Implementation -1 :

```
if n <=0 :          ——> O(1)
    Print ("Invalid Input")

elif n<=2:          ——> O(1)
    ~~Print~~
    return n-1

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

$\therefore T(n) = T(n-1) + T(n-2) + 1$

Let, $T(n-2) \simeq T(n-1)$

$\therefore T(n) = 2T(n-1) + c \quad [c = \text{constant}]$

$T(n-1) = 4T(n-2) + \cancel{c}\, 2c = 2^2 T(n-2) + (2^{2-1})c$

$\therefore T(n) = 2^k T(n-k) + (2^{k-1})c$

$= 2^n T(0) + (2^{n-1})c$

$= 2^n + (2^{n-1})c$

$= 2^n$

$\therefore$ Time Complexity $= O(2^n)$

Implementation-2:

```
fibonacci_array = [0, 1]  → O(1)
if n<0:  → O(1)
    Print("Invalid Input")
elif n<=2:  → O(1)
    return fibonacci_array[n-1]
else:
    for i in range(2,n):  → O(n)
        # atomics
```

$\therefore$ Time Complexity $= O(1) + O(1) + O(1) + O(n)$
$= O(n)$

For I we got $O(2^n)$ and for II we got $O(n)$ time complexity.

so, II is more efficient as, $O(n) < O(2^n)$

Given,

for $i = 0$ to $n-1$ $\rightarrow O(n)$

for $j = 0$ to $n-1$ $\rightarrow O(n)$

for $k = 0$ to $n-1$ $\rightarrow O(n)$

$\therefore$ The time complexity $= O(n^3)$

~~As~~ As there are 3 loops running for n - amount of time, the time complexity can be considered as $= O(n) \times O(n) \times O(n) = O(n^3)$