Fibbonaci Series ⇒ $F(n) = F(n-1) + F(n-2)$

ii) Loop Method :-

$f_0 = f_1 = 1$ $\quad f_n = ?$

```
for i in range(2, n+1):-
    result = f0 + f1
    f0 = f1
    f1 = result
print(result)
```

Time Complexity for this is $O(n)$.

Why is this time complexity $O(n)$?

=> Because

$$T(n) = T(n-1) + c \quad -(1)$$

c → constant time for addition & assigning

In this we can substitute (2)

$$T(n-1) = T(n-2) + c \quad -(2)$$

$$T(n) = T(n-2) + 2c \quad -(3)$$

Now in (3) we can substitute (4)

$$T(n-2) = T(n-3) + c \quad -(4)$$

(4) in (3)

$$T(n) = T(n-3) + 3c \quad -(5)$$

So, we can say

$$T(n) = T(n-k) + k^{th}c \quad -(6)$$

we know $T(0)$ is 1, so we place $K=0$ in (6)

$$T(n) = T(0) + n^{\wedge}c$$

$$T(n) = n^{\wedge}c + 1$$

That's why $O(n)$. ✡

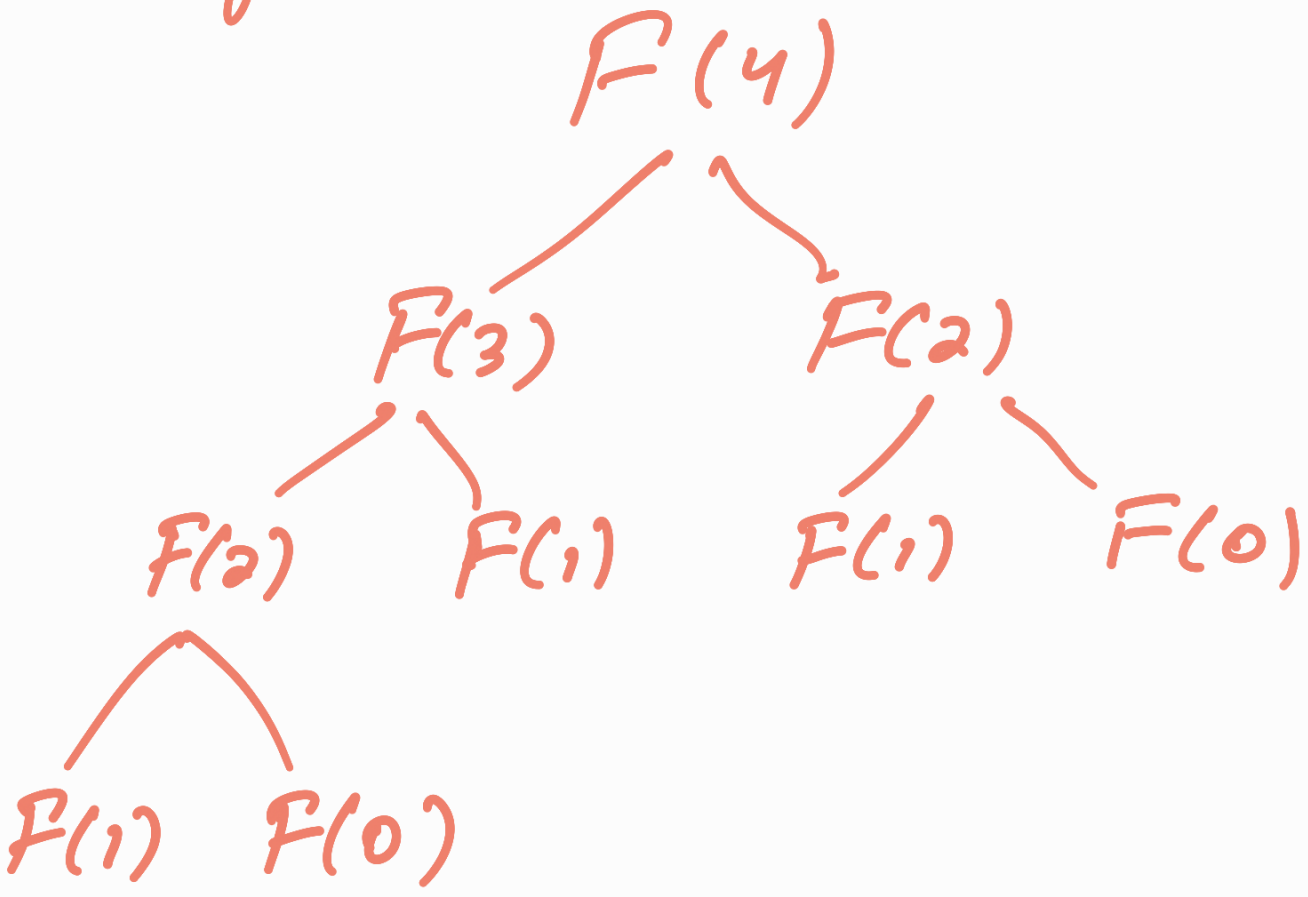
(iii) Recursion Method :-

```
def fibonnaci(n):-
    If (n<=1):-
        return 1
    else
        return fibonnaci(n-1)
               +
               fibonnaci(n-2)
```

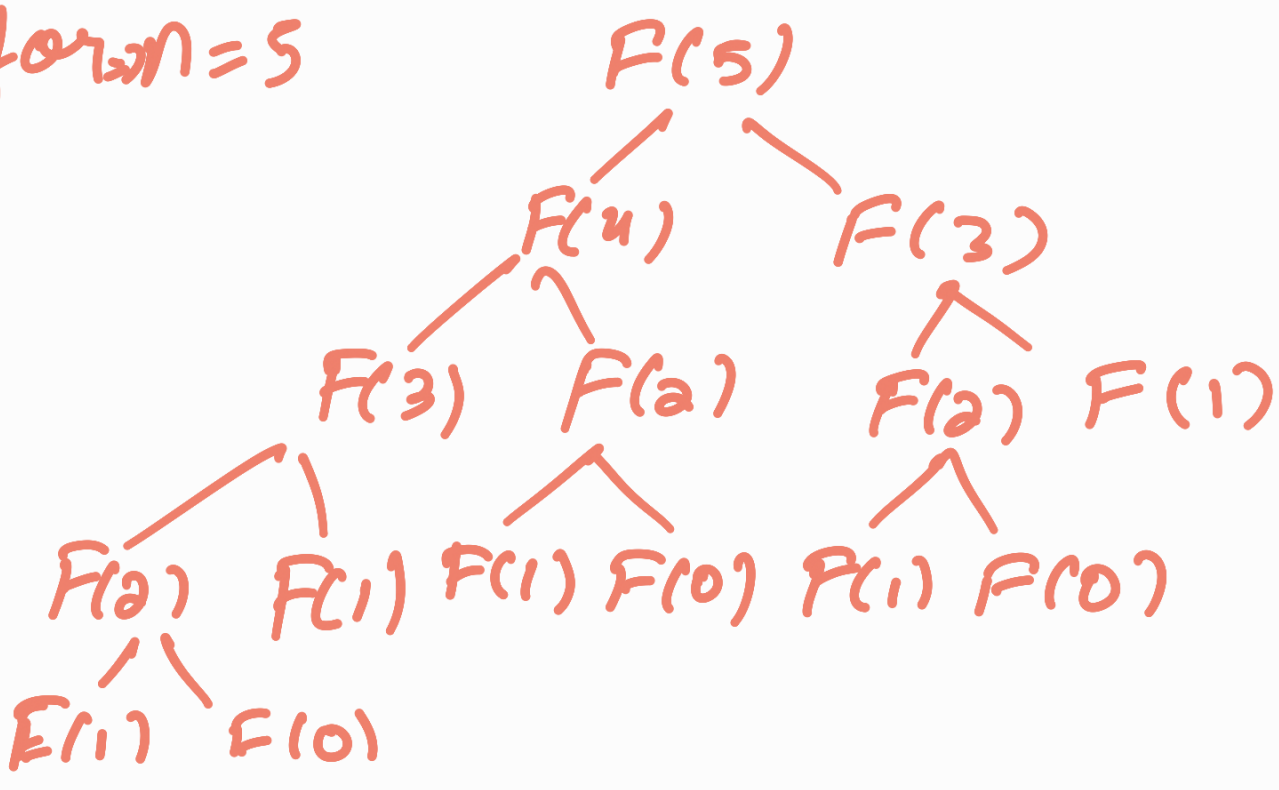
Time complexity is $O(2^n)$.

Why is it $O(2^n)$?

⇒ Let's take an example for $n = 4$



for $n = 5$



It is because

$$T(n) = T(n-1) + T(n-2) + C$$

Let's take $T(n-1) \approx T(n-2)$

$$T(n) = 2 * T(n-1) + C \quad -(1)$$

$$T(n-1) = 2 * T(n-2) + C \quad -(2)$$

(2) in (1)

$$T(n) = 2(2 * T(n-2) + C) + C$$

$$T(n) = 4\,T(n-2) + 3C \quad -(3)$$

$$T(n-2) = 2\,T(n-3) + C \quad -(4)$$

(4) in (3)

$$T(n) = 8\,T(n-3) + 7C \quad -(5)$$

$$T(n) = 2^K\,T(n-K) + (2^K - 1)C$$

$$-(6)$$

$T(0)=1$, we know so we place $K=n$ in (6)

$$T(n) = 2^n T(0) + (2^n - 1) C$$

$$T(n) = 2^n (C+1) - C$$

$$T(n) = 2^n (C_1) - C$$

$$\boxed{T(n) = O(2^n)}$$

(iii) Using Memoization

```
globaldict = {0:1, 1:1}
def fib(n):-
    If n in globaldict:
```

```
        return globaldict[n]
    else:-
        result= fib(n-1)+fib(n-2)
        globaldict[n]= result
        return result
```

Time complexity is O(n).

It is because we will calculate value a single time then we can get value directly from dictionary.