

2) Here, for implementation-1,

```
def fibonacci-1(n):
```

```
    if n <= 0: → O(1)
```

```
        print(" ")
```

```
    elif n <= 2: → O(1)
```

```
    else:
```

```
        return fibonacci-1(n-1) + fibonacci-1(n-2)
```

In the recursion, the time complexity will

be,

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

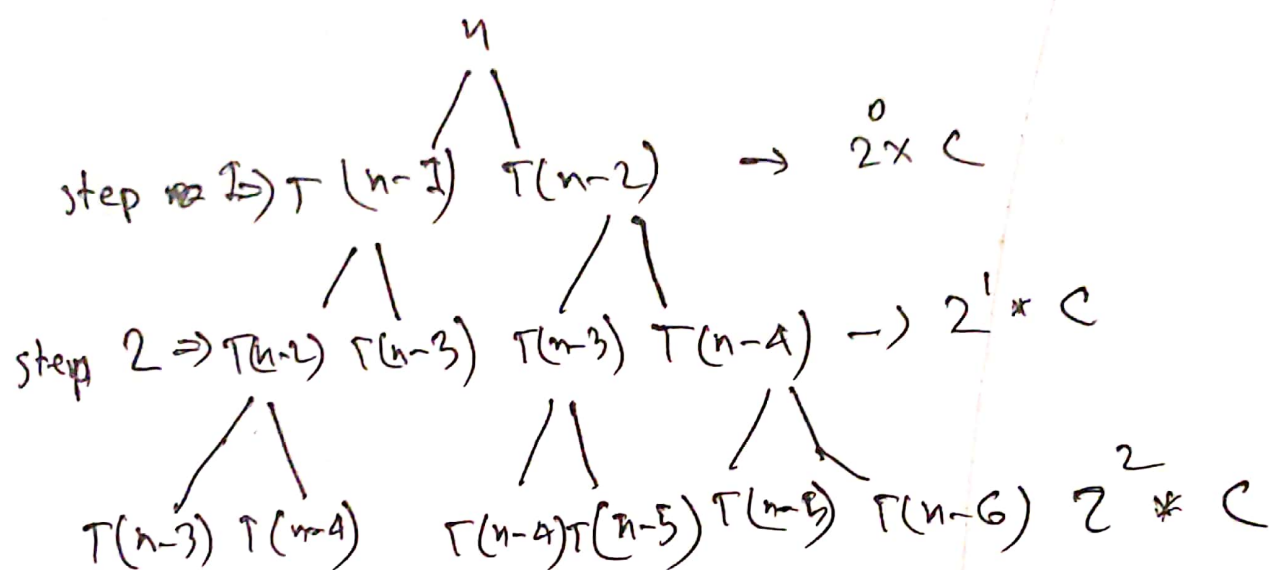
where, "C" is a constant.

PRO

Here in implementation - I,

$$T(n) = T(n-1) + T(n-2) + C \quad [\text{It is a recursive function}]$$

Hence,



$$\therefore T(n) \leq C + 2C + 2^2C + 2^3C + \dots + 2^{n-1} \times C$$

$$\text{or } T(n) \leq C(2^0 + 2^1 + 2^2 + 2^3 + \dots + 2^{n-1})$$

$$\leq C(2^{n-1} - 1)$$

$$\leq C \cdot 2^n \quad [C, \text{ and } 1 \text{ are constants}]$$

$$\therefore \text{Time complexity} = O(2^n).$$

## Implementation - 2

def fibo\_2(n):

    If  $n < 0!$   $\rightarrow 1$

        print(" ")

    elif  $n \leq 2!$   $\rightarrow 1$

        return n

    else:

        for i to (n)  $\rightarrow O(n)$

$$\therefore \text{Total} = O(n) + 1 + 1$$

$$= O(n)$$

[ Constants are  
being ignored.

$$\therefore \text{Qn } O(2^n) > O(n)$$

$\therefore$  Implementation - II is better.

4) Here,

def matrix\_matrix (A, B):

for i to n:  $\rightarrow O(n)$

for i to n:  $\rightarrow n$

for j to n:  $\rightarrow n^2 \rightarrow O(n^3)$

for k to n:  $\rightarrow n^3$

$\therefore$  Total time =  $O(n^3) + O(n^2) + O(n) + O(n)$

$\approx O(n^3)$

(Ans.)