If you don't remember the past,
You are condemned to repeat it.

Fibonacci Series.

index :	1	2	3	4	5	6	7	8
value :	0	1	1	2	3	5	8	13

Q Find n^{th} Fibonacci Number !

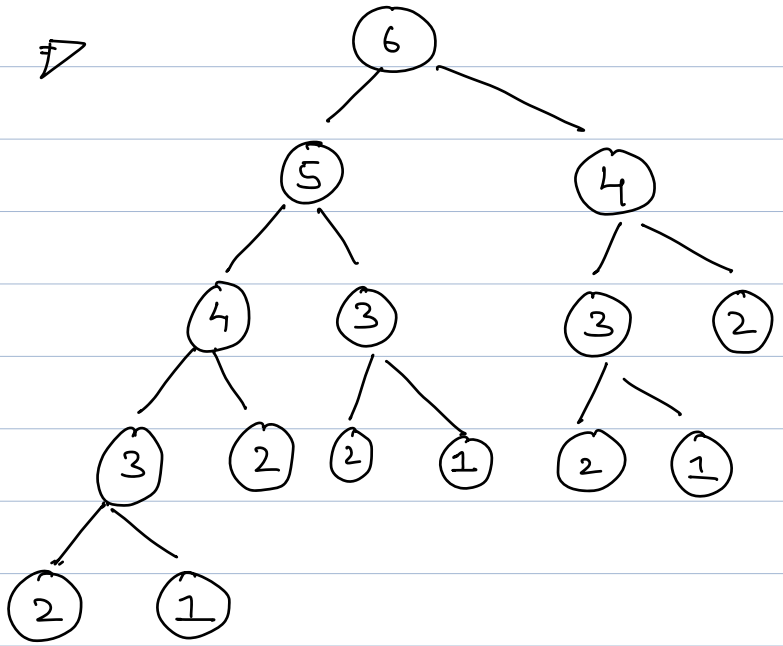
Recurrence Relation : $F(n) = F(n-1) + F(n-2)$

Pseudo Code

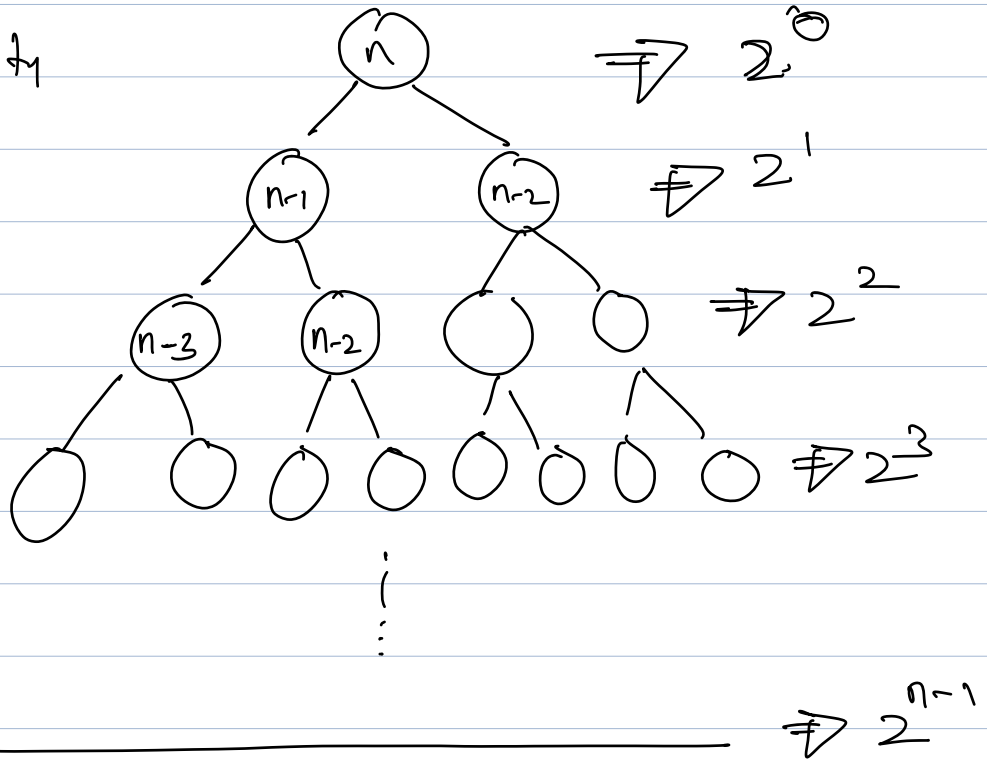
```
int Fib (int n) {
    if (n <= 2)
        return (n-1) ;

    return Fib(n-1) + Fib(n-2);
}
```

Fib(6) ⇒



Time Complexity

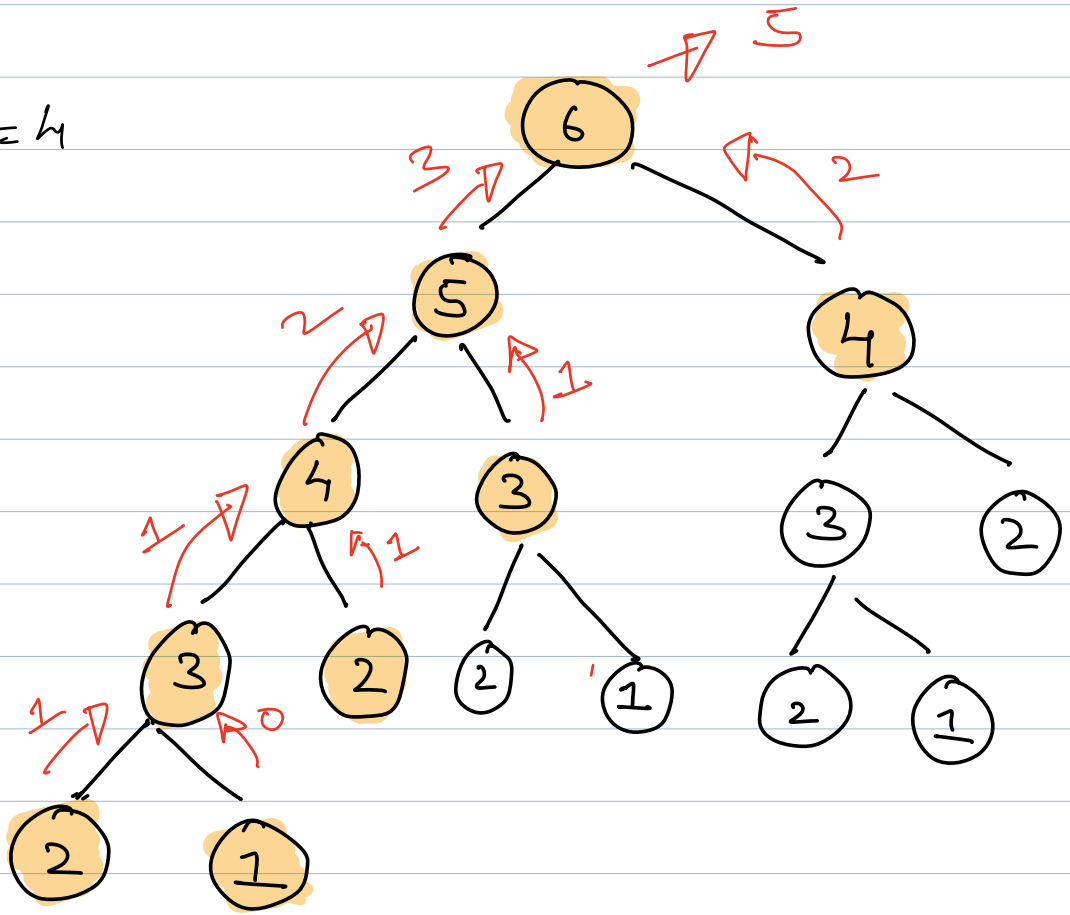


Total func. calls ⇒ 2^n

Time taken by each call ⇒ O(1)

TC: $O(2^n)$ C: O(n)

n = 4



0	1	2	3	4	5	6
-1	-1	-1	1	2	3	5

Total function calls $\simeq 2n$

TC : $O(n)$

SC $\simeq 2n$

SC : $O(n)$

(2) S ( )

When can DP be used ?

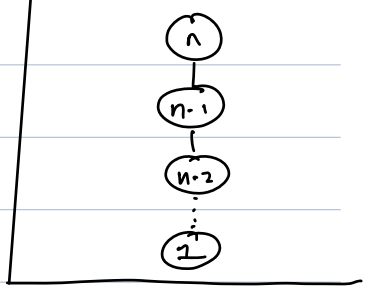
1) Optimal Substructure

Time Complexity ⇑

If you can find optimal solution to a problem using optimal solutions of smaller problems (sub-problems)

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

Sum(n) = Sum(n-1) + 1



2) Overlapping sub-problems.

Repitition of sub-problems

Total Func calls > Total Unique fun calls

Memoization

↳ Storing results of states (sub-problems) to use them in future.

Pseudo Code (Fibonacci Using DP)

```
int dp[n+1] = {-1}, dp[1] = 0, dp[2] = 1      // Store base cases.

int Fib(int n) {

    if (dp[n] != -1)                          // checking if previously calculated
        return dp[n];

    int ans = Fib(n-1) + Fib(n-2)             // calculating ans.
    dp[n] = ans;                              // store
    return ans;                               // return the ans

}
```

Two Ways of solving a DP Problem

Top Down	Bottom up.
→ recursive Code	→ iterative code
→ More space would be needed	→ More control on Space.
→ Easier to write	→ Not as Easy as Top down to code
→ memoization is used.	→ Tabulation.

Bottom fib

```
int dp[n+1]; dp[1] = 0 , dp[2] => 1

for (int i = 3 ; i <= n ; i++) {

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

}

return dp[n];
```

TC: O(n)

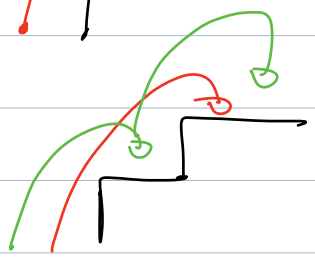
SC: O(n)

→ Can be reduced to O(1) using 2 variables.

Q2 Given N steps.
Find the no of ways of reaching the N^{th} stair. At a time, you can climb 1 stair or 2 stairs

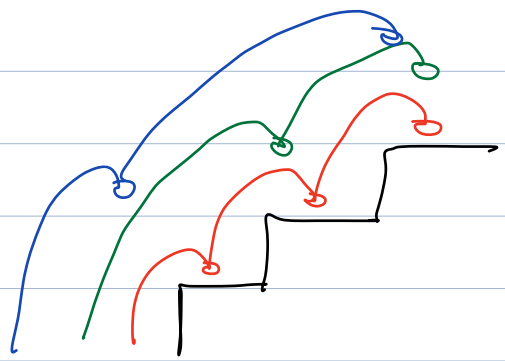
Ex1 n = 1 , ans = 1

Ex2 n = 2 , ans = 2



Ex3 n = 3

ans = 3



Steps to solve DP

1) Element of Choice.

To reach n^{th} stair → $n-1$
→ $n-2$

2) What does a state represent.

int ways(n) ⇒ No of ways to reach n^{th} stair.

3) Recurrence relation

ways(n) ⇒ ways(n-1) + ways(n-2)

4) Which state gives you final answer.

return ways(n);

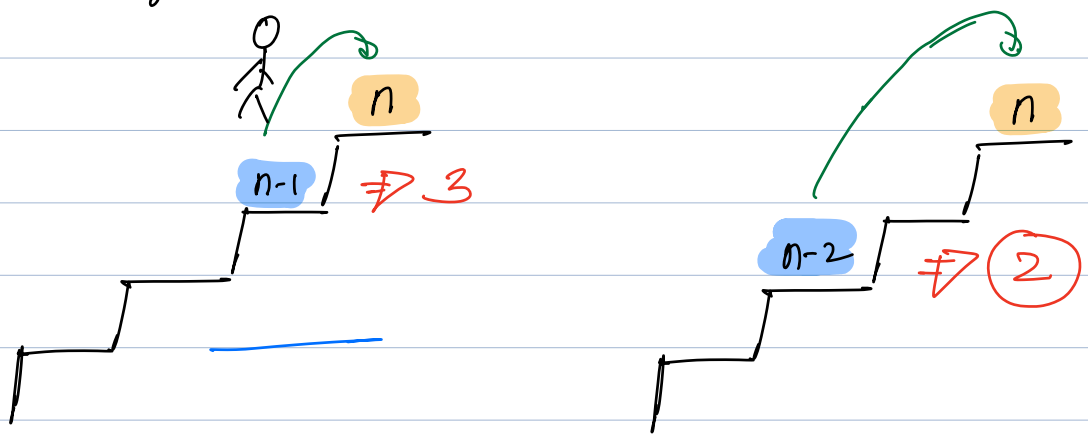
ways(n) ⇒ ways(n-1) + ways(n-2)

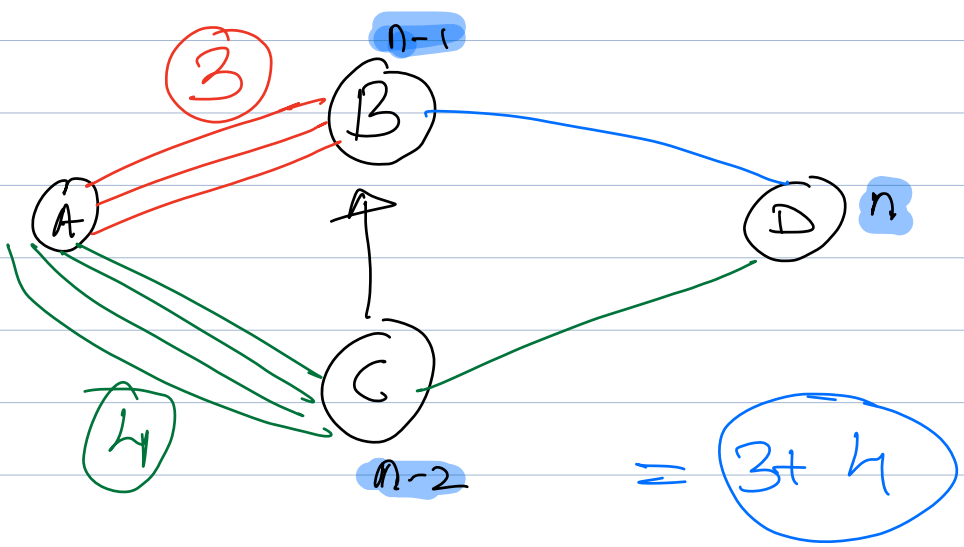
ways(n) ⇒ ways(n-1) + ways(n-2) + 2

ways(n) ⇒ ways(n-1) + ways(n-2) + 1

ways(n) ⇒ ways(n-1) X ways(n-2)

ways(n) ⇒ ways(n-1) + ways(n-2)





Q3 Let's Party

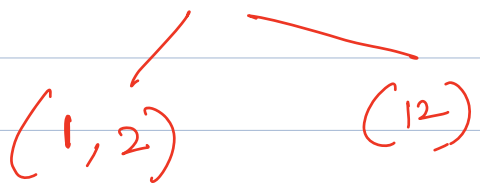
You are at a party. There are n people

1) Dance alone
2) Dance with someone

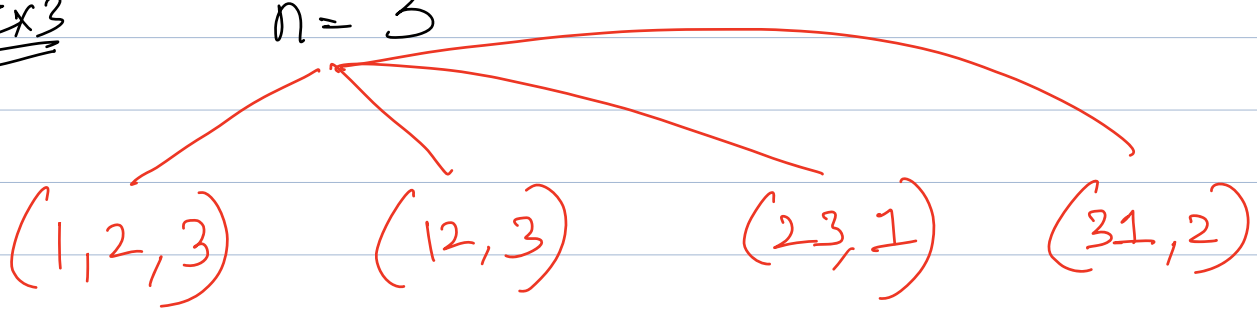
How many different ways of dancing is there.

Ex1 n = 1 ans = 1

Ex2 n = 2 ans ⇒ 2



Ex3 n = 3



Steps to solve DP

1) Element of Choice.

Person → dance alone

Person → Choose a partner & dance.

2) What does a state represent.

int ways(n) ⇒ no of ways to make n people dance.

3) Recurrence relation

ways(n) ⇒ ways(n-1) + (n-1) * ways(n-2)

4) Which state gives you final answer.

return ways(n);

ways(n)

Case 1

Dance alone

Paste

n-1

1

ways(n-1)

5

Case 2

Dance with someone

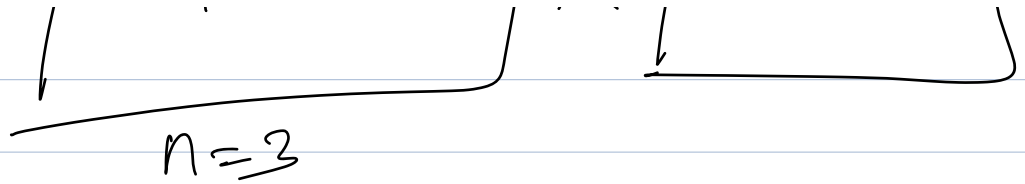
⇒ n-1

ways(n-2) x (n-1)

n-1 n

n-2

10 x 4



n=5

Ex1 n = 1 ans = 1

Ex2 n = 2 ans ⇒ 2

(1,2) (12)

Ex3 n = 3

(1,2,3) (12,3) (23,1) (31,2)

Ex4 n = 4 → (1,2,3,4), (12,3,4)

(23,1,4), (31,2,4)

(4 1, 2,3) , (4 1, 23)

(42, 1,3) , (42, 13)

(43, 1,2) , (43,12)

ways(3)

+

3 x ways(2)