Dynamic Programming

Those who cannot remember the past are condemned to repeat it..

~ DP.

1) Tabulation (Bottom UP DP) (0 → N)
2) Memoization (Top Down DP) (Doesn't Always means (N → 0))

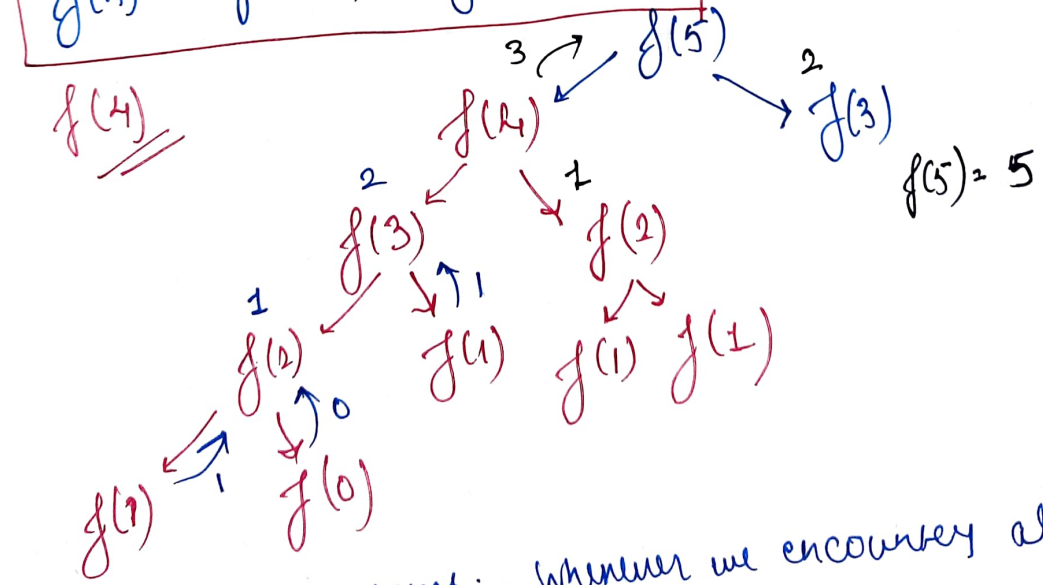
Memoization → Tabulation
- Memoization: TC, SC
- Tabulation: TC, SC → Optimise → final soln

L1.

6th, 7th → Recursion Playlist

0 1 1 2 3 5 8 13 21 ....

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



Overlapping Subproblems :- Whenever we encounter already solved subproblems.

MEMOIZATION – Technique of storing value of subproblems in map / table. (2)

dp, fibo

1	2	3	5	8

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

n+1

1. Declare dp[n+1]

```
f(n) {
    if (n ≤ 1)
        return n;
    return f(n-1) + f(n-2);
}
```

(2)

(3) → `if(dp[n] != -1) return dp[n];`

RECURSION → DP.
(3) steps.

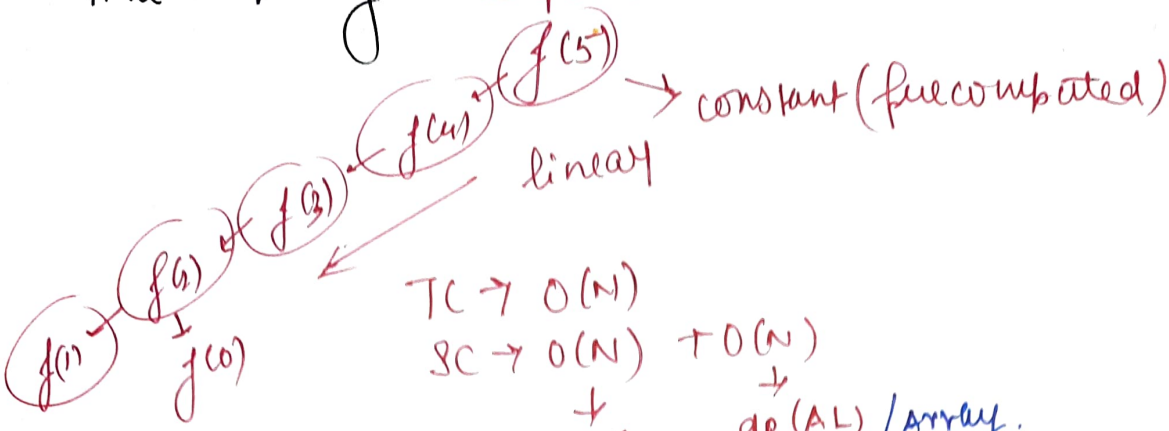
In Interview you won't be allowed to use global variables.

CODE:

```
f(int n, AL<Integer> dp) {                    → (1)
    if (n<=1)
        return n;
    if (dp[n] != -1) return dp[n];            → (2)
    return dp[n] = f(n-1, dp) + f(n-2, dp);
}

main() {
    int n;
    n = 5;
    ArrayList<Integer> dp(n+1, -1)
                          ↓    ↓
                        size  value fill.
    sout(f(n, dp));
```

Time Complexity — deeper calls.



TC → O(N)

SC → O(N) + O(N)

↓ stack space (Recursion)

↓ dp (AL) / array.

Recursion → Tabulation (Bottom up)

Bottom up.

Base case → Required ans.

Top down

Base case

dp[0] = 0 Prev 2

dp[1] = 1 prev

```
for(int i=2; i<=n; i++){
    dp[i] = dp[i-1] + dp[i-2];
}
```

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

prev prev2

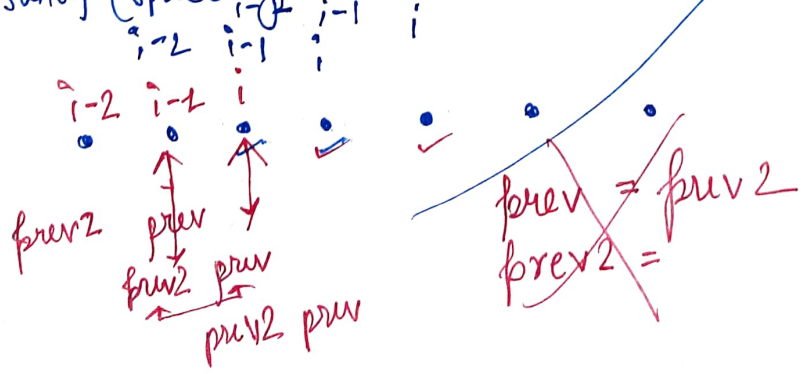
prev2 = prev

prev = i;

→ Recursion stack space required

SC → O(2N) → O(N)

Optimisation (Space further)



```
int n=4;
int prev2=0;   int prev2=0;
int prev=1;    int prev=1;

for(int i=2;i<=n;i++){
    int curr1 = prev + prev2
        prev2 = prev
        prev = curr1;
}
```

Always a space optimisation;

o/p = 5

Recursion → Tabulation (Bottom up)

Base case → Always Required

```
dp[n+1]
dp[0] = 0
dp[1] = 1
for(int i<=2; i<=n; i++)
    dp[i] = dp[i-1] + dp[i+2];
```

TC = O(N)

SC - O(N)

+ ?

O(1)

L2

1D problem

→ Understand a DP problem.

Count the total no of ways.

Try all possible ways

best ↙ ↘ count.

Shortcut trick

↓

1. Try to represent problems on index.
2. Do all possible stuffs on that index
3. Find the best

0, 1 ..... n

f(n) → no. of ways to reach (0 → n)

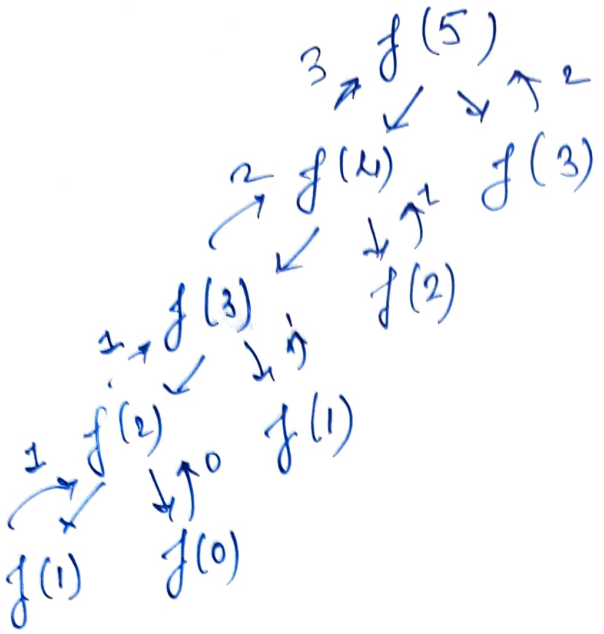
```
f(index)
    if (num == 0) return 1;
            f(index - 1)
            f(index - 2)
    if (index <= 1)
            return index;
```

0 → 0
1 → 1

~~ans~~ return.

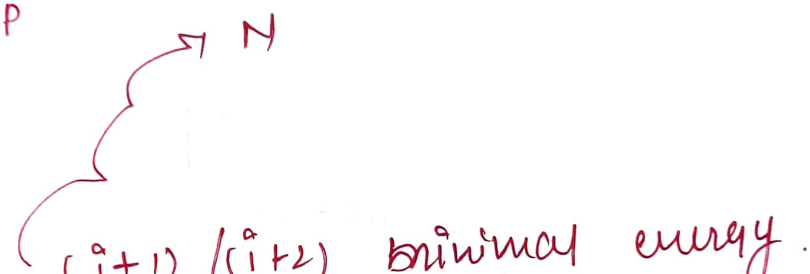
```
int left = f(index - 1)
int right = f(index - 2);
```

n = 5

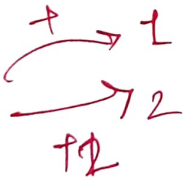


L3.

Frog Jump



index
do all things on index




```
if (index == 0) return 0;

int left = f(index - 1) + f( abs( index
```