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,

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

where, "c" is a constant.

RTO

Here in implementation - I,

$T(n) = T(n-1) + T(n-2) + C$ [It is a recursive function]

Here, ,

n

step 1 $\Rightarrow$ $T(n-1)$ $T(n-2)$ $\rightarrow$ $2^0 \times C$

step 2 $\Rightarrow$ $T(n-2)$ $T(n-3)$ $T(n-3)$ $T(n-4)$ $\rightarrow$ $2^1 * C$

$T(n-3)$ $T(n-4)$ $T(n-4)$ $T(n-5)$ $T(n-5)$ $T(n-6)$ $2^2 * C$

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

or, $T(n) \le c(2^0 + 2^1 + 2^2 + 2^3 + \dots + 2^{n-1})$

or, $\le c(2^{n-1} - 1)$

$\le c\, 2^n$ [c, and 1 are constant]

$\therefore$ ~~Time~~ time complexity $= O(2^n)$.

Implementation - 2

def fibo_2 (n):

If n < 0: → 1
print (" ")

elif n<=2: → 1
return n

else:
for i to (n) → $O(n)$
... ..

∴ Total = $O(n)$ + 1 + 1
= $O(n)$ [∵ Constants are being ignored.

∴ As $O(2^n) > O(n)$

∴ Implementation - II is better.

4) Here,

def Matrix_matrix (A, B):

for ~~in~~ 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.)