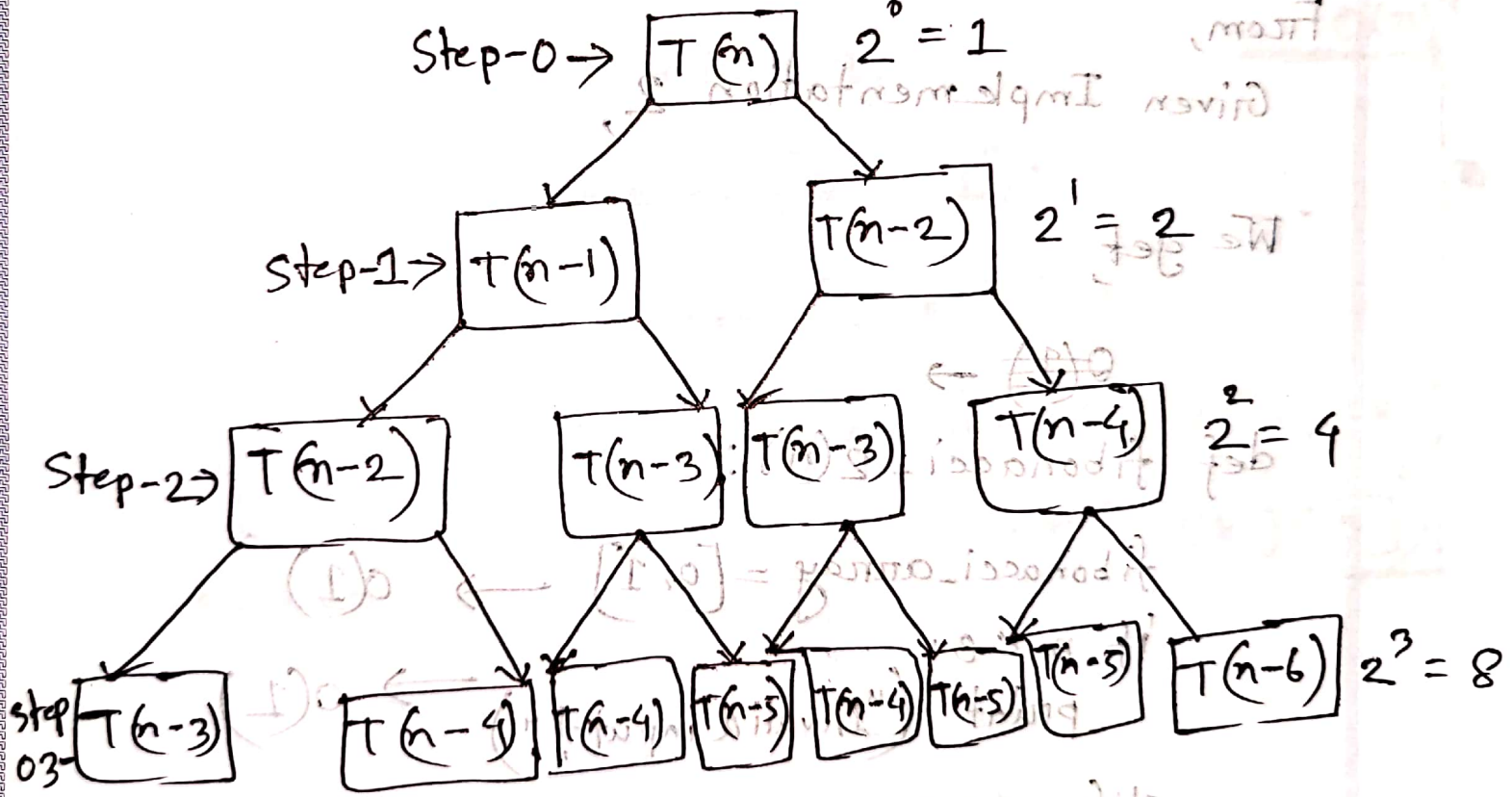
Ans. to the ques. no. 2

ID: 19101465

From,
Given Implementation 1,



STep-K → 2^K

Suppose, k = n, Then T(n-k) = T(n-n) = T(0)

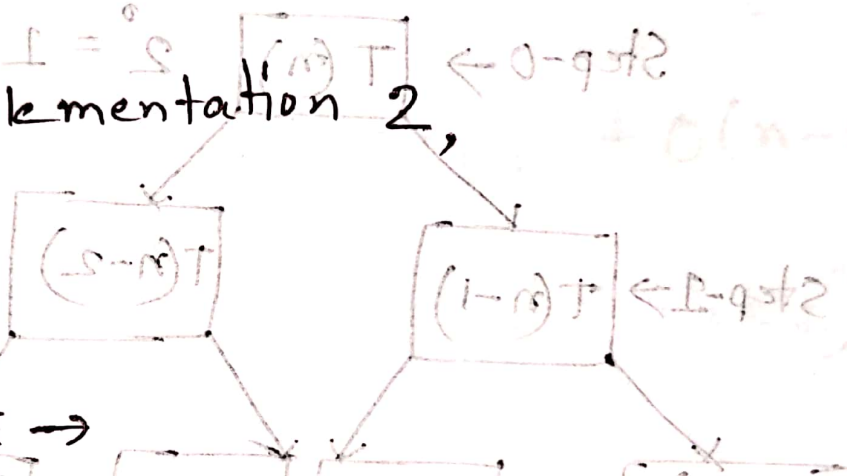
As. T(0) is a base case, T(0) gives 1 operation

∴ Total operation = $2^0 + 2^1 + 2^2 + 2^3 + \dots 2^n = 2^{n+1} - 1$

So, Time complexity for Implementation 1:

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

From,
Given Implementation 2,



We get,

~~O(1)~~ →

```
def fibonacci_2(n):
    fibonacci_array = [0, 1]  ⟶ O(1)
    if n<0:
        print("Invalid input!") ⟶ O(1)
    elif n<=2:
        return fibonacci_array[n-1] ⟶ O(1)
    else:
        for i in range(2, n): ⟶ O(n-2)
            fibonacci_array.append(fibonacci_array
            [i-1]+fibonacci_array[i-2]
        return fibonacci_array[-1]
```

```
n = int(input("Enter a number: "))
nth_fib = fibonacci-2(n)
print("The %d-th fibonacci number
        is %d" %(n, nth_fib))
```

∴ Time complexity = $O(1) + O(n-2)$

$= O(1) + O(n-2)$

$= O(n-2)$

$= O(n)$

Ans. to the ques. no. - 4

```
def Multiply_matrix (A, B):
    global n
    C = [[0] * n for i in range (n)]
    for i in range (0, n, 2):
        for j in range (0, n, 2):
            for k in range (0, n, 2):
                C[i][j] += int(A[i][k]) *
                           int(B[k][j])
    return C
```

$O(n)^3$

∴ Time complexity = $O(n)^3$

$= O(n)^3$