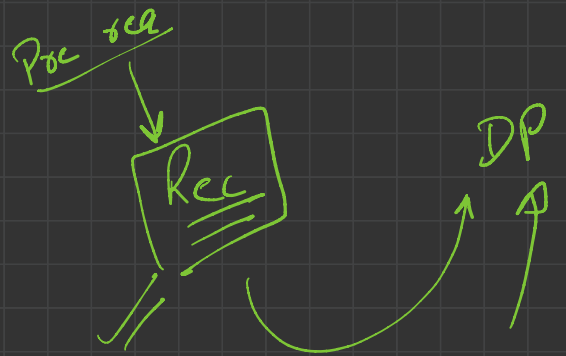
Dynamic Programming

what a → ?



[Those who forget the past,
are condemned to repeat it]
- DP

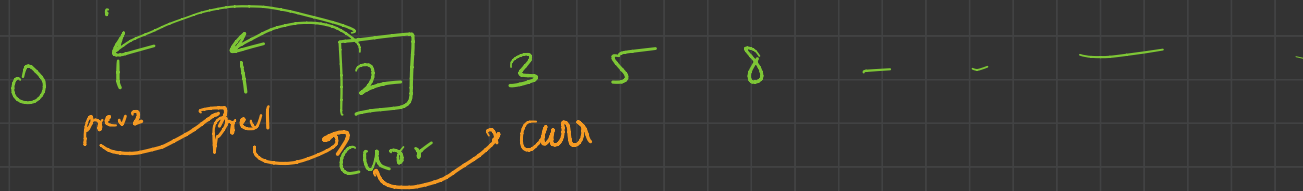
2 approaches :-

(i) → Top - Down → Recursion + Memoization

(ii) → Bottom - Up → Tabulation

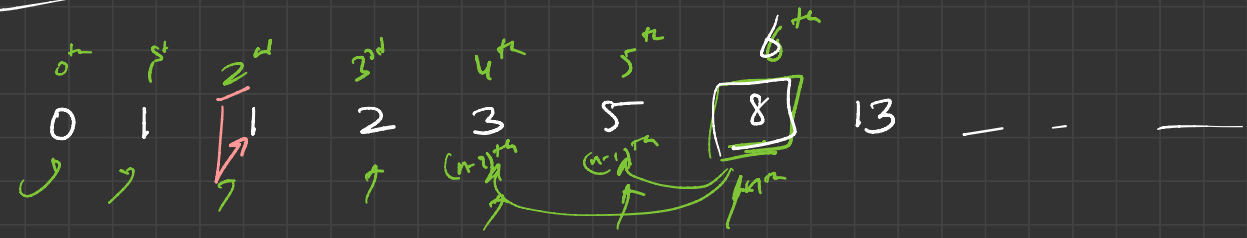
Memoization → store the value of subproblems in map / table

(iii) space Optimisation → Brain Storming



curr = prev1 + prev2

Fibonacci Series:-



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

n^{th} fibonacci number

Code

Recursive Implementation

① dp[n+1];

```
func ( int n)
{
    // B.C
    if (n == 1 || n == 0)
        return n;

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

    return dp[n] = f(n-1) + f(n-2);     II

}
```

Recursive soln

DP Sol

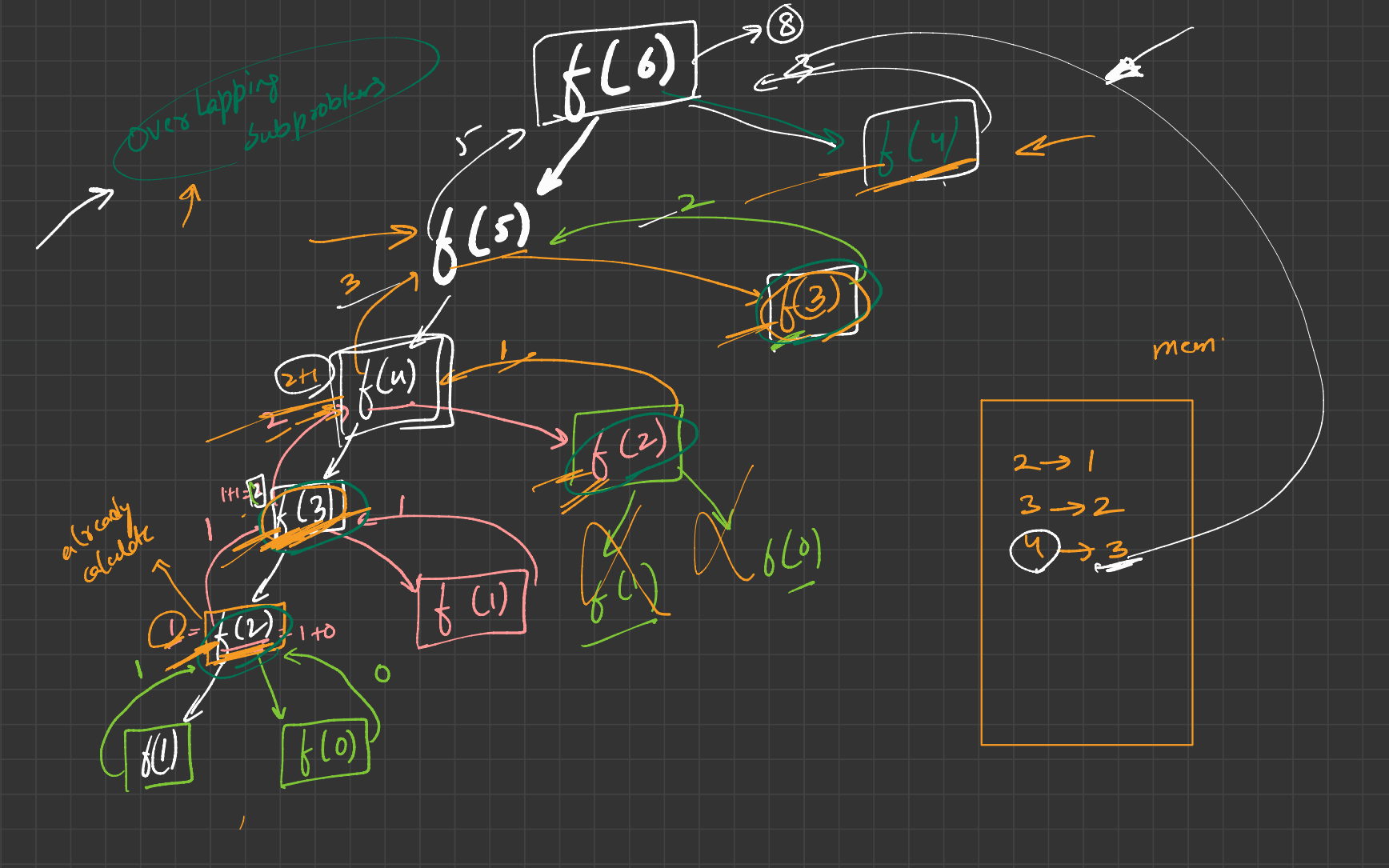
Top Down

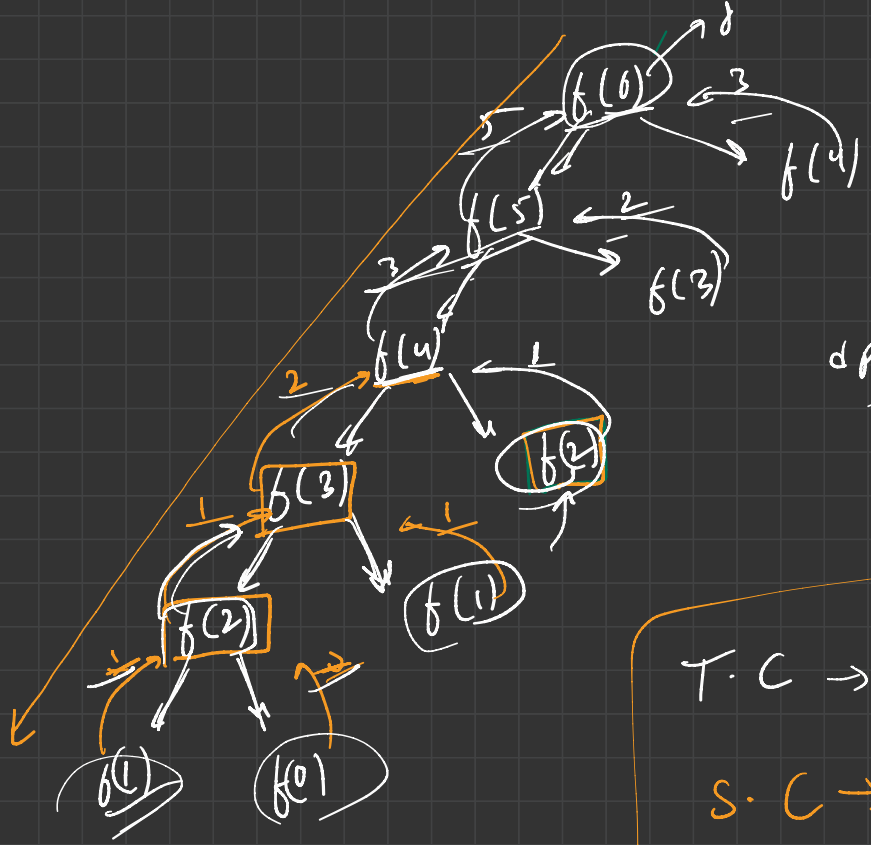
Rec + Memoization

DP

→ I

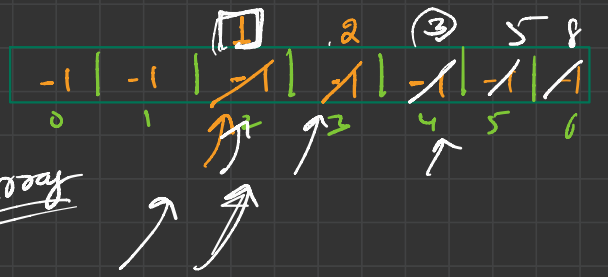
→ II


Overlapping subproblems
f(6)
8
f(4)
f(5)
2
3
f(3)
2+1
f(4)
1
f(2)
already calculate
f(3)
1+1=2
1
1
f(1)
f(2)
1+0
1
0
f(1)
f(0)
f(1)
f(0)
mem
2 → 1
3 → 2
4 → 3



1D array

(n+1) size



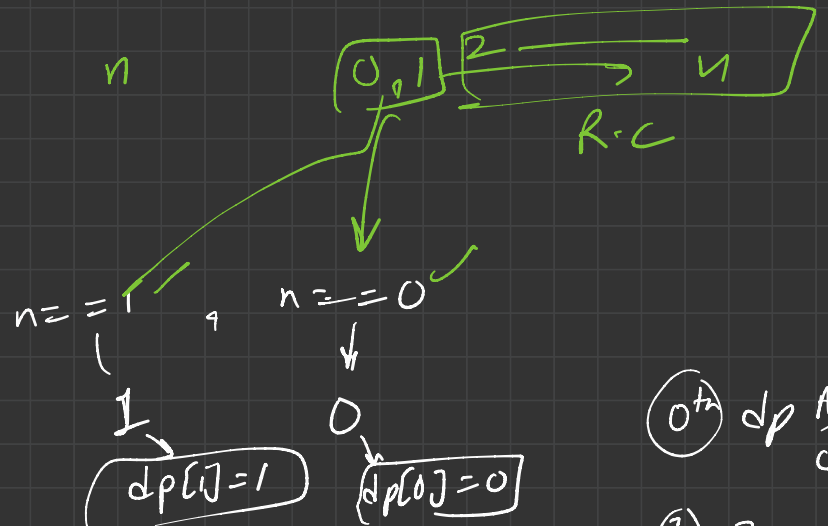
T.C → O(n)

S.C → O(N) + O(N)

→ O(N)

Code

```
fib( n , dp)
{
    if (n <= 1)
        return n;
    if (dp[n] != -1)
        return dp[n];
    dp[n] = fib(n-1) + fib(n-2);
    return dp[2];
}
```



0th dp Array Local

① Base case check

→ dp[1]=1

→ dp[0]=0

②

0

2
3
4
5
6
7
:
:
n

① fib (n)

f_n

{

① dp [n+1];

② prev1 dp[1] = 1;
prev2 dp[0] = 0;

③ for (int i=2 ; i<=n ; i++)

{

curr prev1 prev2
dp[i] = dp[i-1] + dp[i-2]

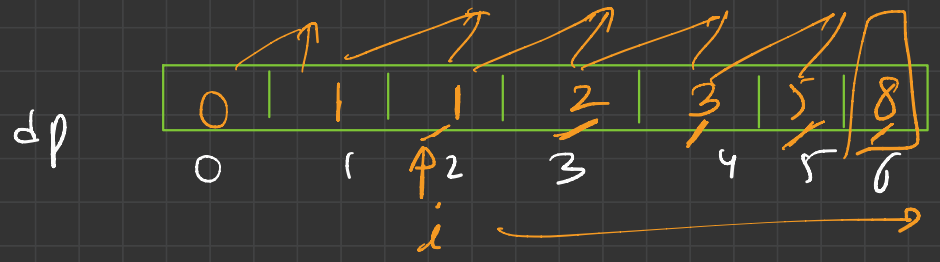
} ⟹ prev2 = prev1;
prev1 = curr;

else dp[n];
prev1

}

T.C -> O(n)

S.C -> O(n)



T.C → O(N)

S.C → O(1)

O(N)
O(N) + O(N)

Top-Down
Rec + Mem

O(N)
O(N)

Bottom Up
tabulation

O(N)
O(1)

space Optimiz

→ "Samajh aagya" → Like → Subscribe

→ "Maza aagya"

→ Love

(10)

consistency