Why Dynamic Programming (DP)

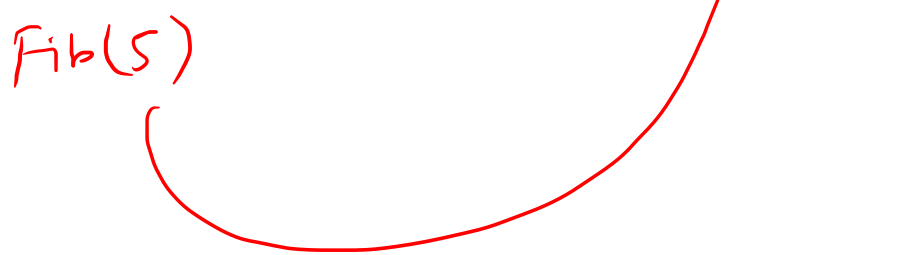
DP is Enhanced Version of Recursion

> Let's understand this with some example

(N^{th} Fibonacci Number)

=> (N = 5)

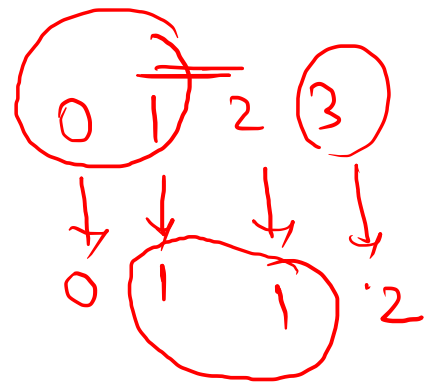
Fibonacci Number : { 0 1 1 2 3 5 }



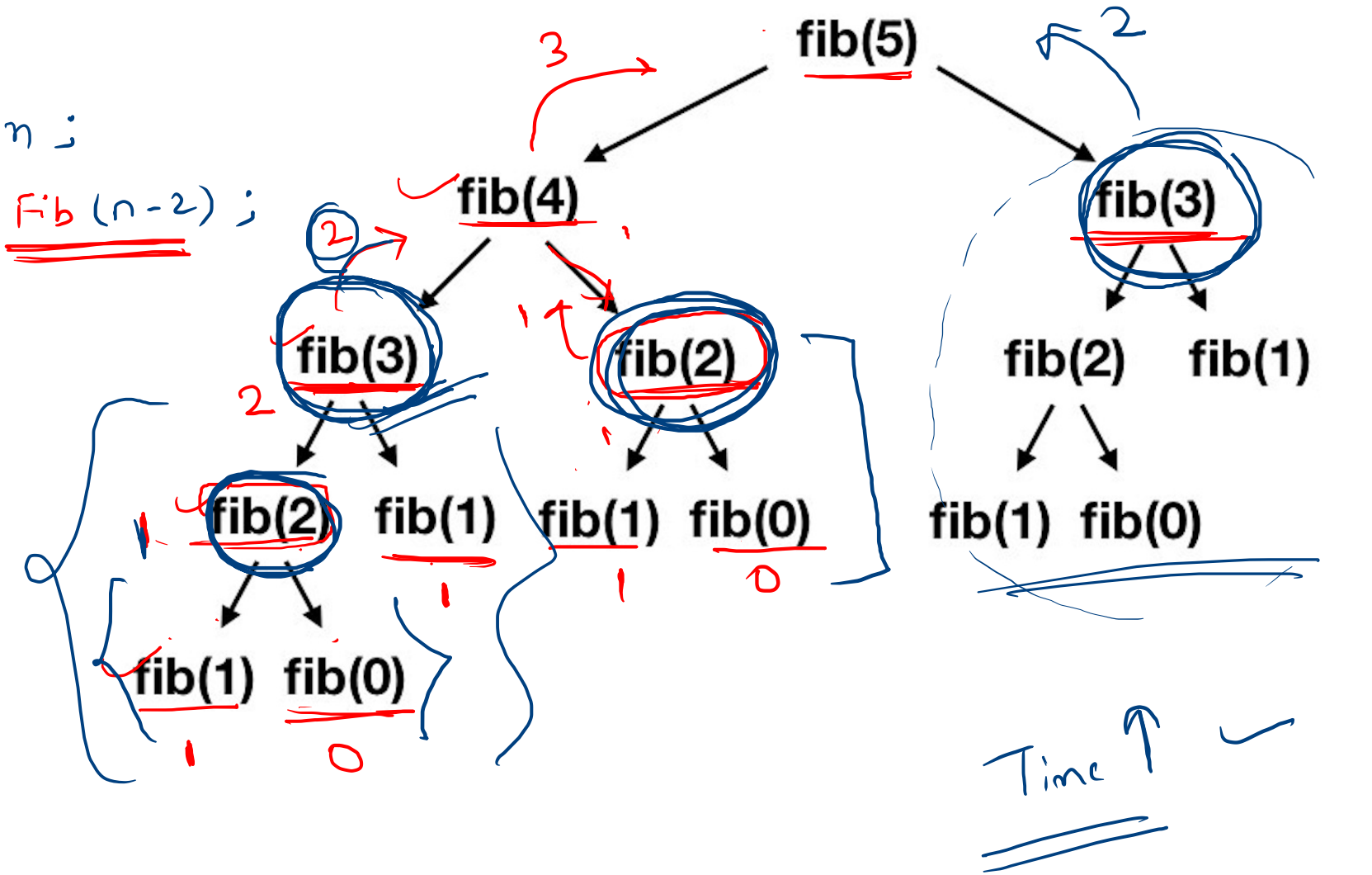
Ilp: N=5

Olp: 5 [1 1 2 3 5]

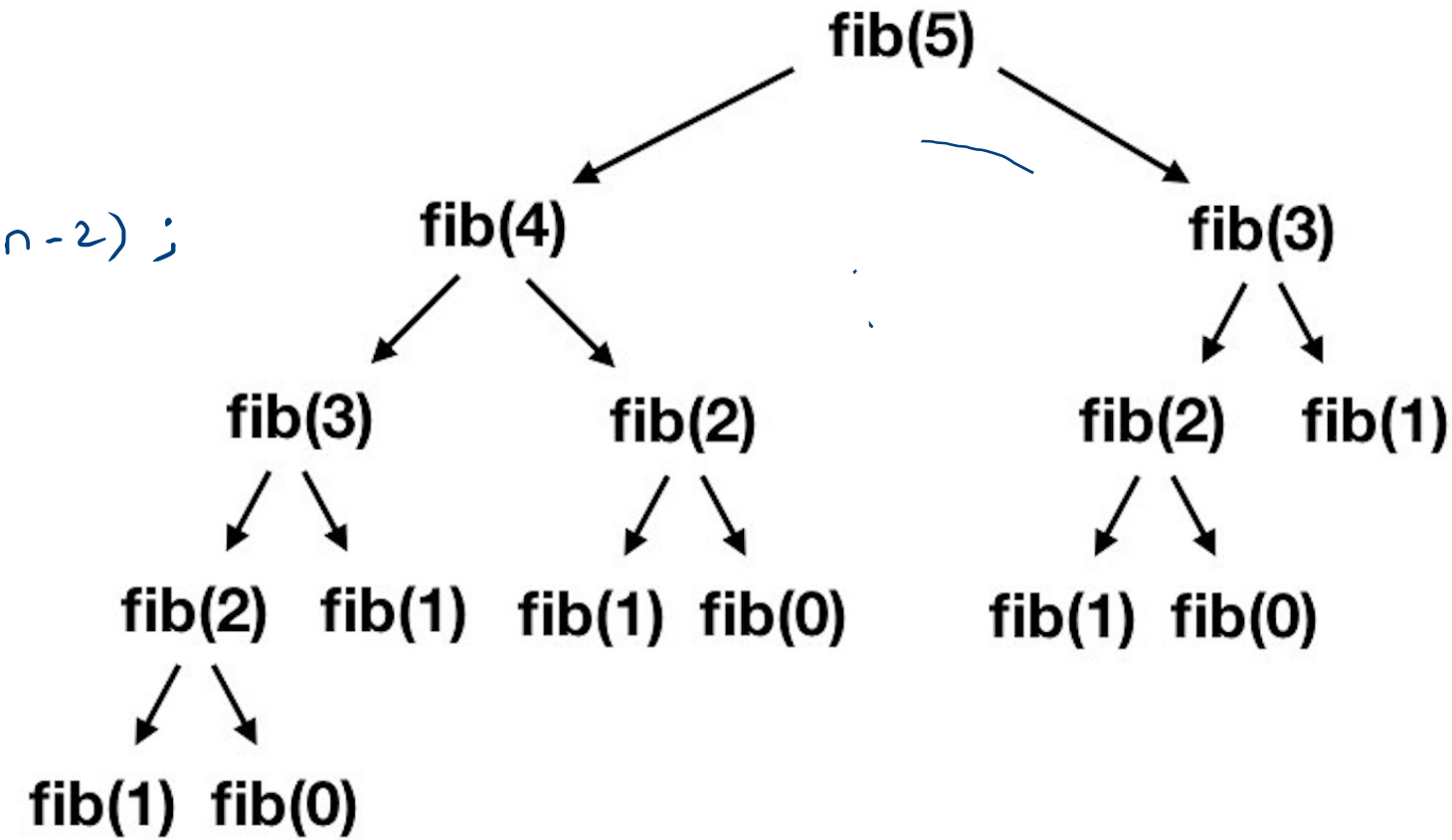
```
int Fib (int n)
{
    if (n<=1) return n;
    return Fib(n-1) + Fib(n-2);
}
```



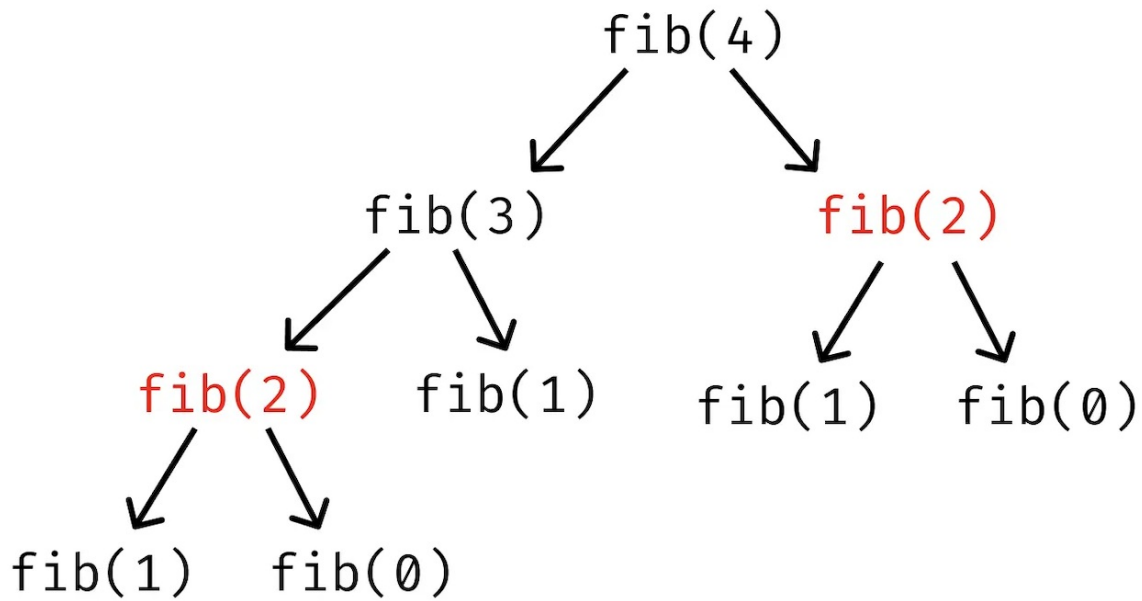
```
int Fib (int n)
{
    if (n <= 1) return n;
    return Fib(n-1) + Fib(n-2);
}
```



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

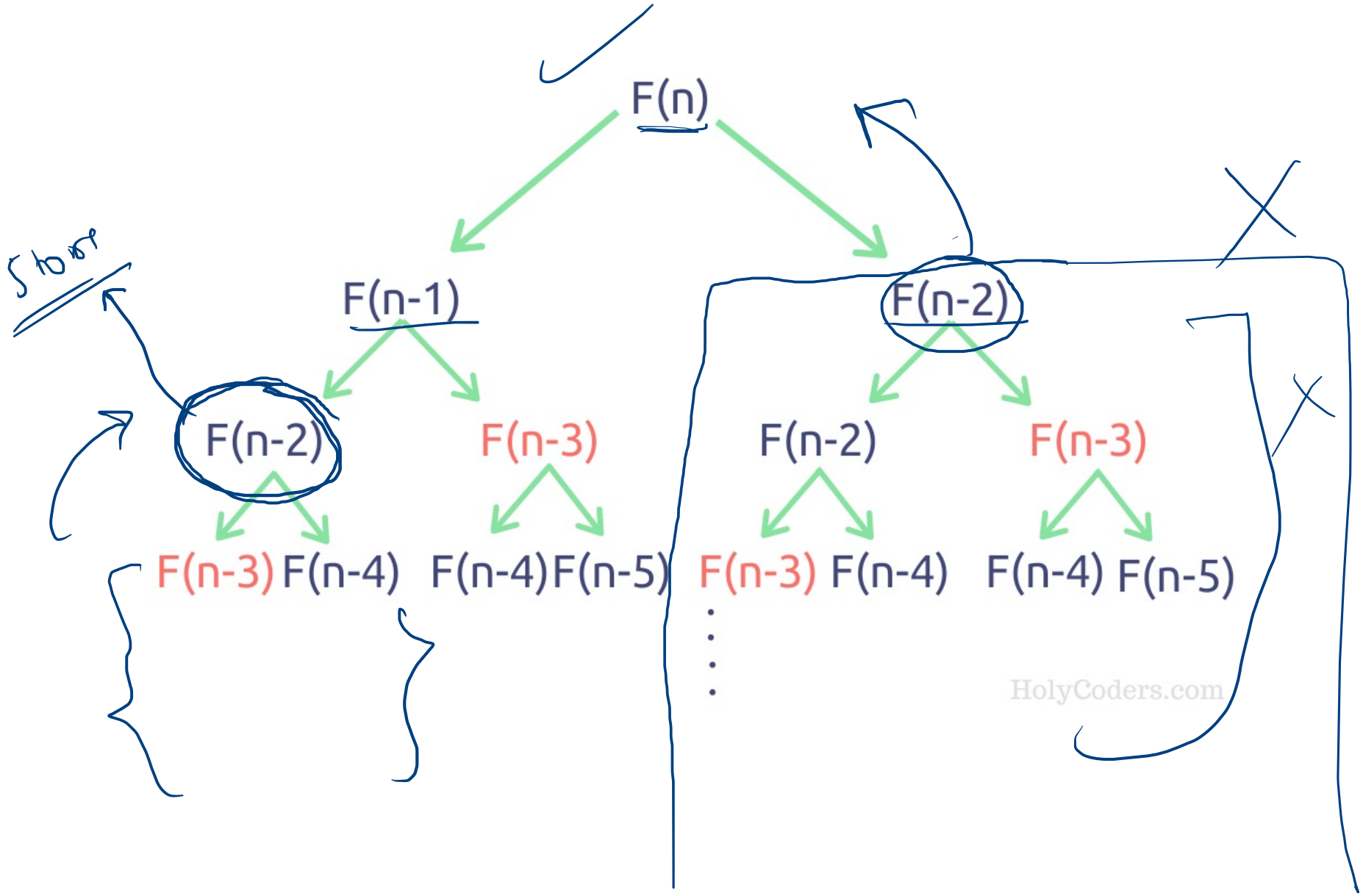


OPTIMIZING RECURSIVE FIBONACCI

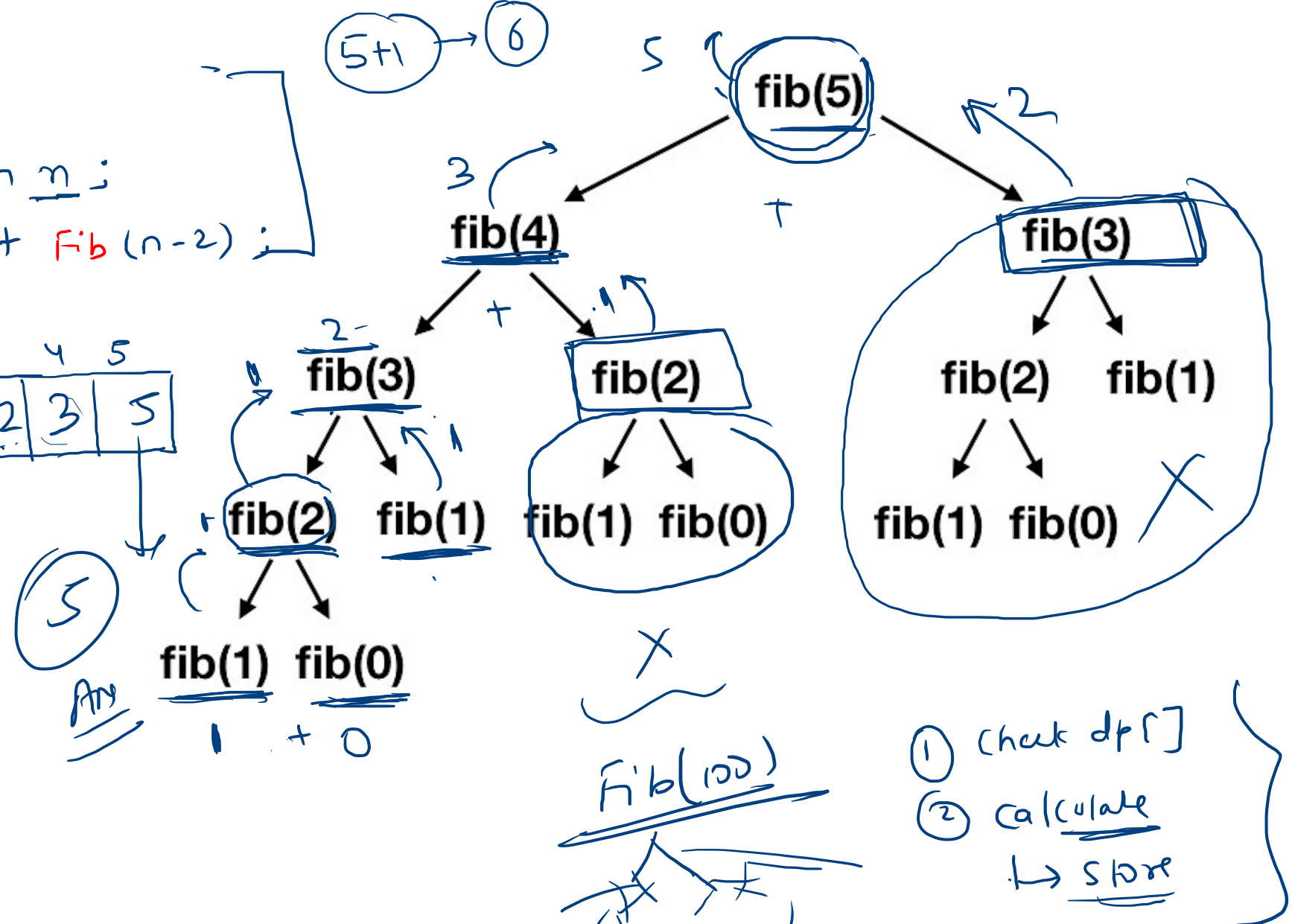


The larger Fibonnaci sequence has overlapping function call.

You can reduce the call with **memoization**


F(n)
F(n-1)
F(n-2)
F(n-2)
F(n-3)
F(n-2)
F(n-3)
F(n-3)
F(n-4)
F(n-4)
F(n-5)
F(n-3)
F(n-4)
F(n-4)
F(n-5)
HolyCoders.com

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



Fib(100)

① check dp[]
② calculate
↳ store

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

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

⟹

```
vector<int> dp(1001, -1);

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

↳ Memoization

Problem Statement
Solution
Recursive Solution
RC
time
95.1
Memoization
RC +
Array
Top down
dp[n][m]

Q How to identify a problem as DP ?

Ans : If there is a repetition of function call then we will think of DP solution.