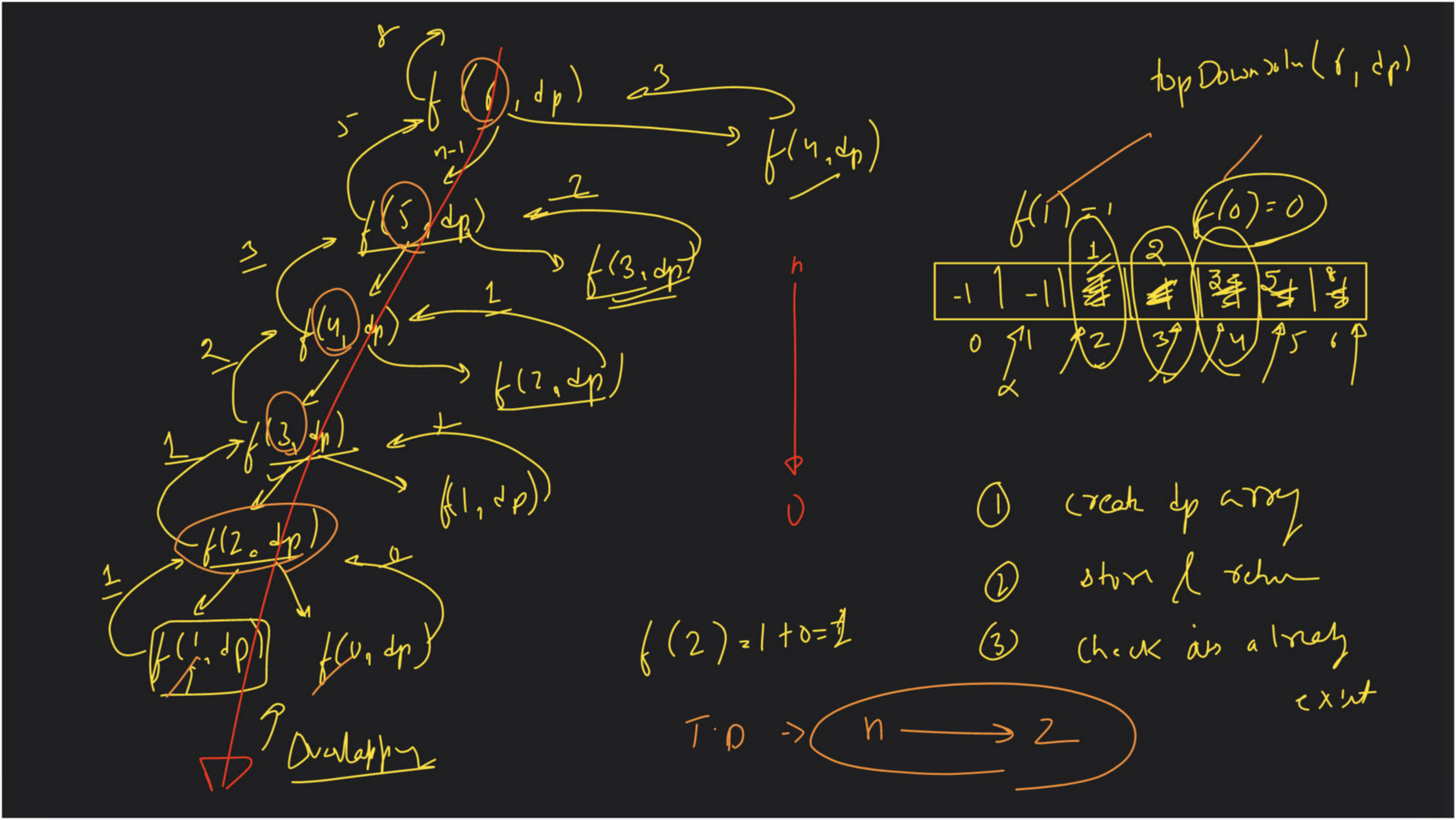
① Create dp array

$\boxed{\text{vector <int> dp (n+1, -1);}}$

② ans store dp

③ B.C & just based

check if answer already exists

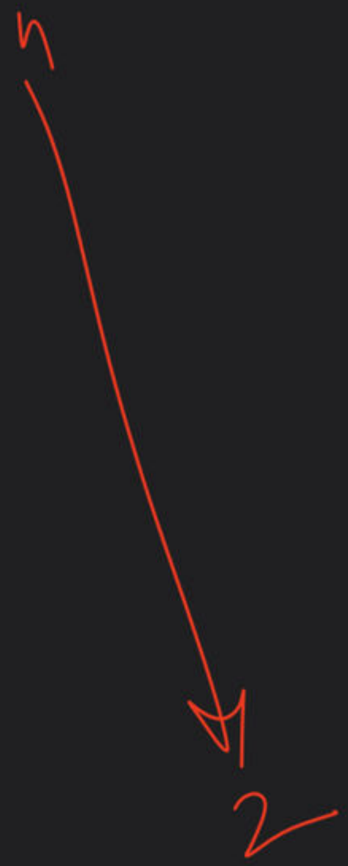

f(6, dp)
f(5, dp)
f(4, dp)
f(3, dp)
f(2, dp)
f(1, dp)
f(0, dp)
Overlapping
n
0
f(2) = 1 + 0 = 1
T.D -> n → 2
topDownSolve(6, dp)
f(1) = -1
f(0) = 0
-1 | -1 | 1 | 2 | 3 | 5 | 8
0 1 2 3 4 5 6
① create dp array
② store & return
③ check if already exist

top down

$n \to 2$

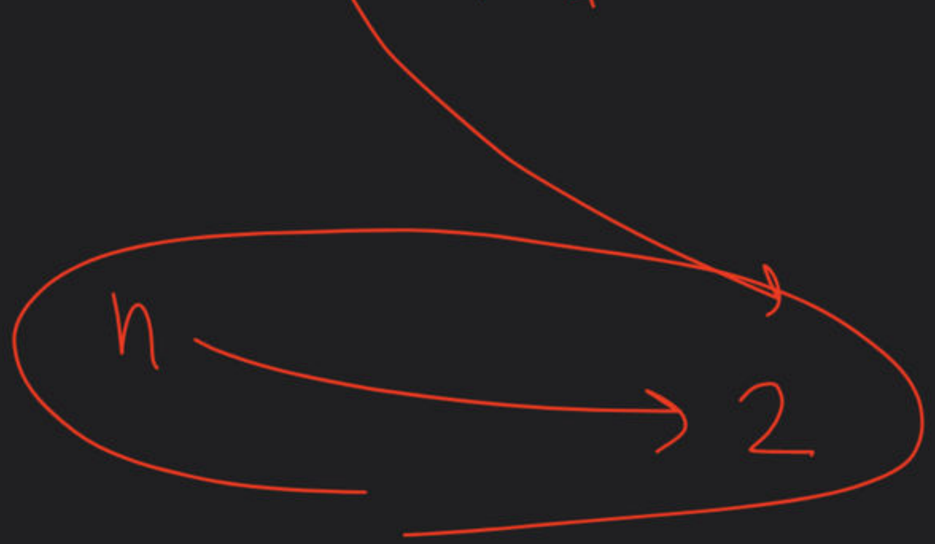
$2 \to n$

top down



$n \to n-1, n-2$

$n-1 \to n-2, n-3$



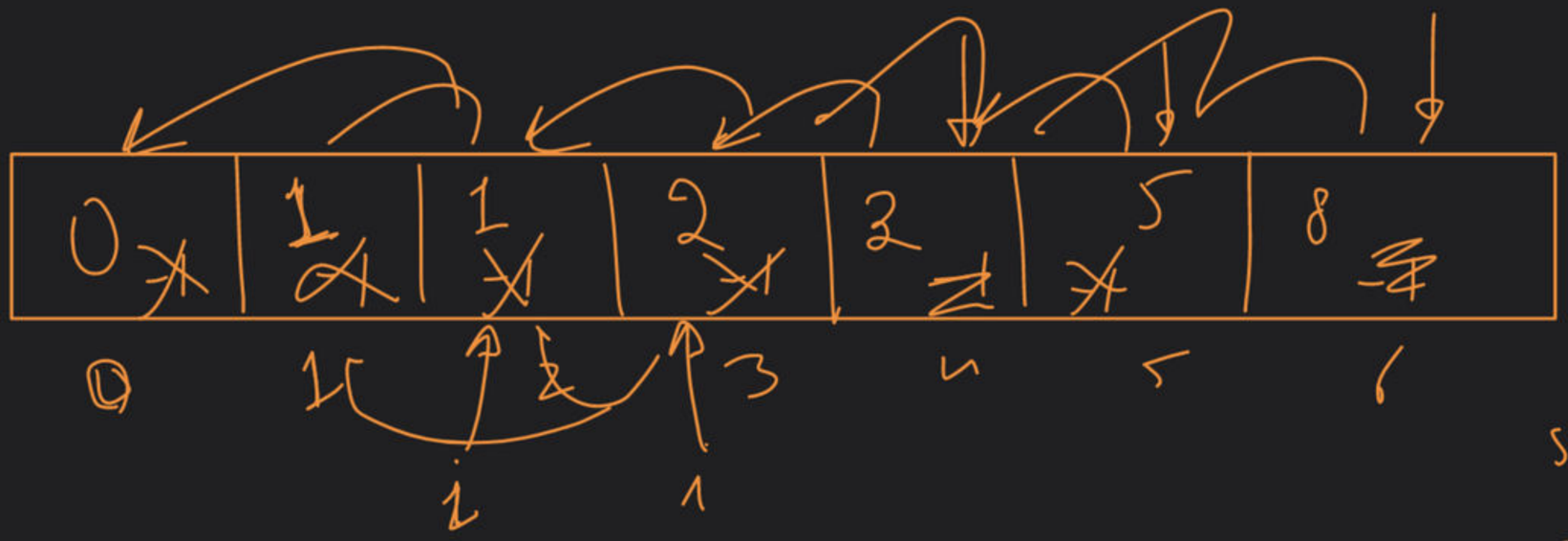
$n-2 \to n-3, n-4$

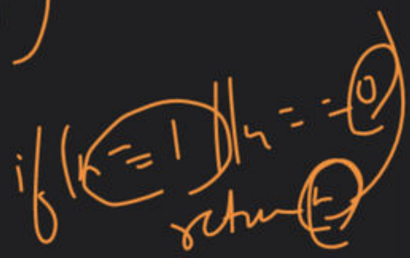
$n \to 2$

1, 0

B.C

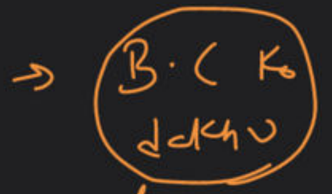
Bottom-Up [Tabulation method]





Step 1 →

Step 2 → B.C Ko dekho



accordingly value fill karo

Step 3:

2 → n

for(i=2 : i<=n)

{

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

}

return dp[n];

i/p → n = 0

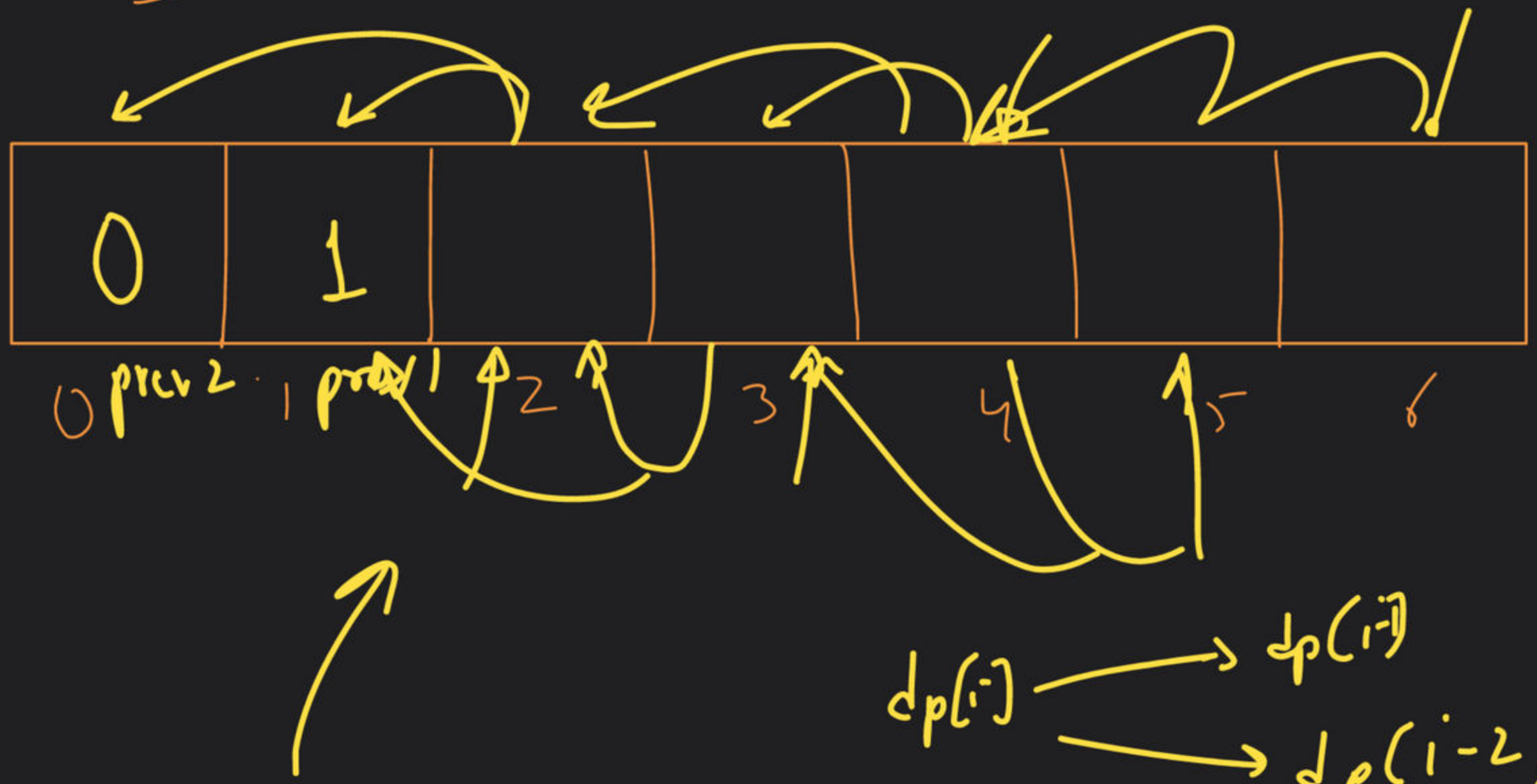
step 1 → dp array (n+2) → 1 size

step dp[0] = 0

dp[1] = 1

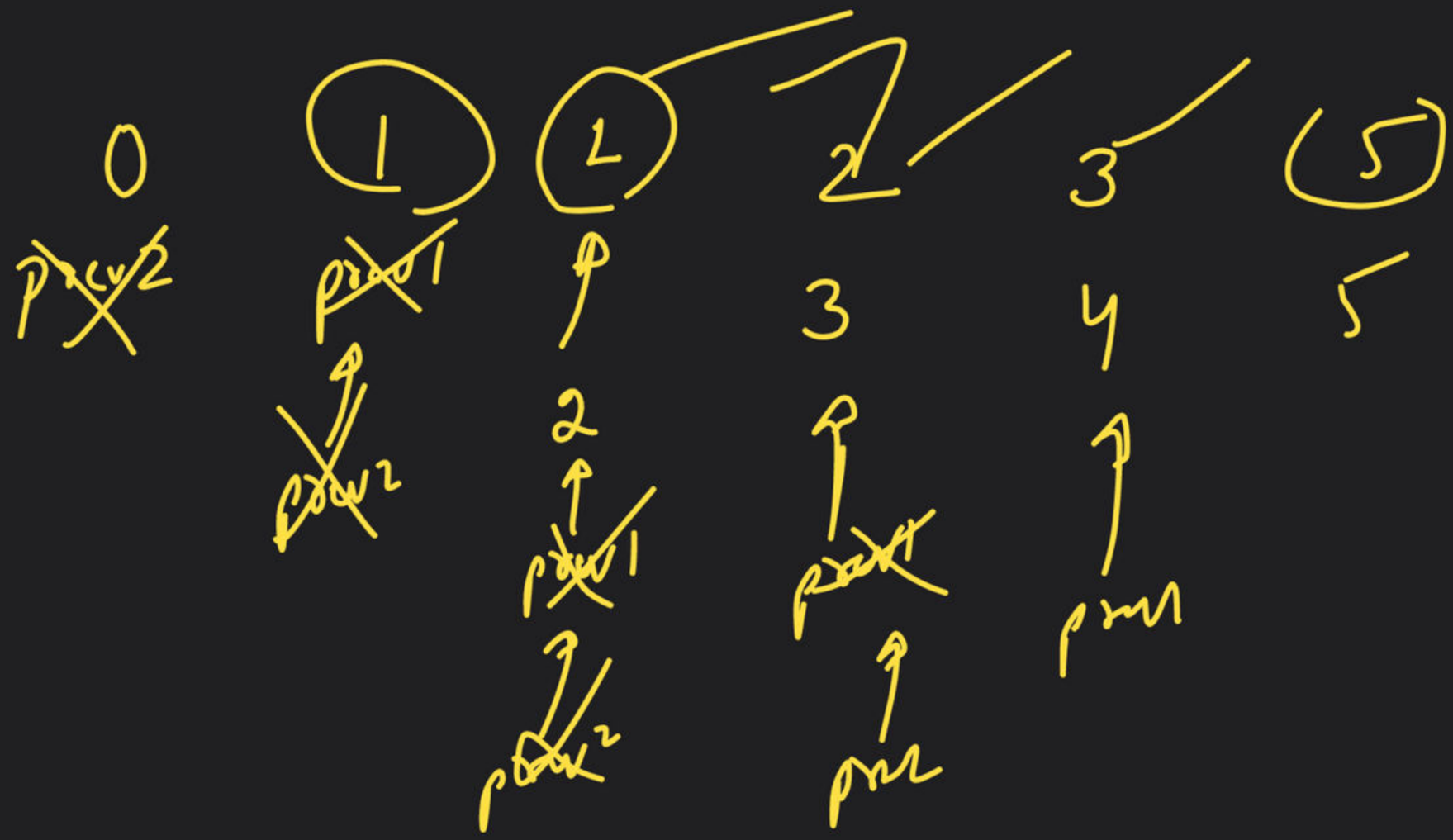
iski wajah ie fatra tha

Space Optimisation:-

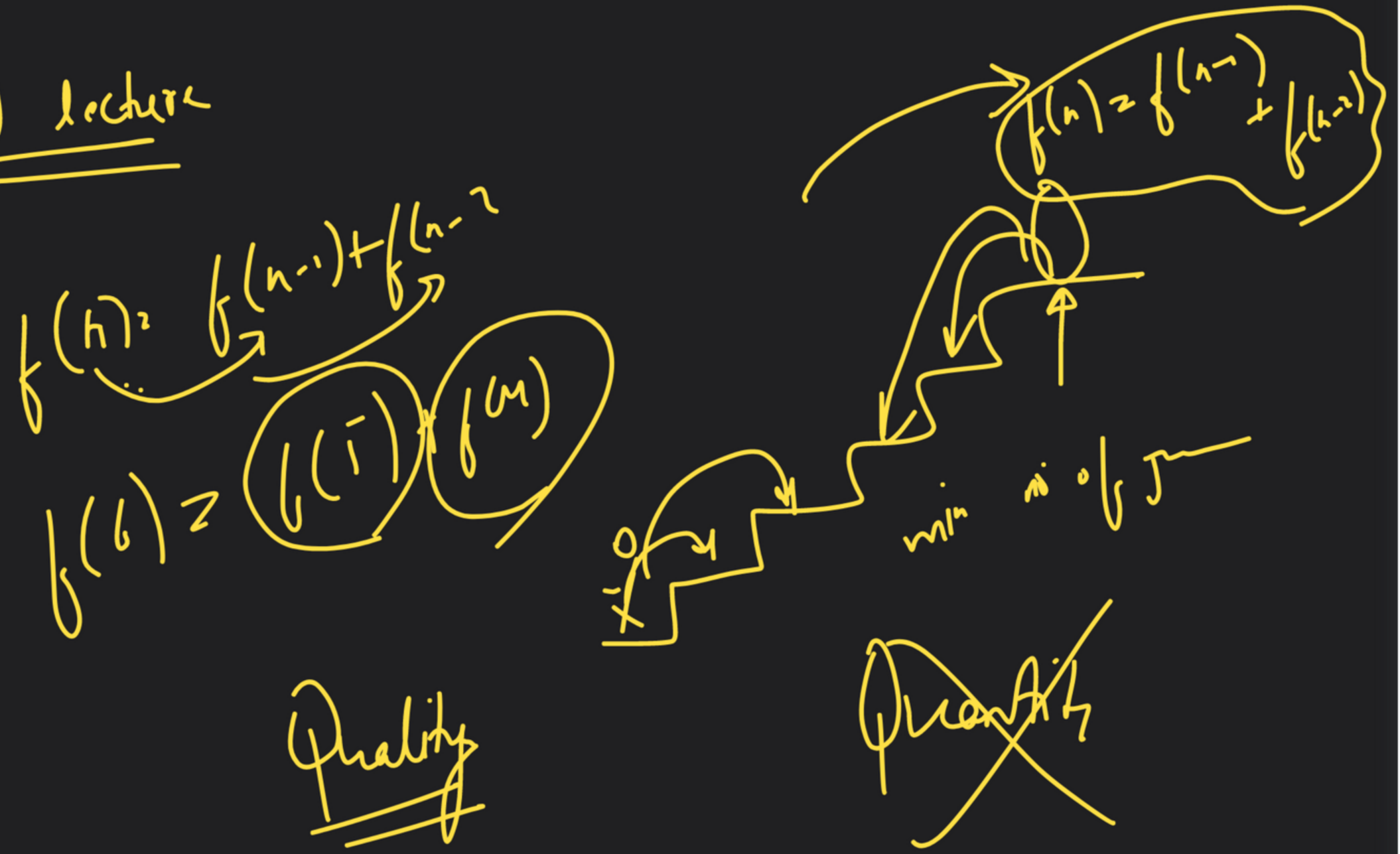


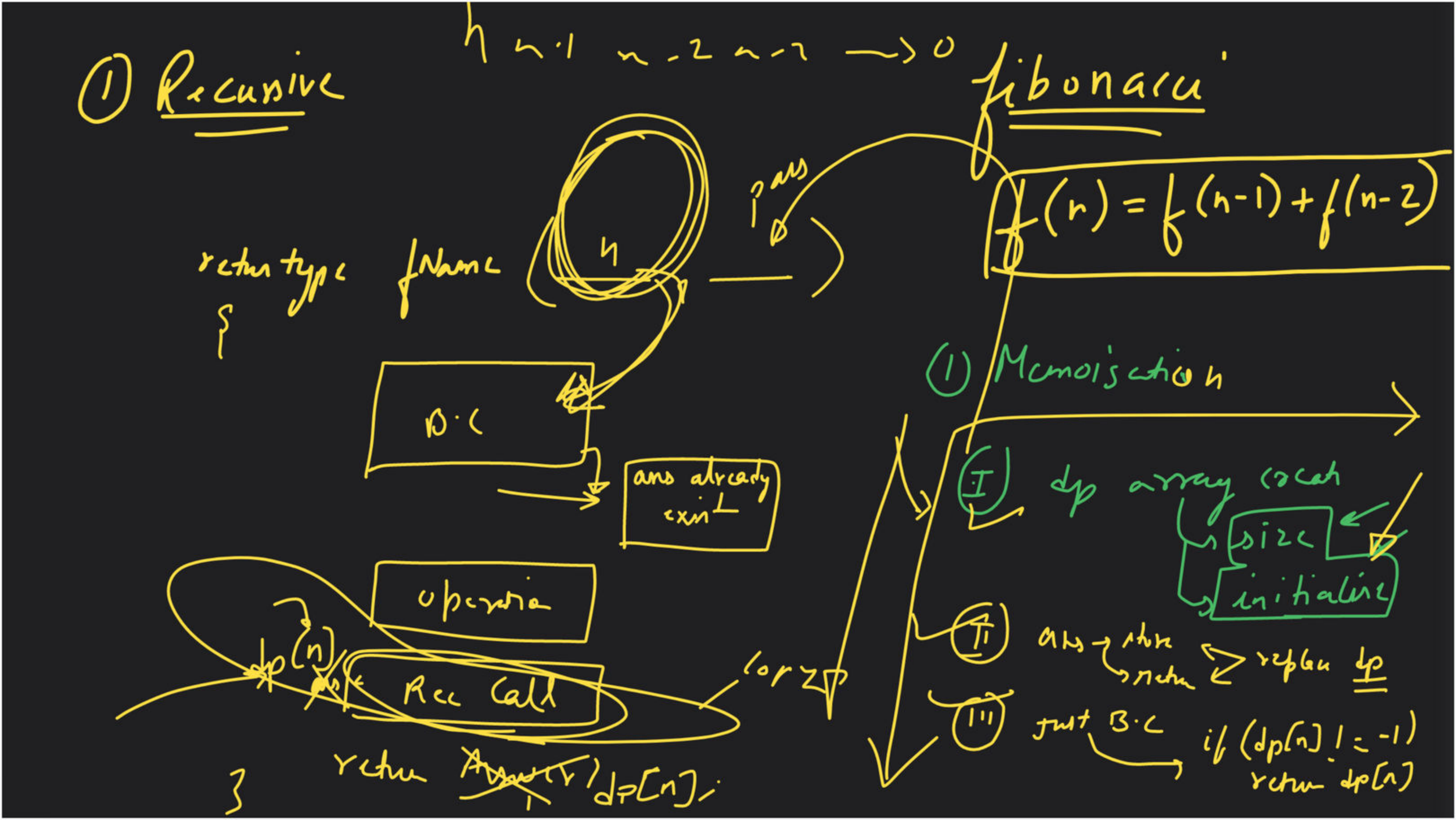
dp[i] → dp(i-1)

dp[i] → dp(i-2)


0
1
2
2
3
5
3
4
5
2
null
null
null
null

Kal lecture





① Recursive

$n \quad n-1 \quad n-2 \quad n-2 \longrightarrow 0$

fibonacci

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

return type fName (n) pass

{

B.C

ans already exist

operation

Rec Call

dp[n]

return ~~Answer~~ dp[n];

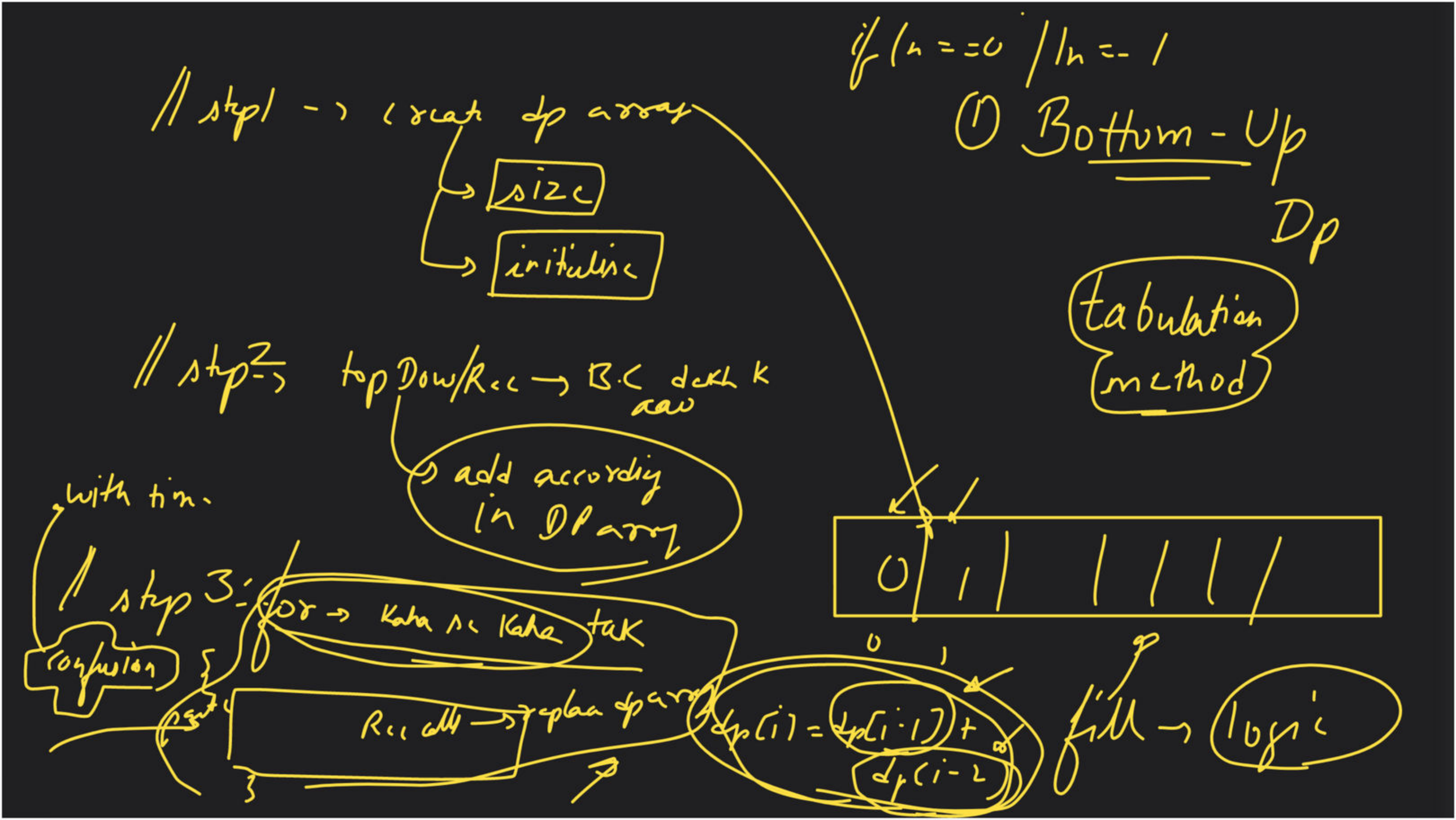
}

① Memoisation

(I) dp array create
- size
- initialise

(II) ans → store → return ⇄ replace dp

(III) just B.C → if (dp[n] != -1) return dp[n]



if ln == 0 / ln == 1

① Bottom-Up Dp

tabulation method

// step1 → create dp array

→ size

→ initialise

// step2 → top Dow/Rec → B.C dekh k aao

→ add according in DP array

with time

// step 3: for → Kaha se Kaha tak

confusion

{

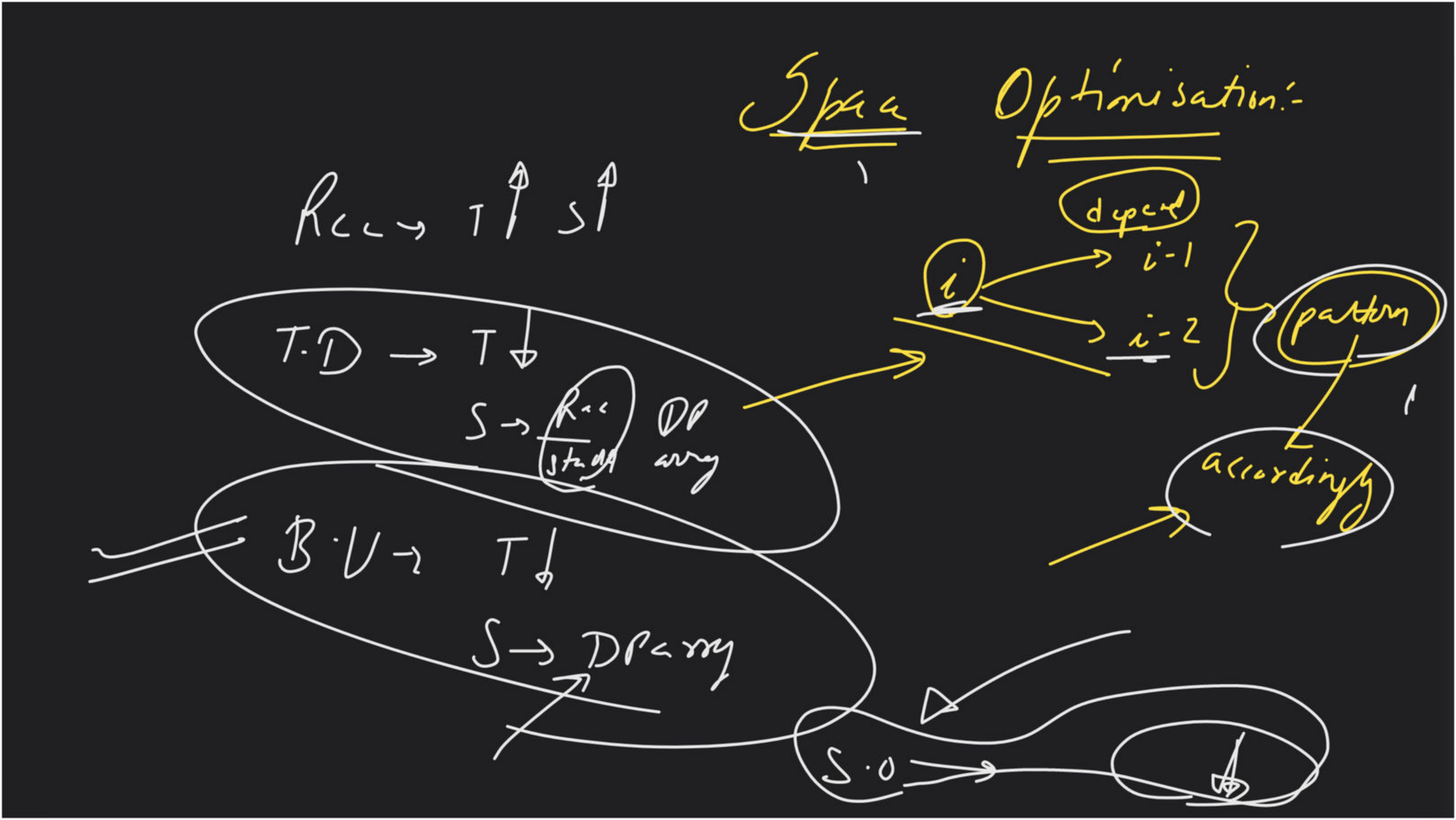
Rec call → replace dp array

}

0	1					
0	1					∞

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

fill → logic



Space Optimisation:-

Rec → T↑ S↑

T.D → T↓

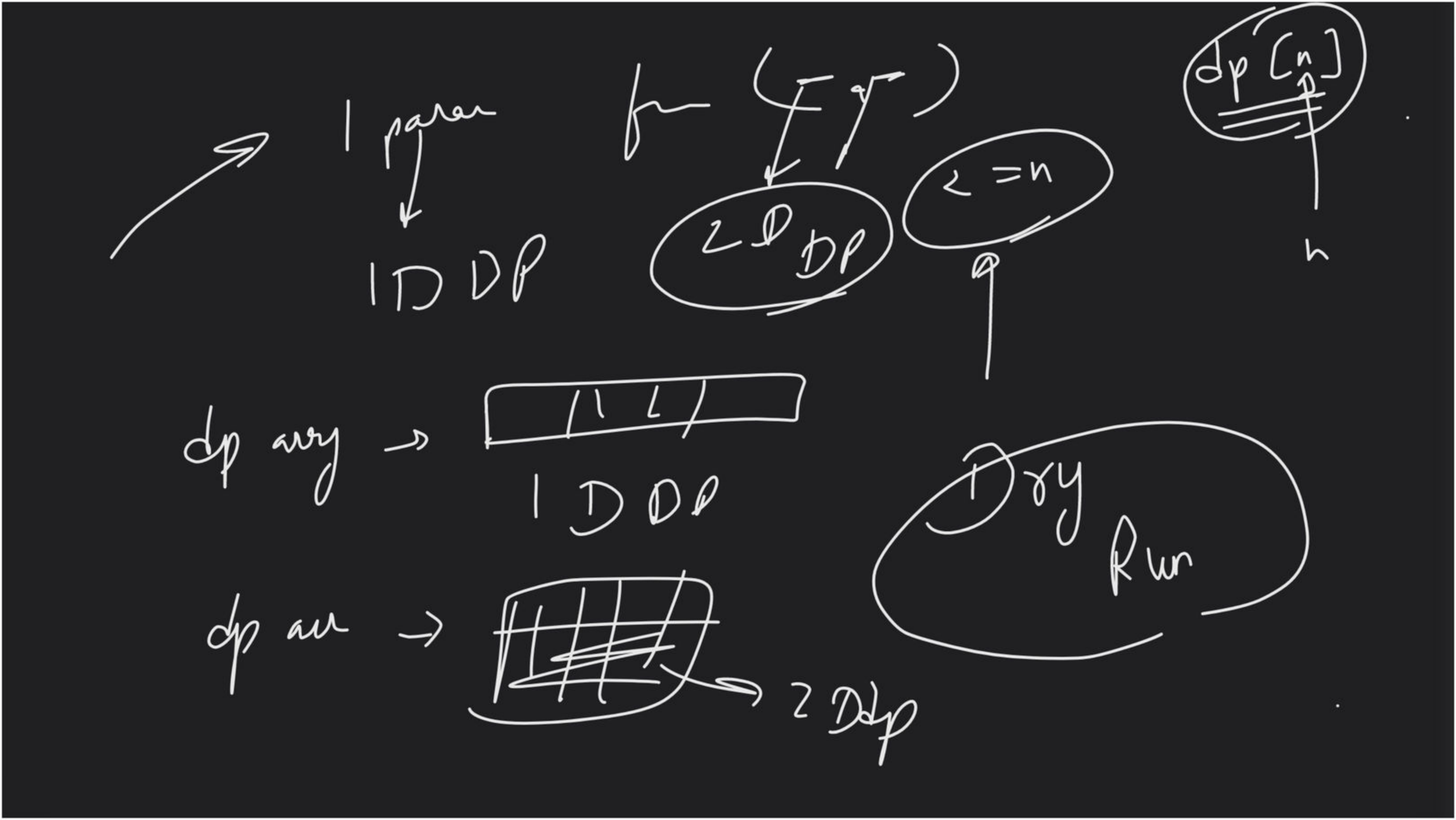
S → Rec / stack DP array

B.U → T↓

S → DP array

S.O → ↓

i → depend i-1, i-2 → pattern → accordingly



1 param → 1D DP

from (Eg) → 2D DP

2 = n

dp[n] → n

dp arry → 1D DP

dp arr → 2D dp

Dry Run