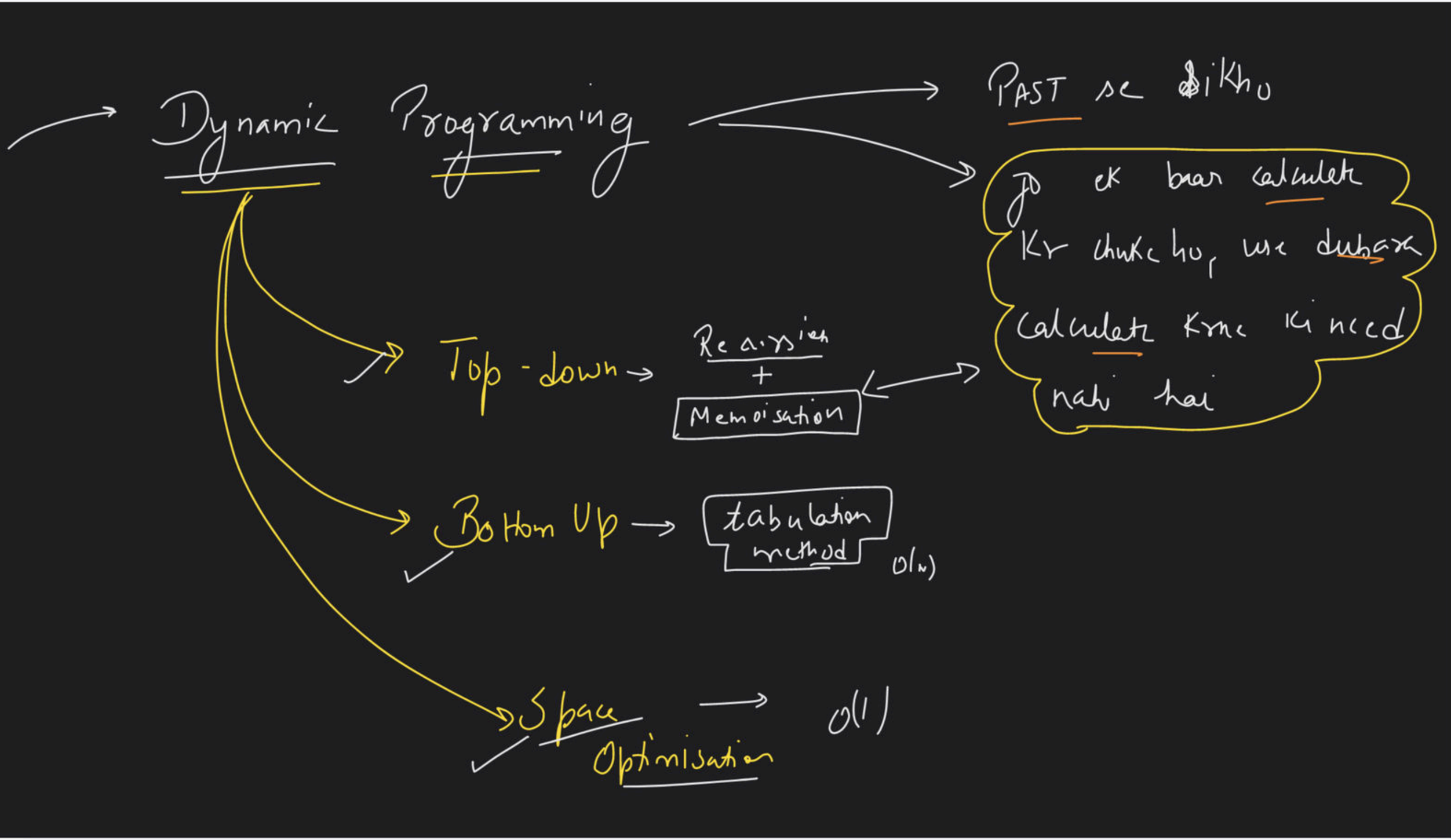
Dynamic Programming Class - 1

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

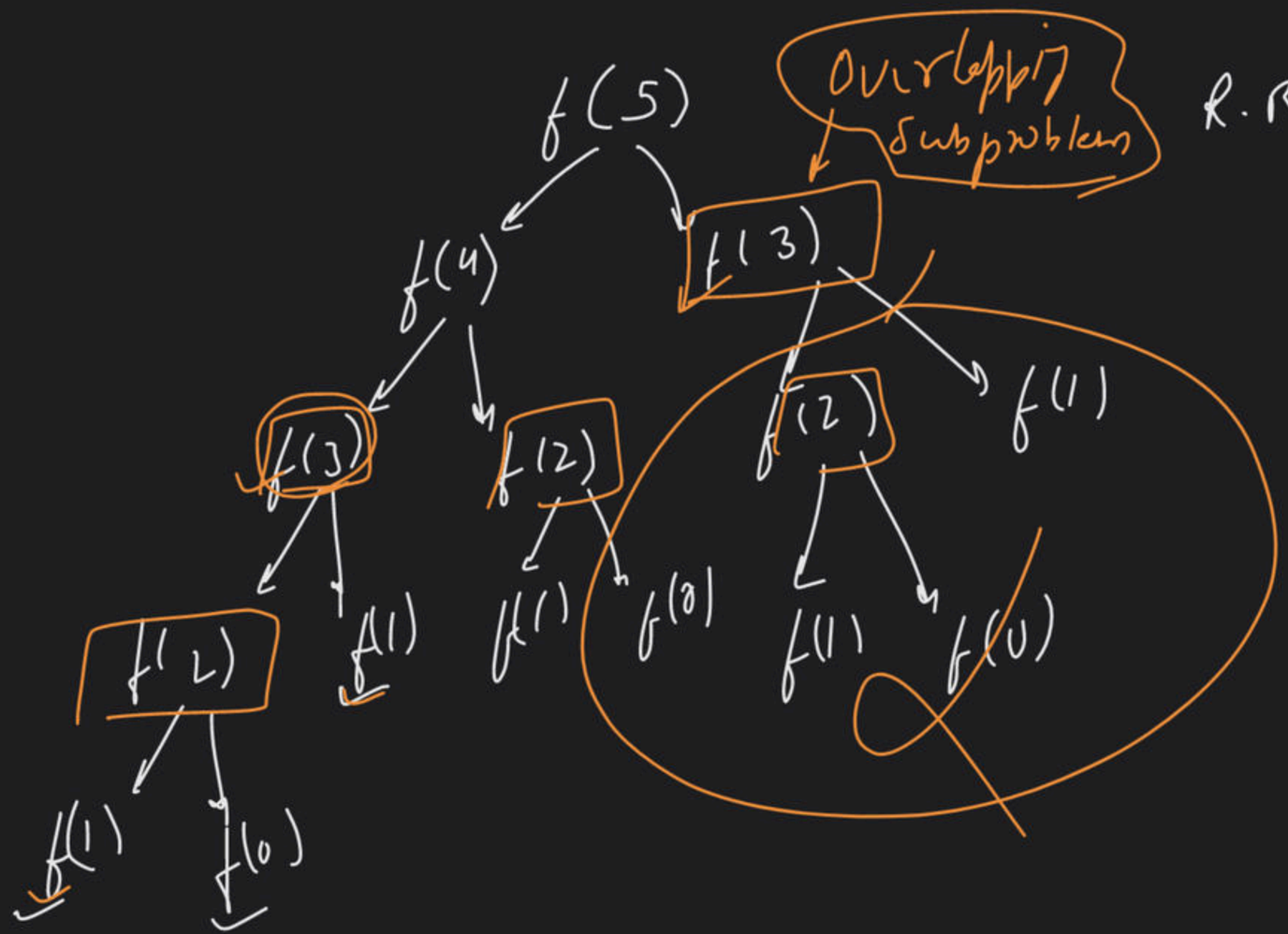
Same problem ko baar baar solve krre ho

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 → n (n → 0) 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

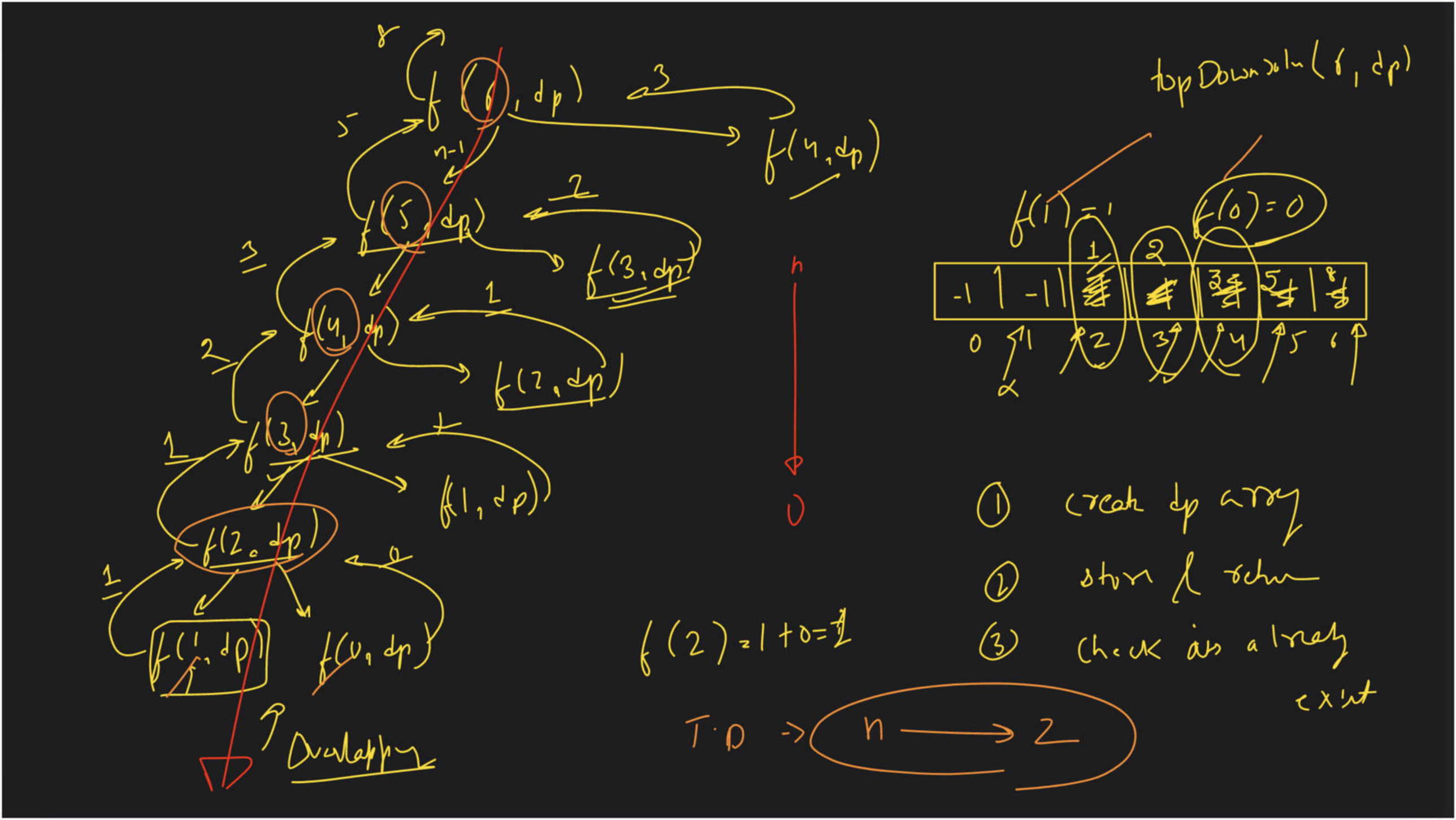
```
vector<int> dp (n+1, -1);
```

② ans store dp

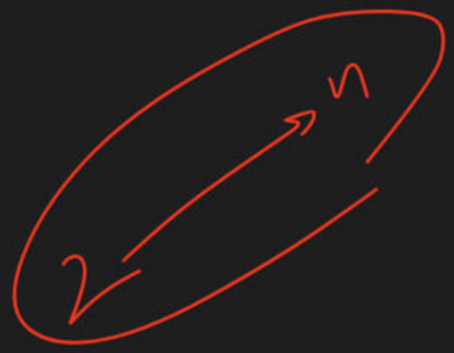
③ B.C ✗ Just base

↓

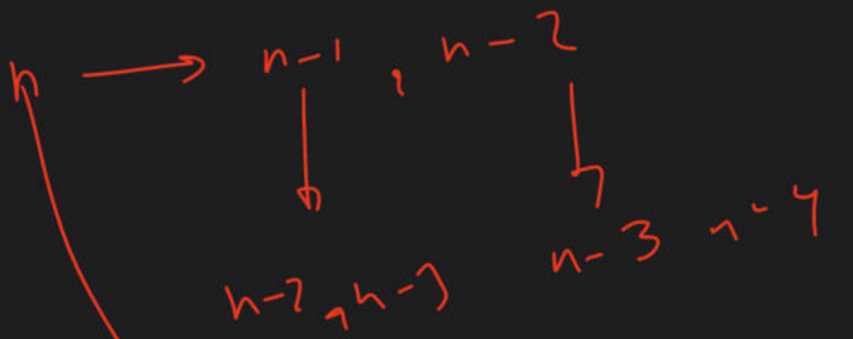
check if answer already exists

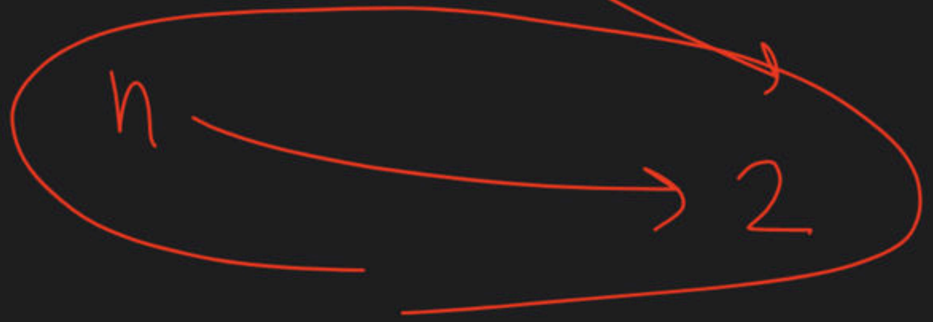

topDownSolve(6, dp)
f(6, dp)
f(5, dp)
f(4, dp)
f(3, dp)
f(2, dp)
f(1, dp)
f(0, dp)
n-1
f(1) = -1
f(0) = 0
-1 | -1 | 1 | 2 | 3 | 5 | 8
0 1 2 3 4 5 6
n
0
f(2) = 1 + 0 = 1
T.D -> n ⟶ 2
Overlapping
① create dp array
② store & return
③ check if already exist

top down



top down



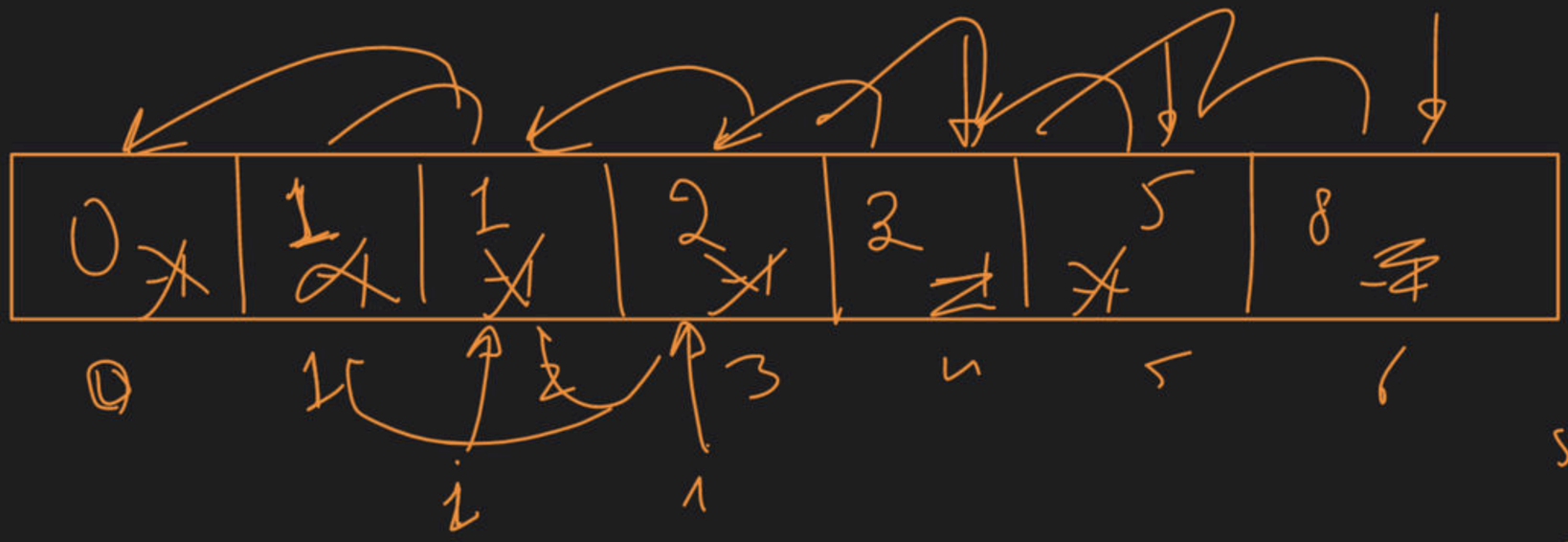


1.0

P

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

n+2

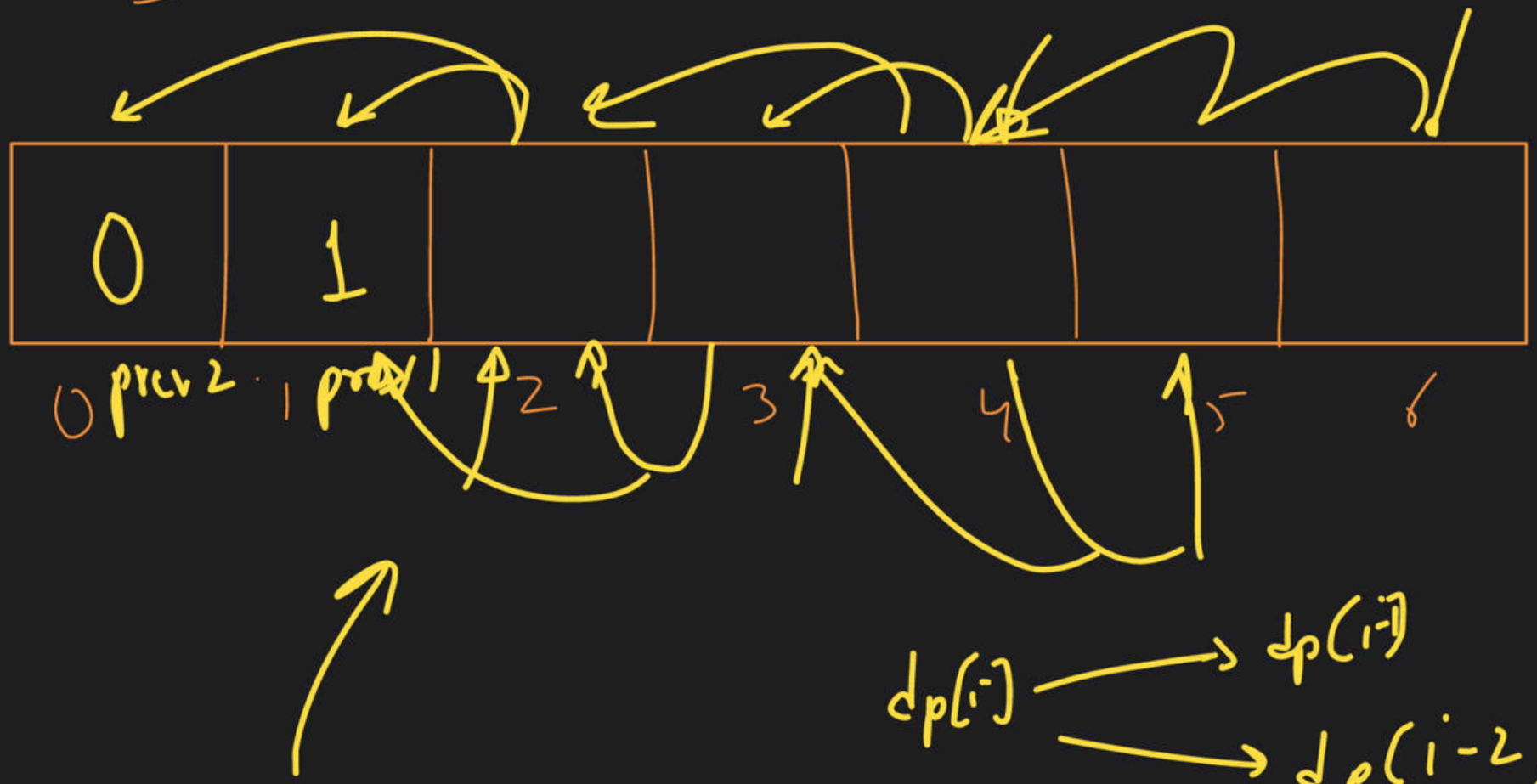
step 1 → dp array → 1 size

step dp[0] = 0

dp[1] = 1

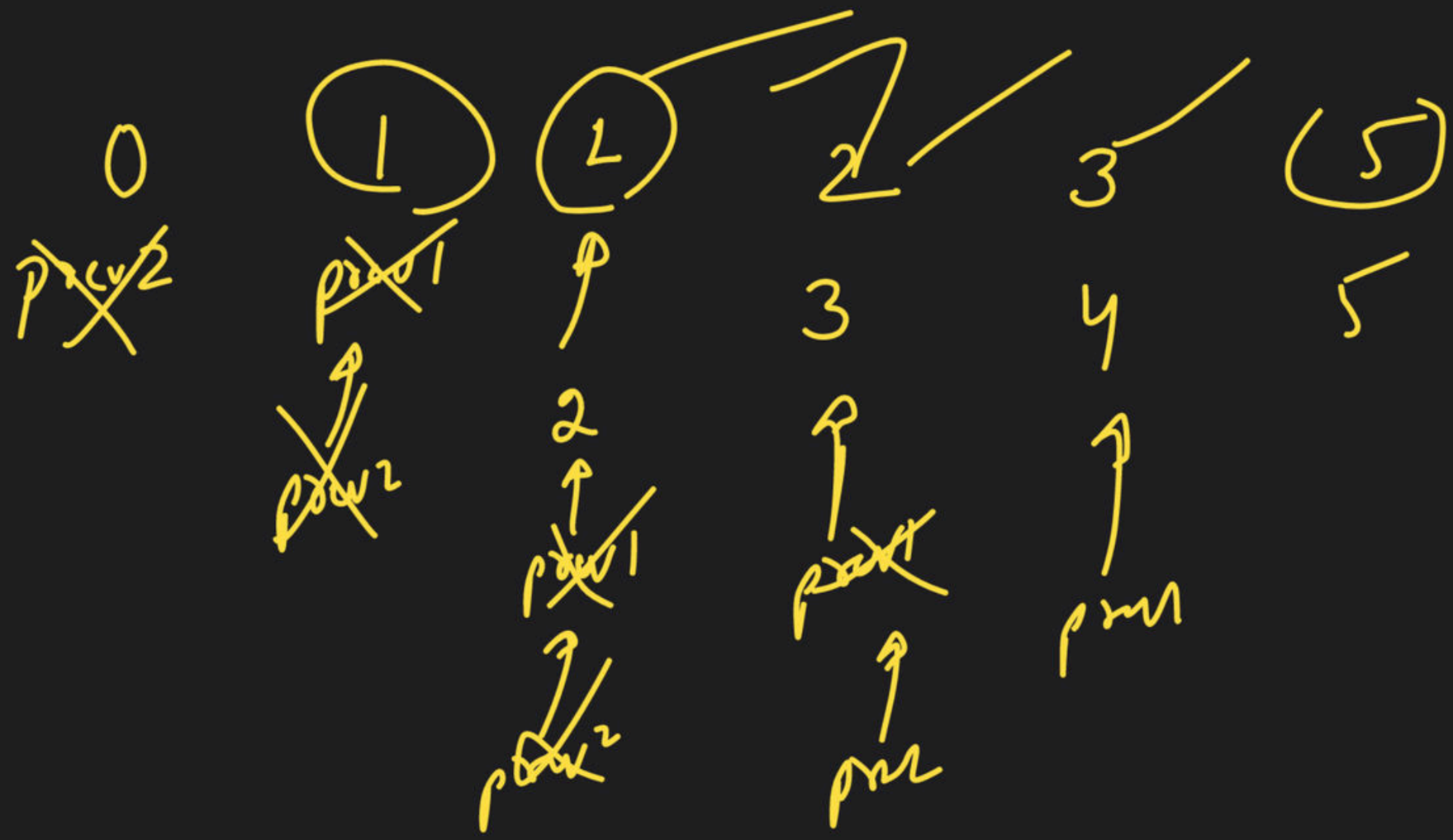
isKi wajah se factorial

Space Optimisation:-

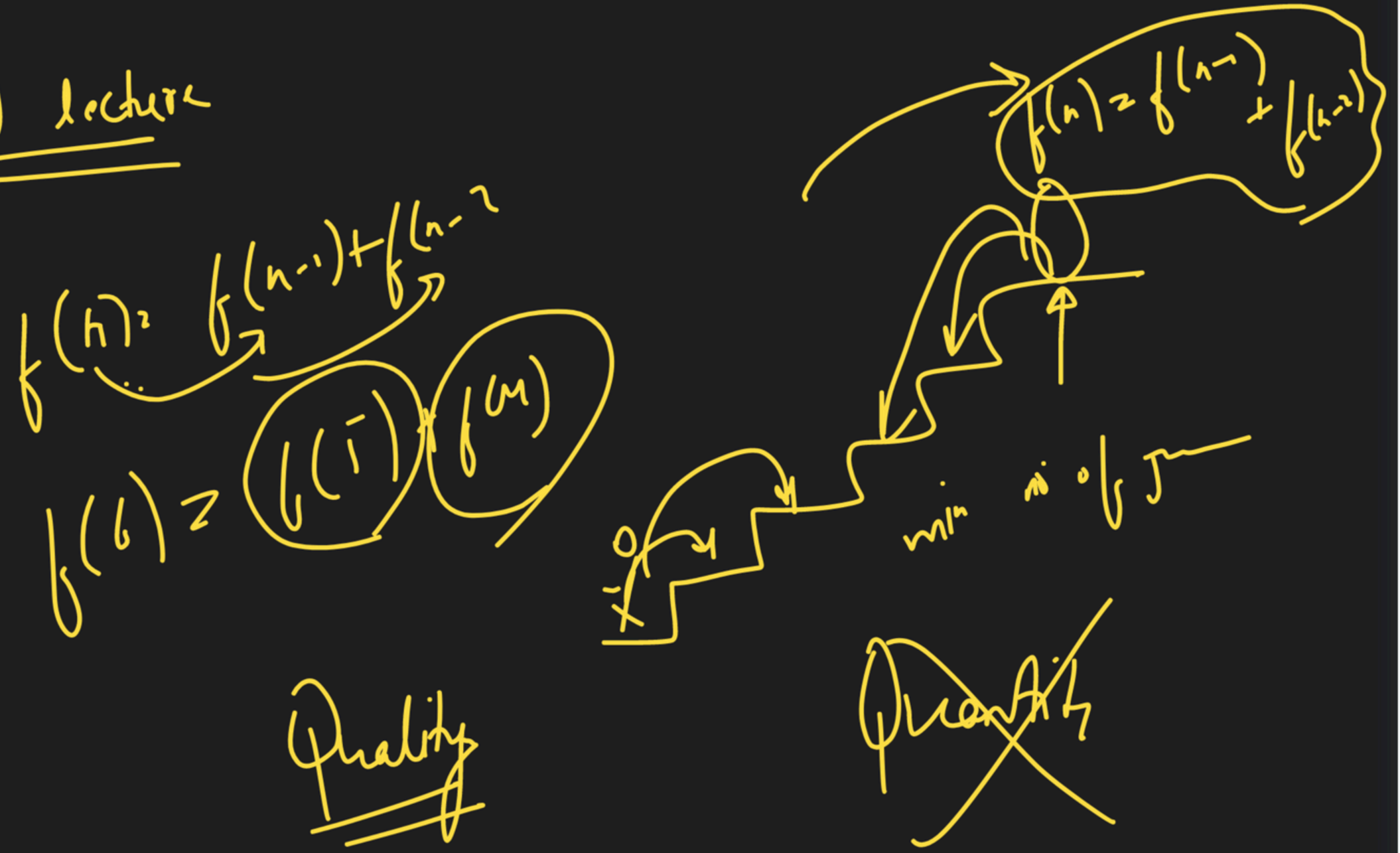


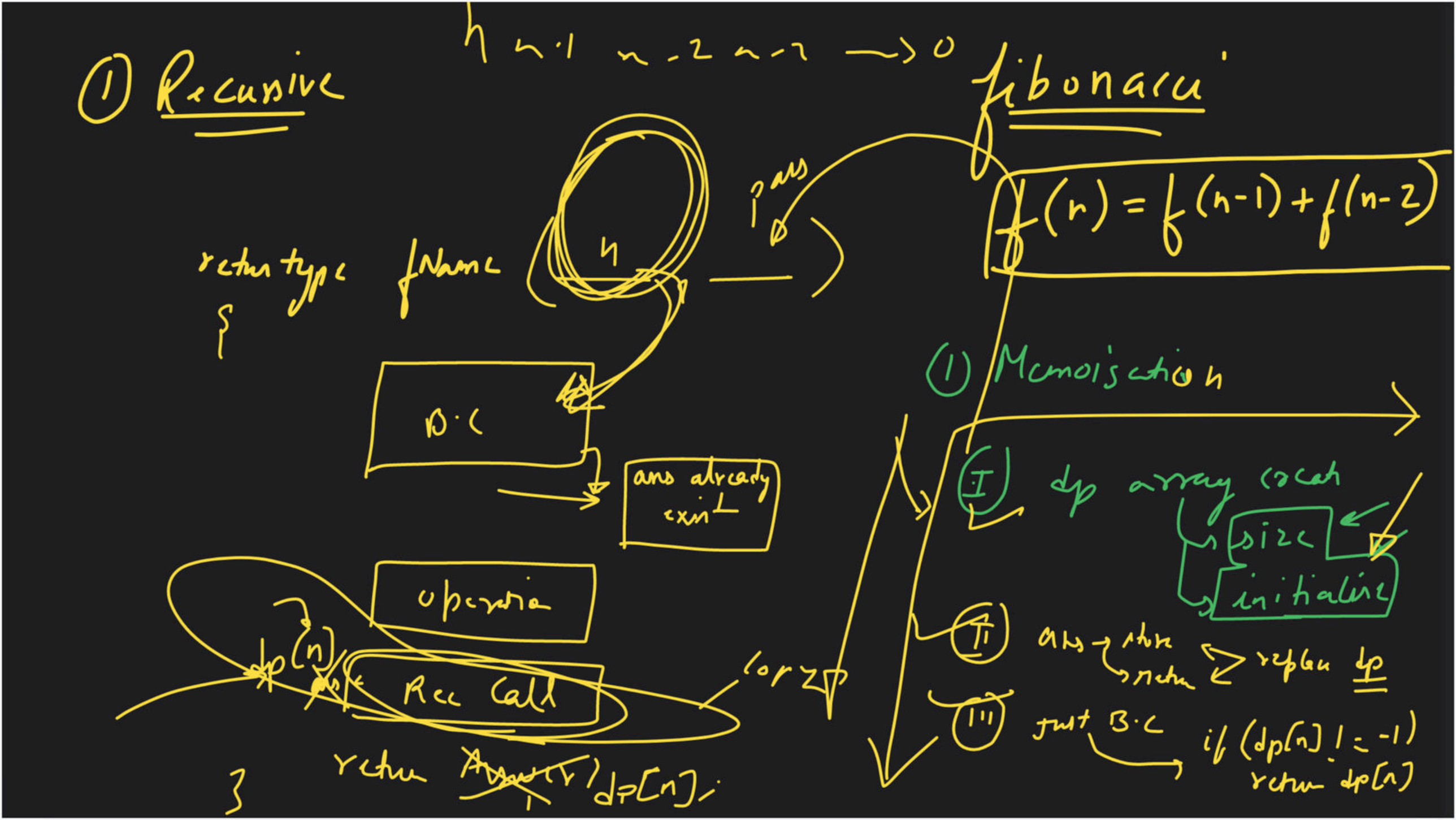
dp[i] → dp(i-1)

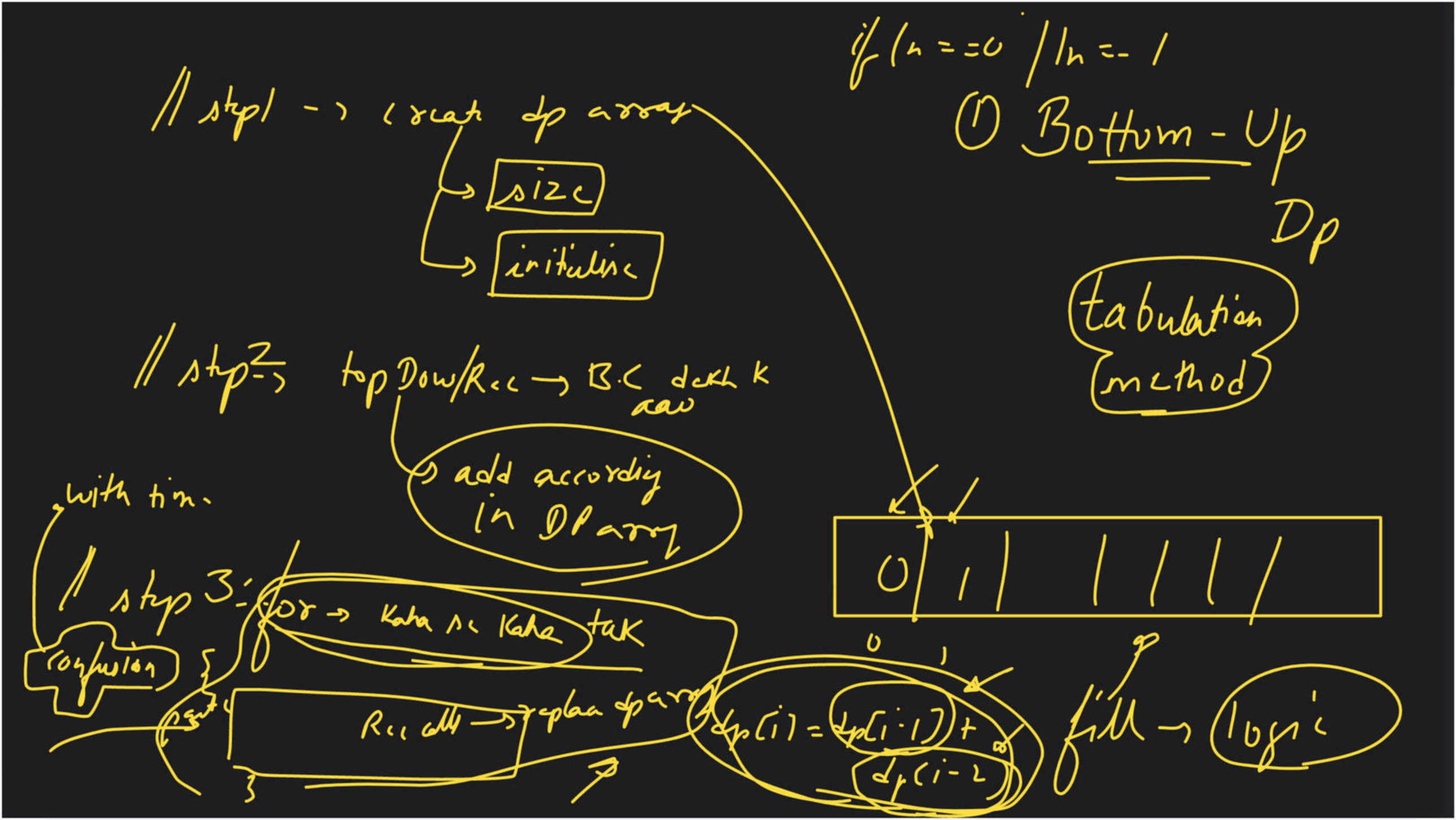
dp[i] → dp(i-2)

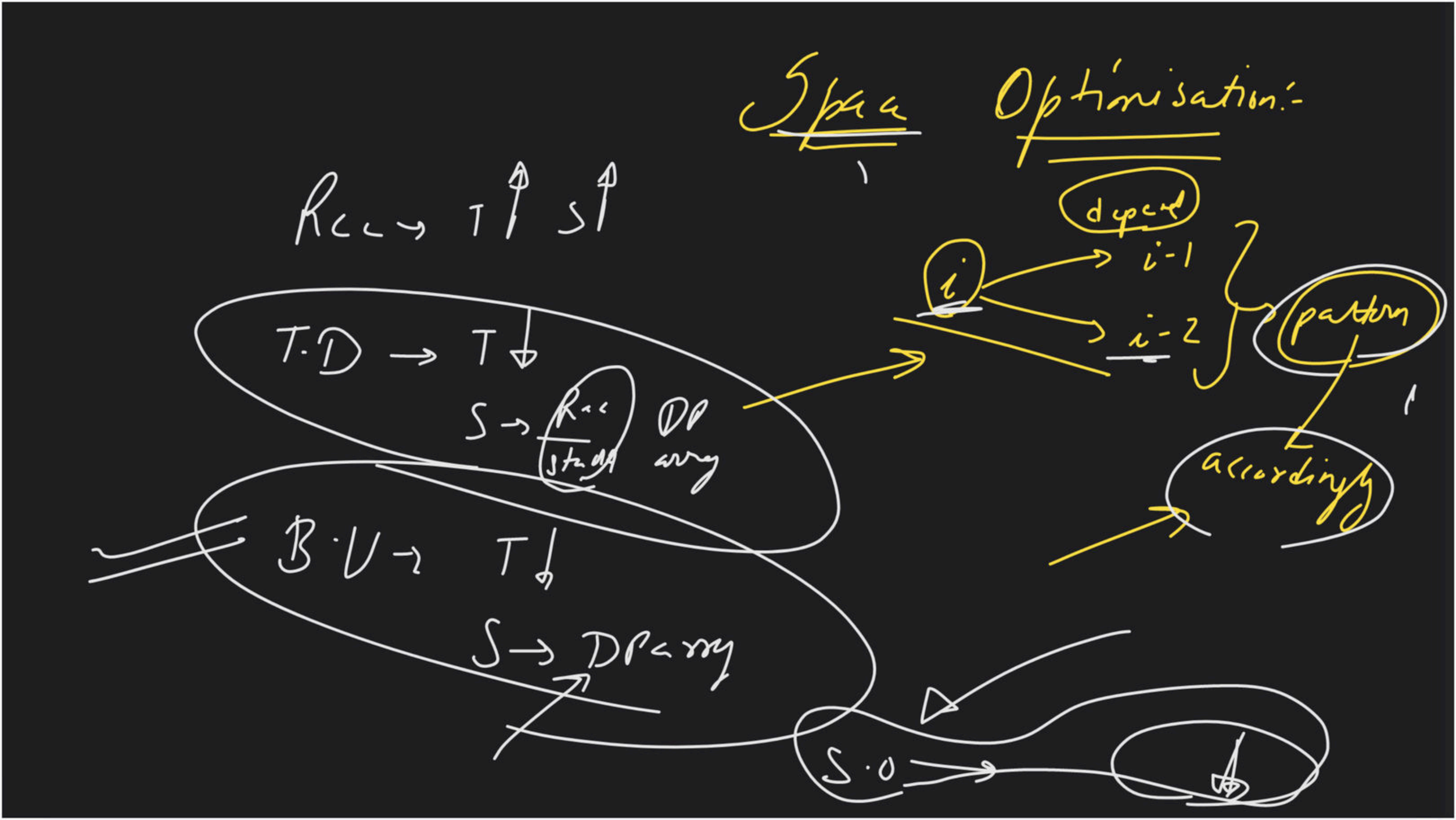

0
1
2
2
3
5
3
4
5
2
null

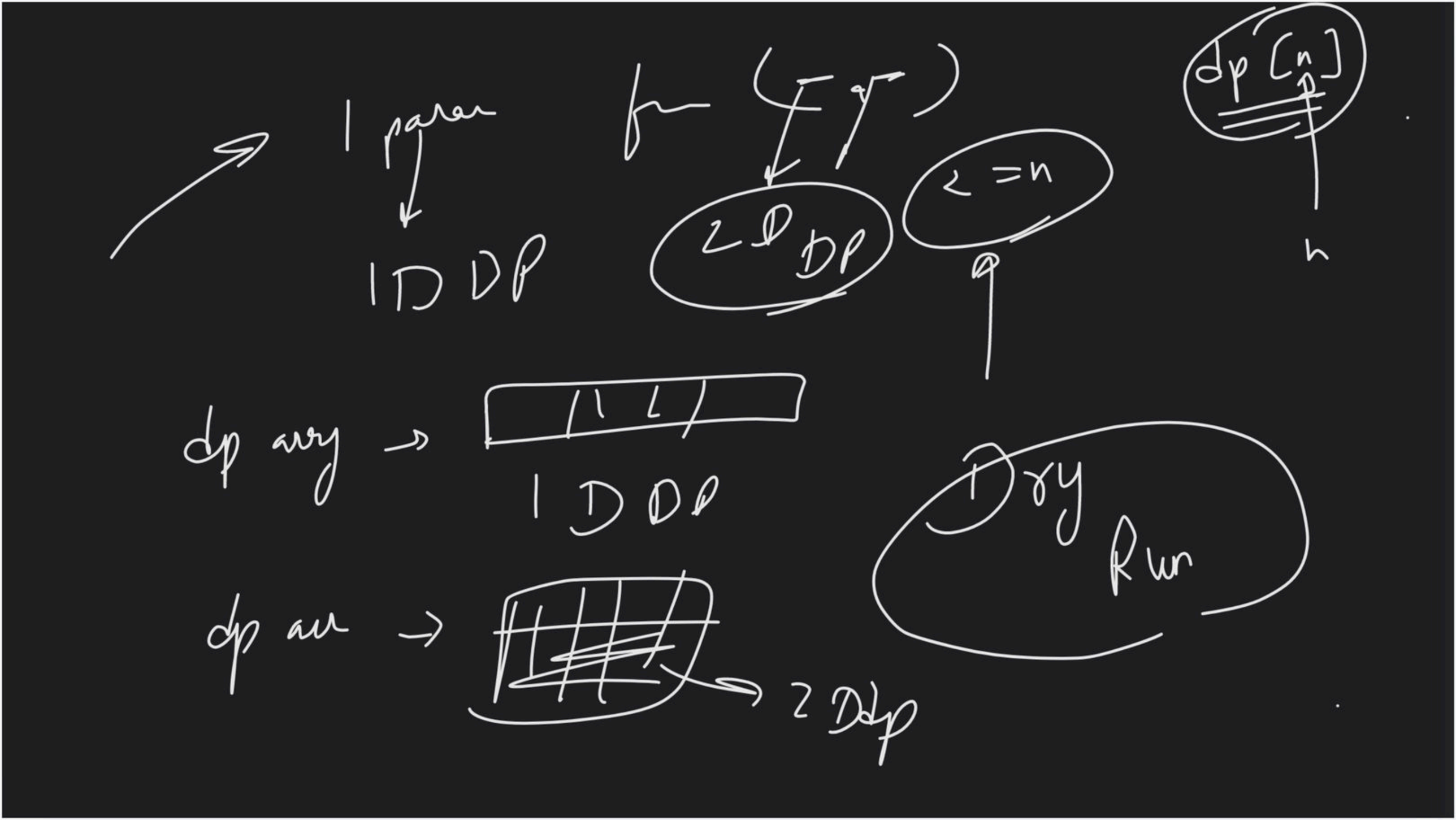
Kal lecture




① Recursive
n n-1 n-2 n-3 → 0
fibonarci
f(n) = f(n-1) + f(n-2)
pass
n
return type fName
{
B.C
ans already exist
operation
dp(n)
Rec call
return dp[n];
}
① Memoisation
(I) dp array (each
size
initialise
(II) ans → store return → return dp
(III) just B.C
if (dp[n] != -1)
return dp[n]


if (n == 0 || n == 1
// step1 -> create dp array
size
initialise
// step2 -> topDown/Rec -> B.C dekh k aao
add according in DP array
tabulation method
Bottom - Up
Dp
with tim.
// step 3:
for -> kaha se kaha tak
recursion
Rec call -> replace dp array
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
dpcal
i
i-1
i-2
pattern
according
S·O


1 param
fun(int n)
dp[n]
1D DP
2D DP
<=n
n
dp arr
1D DP
Dry
Run
dp arr
2D DP