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

— DP.

1) Tabulation → Bottom Up

2) Memoization → Top-Down

Memoization → Tabulation

↳ T.C

S.C

↳ TC

SC → Space Optimization

① Fibonacci :

0	1	2	3	4	5	6	7	8	
0	1	1	2	3	5	8	13	21	. . .

$f(n) = f(n-1) + f(n-2)$ // Recursion Relation.

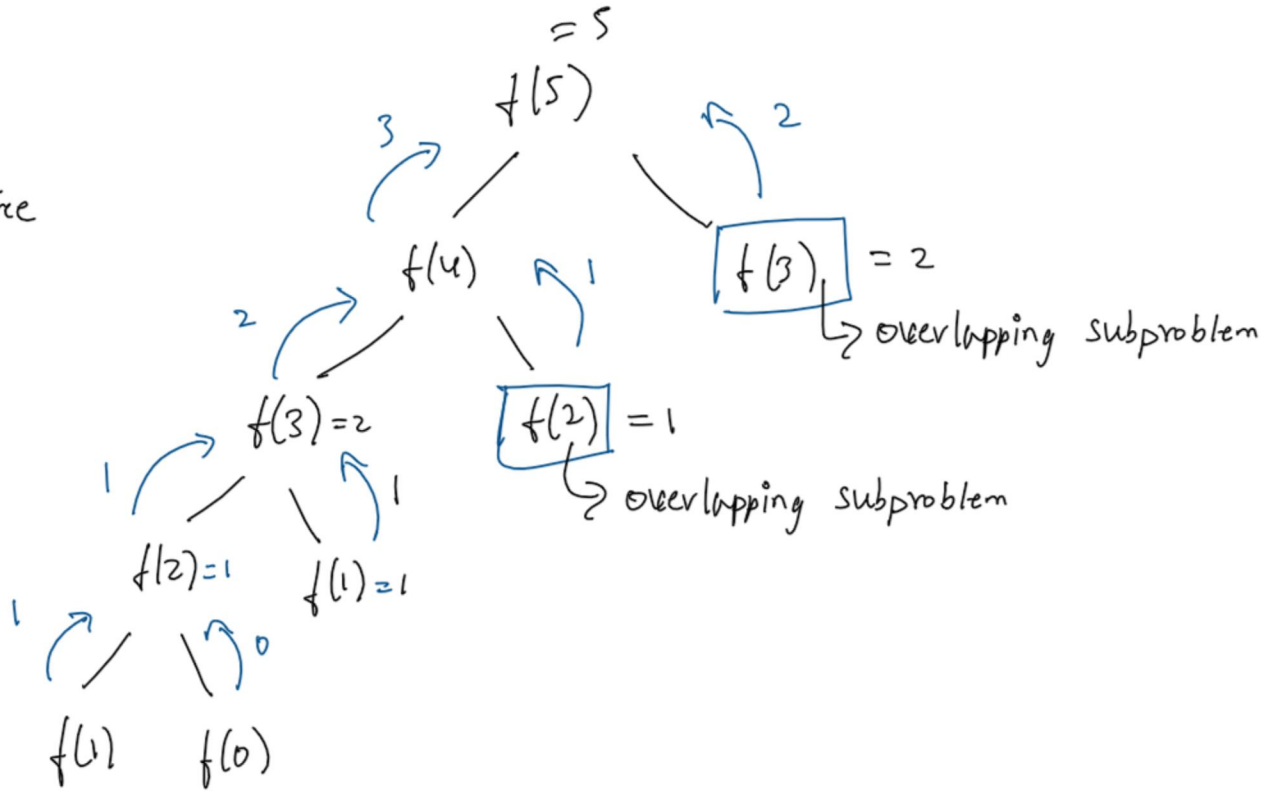
P-Seudo Code :

```
f(n)
{ if (n<=1)
      return n;
  return f(n-1) + f(n-2);
```

}

$\underline{\underline{n = 5}}$

Recursive Tree



<u>Memoization</u>: We tend to store the value of subproblem in some map/table.

Here we have f(2), so only one parameter there. So, then it's 1D array.

dp	-1	-1	~~-1~~ 1	~~-1~~ 2	-1	-1
	0	1	2	3	4	5

<u>Steps</u>:

1. Create a dp [n+1] array initialized to -1.
2. Whenever we want to find the ans of a particular value (say n) we 1st check whether the ans is already calculated using the dp array (i.e. dp[n] != -1).

3. If yes, simply return the value from the dp array.

4. If NOT, then we are finding the answer for the given values for the 1st time, we will use recursive relation as usual but before returning from the function, we will set dp[n] to the solution we get.

declare dp[n+1] S-1

```
f(n)
{ if (n<=1)
      return n;
  if (dp[n] != -1) return dp[n]; S-3
  return f(n-1) + f(n-2);
}
```

dp[n]= S-2

Recursion → DP (3) Steps.

```
#include<bits/stdc++.h>
using namespace std;

int f(int n, vector<int>& dp){
    if(n≤1) return n;

    if(dp[n]≠ -1) return dp[n];
    return dp[n]= f(n-1,dp) + f(n-2,dp);
}

int main(){
    int n = 5;
    //declare the dp of size n+1
    vector<int> dp(n+1, -1);
    cout<<f(n, dp);
    return 0;
}
```

T.C = O(N)

S.C = O(N)

Tabulation : (Bottom-UP)

Base case to the required

Recursion $\longrightarrow$ Tabulation (Bottom UP)
(Top Down)

answer
$\downarrow\uparrow$
base case

Steps:

* Declare a dp[] array of size n+1
* First initalize the base condition values, i.e. i=0 and i=1 of the dp array as 0 and 1 respectively.
* loop which traverse an array ($i=2 \longrightarrow n$) and for every index set its value as dp[i-1] + dp[i-2]

<u>Pseudo Code:</u>

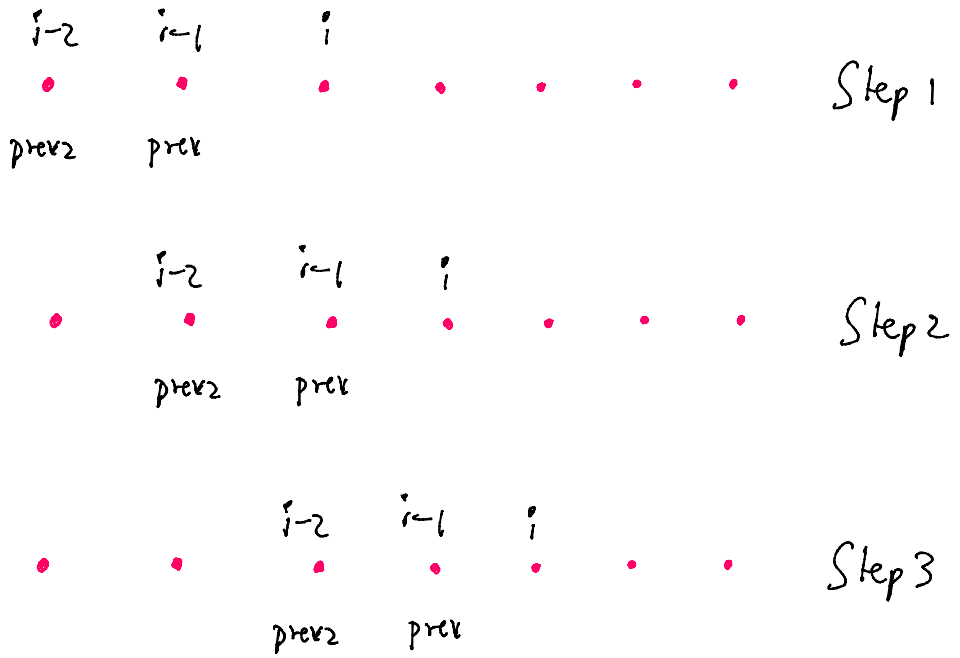
```
dp[n+1]

dp[0] = 0     dp[1] = 1

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

```cpp
#include <bits/stdc++.h>
using namespace std;


int main() {

  int n=5;
  vector<int> dp(n+1,-1);

  dp[0]= 0;
  dp[1]= 1;

  for(int i=2; i≤n; i++){
      dp[i] = dp[i-1]+ dp[i-2];
  }
  cout<<dp[n];
  return 0;
}
```

T.C = O(N)

S.C = O(N)

Space Optimization :



Steps :

1) At each iteration's curi and prev becomes the next iteration's prev and prev2 respectively.

2) So after calculating curi, if we update prev and prev2 according to the next step, we will always get the answer.

3) After the iterative loop has ended we can simply return

prev as our answer

<u>Pseudo Code</u> :

```
prev2 = 0          prev = 1
for(int i=2; i<=n; i++)
  {
    curr i  =  prev + prev2;
    prev2  =  prev;
    prev   =  curr i;
  }
print(prev)
```

T.C ⇒ O(N) S.C ⇒ O(1)