② Two different implementations of finding the n^{th} fibonacci number are given.

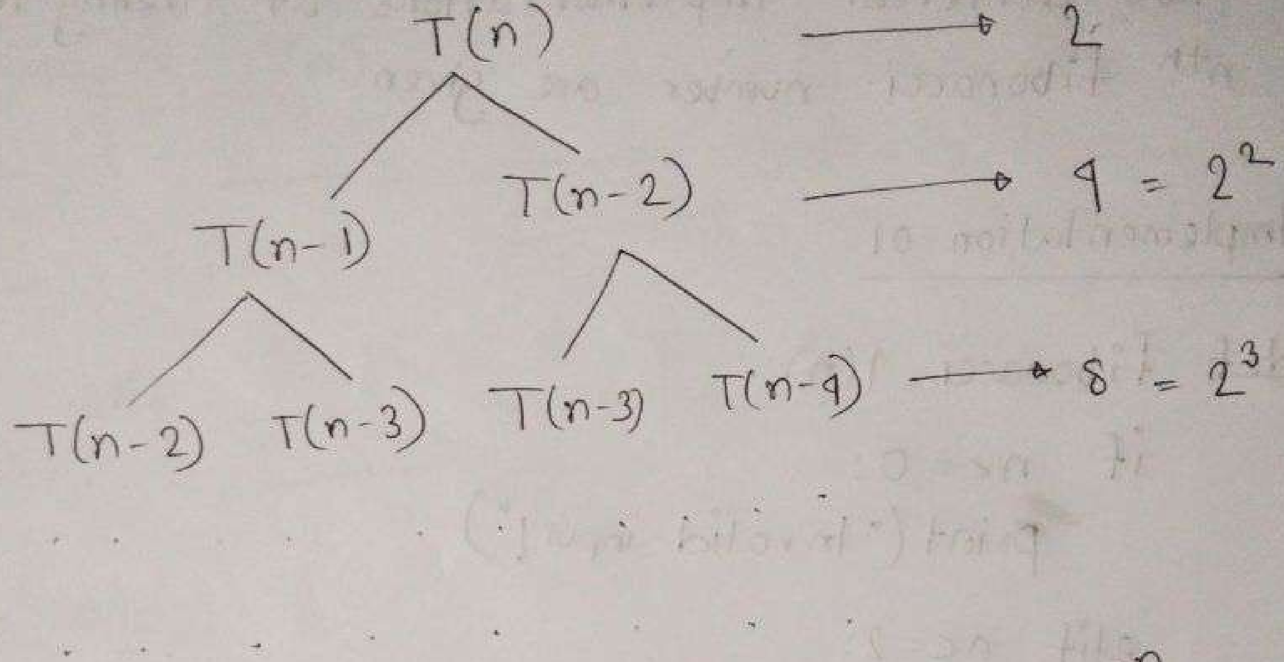
Implementation - 01

```
def fibonacci_1(n):
    if n<=0:                                    ....1
        print("Invalid input!")
    elif n<=2:                                  ....1
        return n-1
    else:
        return fibonacci_1(n-1) + fibonacci_1(n-2)
```

For the recursive case, $T(n) = T(n-1) + T(n-2) + 2$

For the base case, $T(2) = 0$

Now, $T(n) = T(n-1) + T(n-2) + 2$



After n steps $\longrightarrow 2^n$

Adding the constants,

$$T(n) = 2^1 + 2^2 + 2^3 + 2^4 + \cdots + 2^n$$

$$= 2^{n+1} - 1$$

$$= 2^n . 2 - 1$$

$$\therefore T(n) = O(2^n)$$

Implementation - 02

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

For this implementation, $f(n) = 1 + 1 + 1 + n - 2$

$$= 3 + n - 2$$

$$= n + 1$$

So, for this case, $\cancel{\theta(n) = n}$ $f(n) = O(n)$

In Implementation - 1, recursion is used to solve the problem. But for this particular problem, the same sub-problem is counted multiple times for which it became slower than the 2nd implementation. In the 2nd implementation, due to use of iteration, ~~each~~ it is not divided into sub problems. So, 2nd implementation is faster than implementation 1.

⑤ Finding worst case complexity

$\boxed{1}$ $T(n) = T(n/2) + n - 1$, $T(1) = 0$

Using recursive tree,

$T(n) \longrightarrow n-1$

$T(n/2) \longrightarrow \frac{n-1}{2}$

$T(n/4)$ i.e. $T(n/2^2) \longrightarrow \frac{n-1}{2^2}$

$T(n/8)$ i.e. $T(n/2^3) \longrightarrow \frac{n-1}{2^3}$

. . . . . .

after k steps $\longrightarrow \frac{n-1}{2^k}$

Adding the constants,

$$T(n) = n-1 + \frac{n-1}{2} + \frac{n-1}{2^2} + \frac{n-1}{2^3} + \dots + \frac{n-1}{2^k}$$

$$= (n-1)\left(\frac{1}{2} + \frac{1}{2^2} + \dots + \frac{1}{2^k}\right)$$

$$= (n-1) \cdot 1$$

$$= n-1$$

$$= O(n)$$ [neglecting the constant]

$\boxed{2}$ $T(n) = T(n-1) + n - 1$; $T(1) = 0$

Using recursive tree,

$T(n) \longrightarrow n-1$

$T(n-1) \longrightarrow n-2$

$T(n-2) \longrightarrow n-3$

$\vdots$

$T(1) \longrightarrow 0$

$$\therefore T(n) = n-1 + n-2 + n-3 + \ldots + 0$$

$$= 0 + 1 + 2 + \ldots + n-1$$

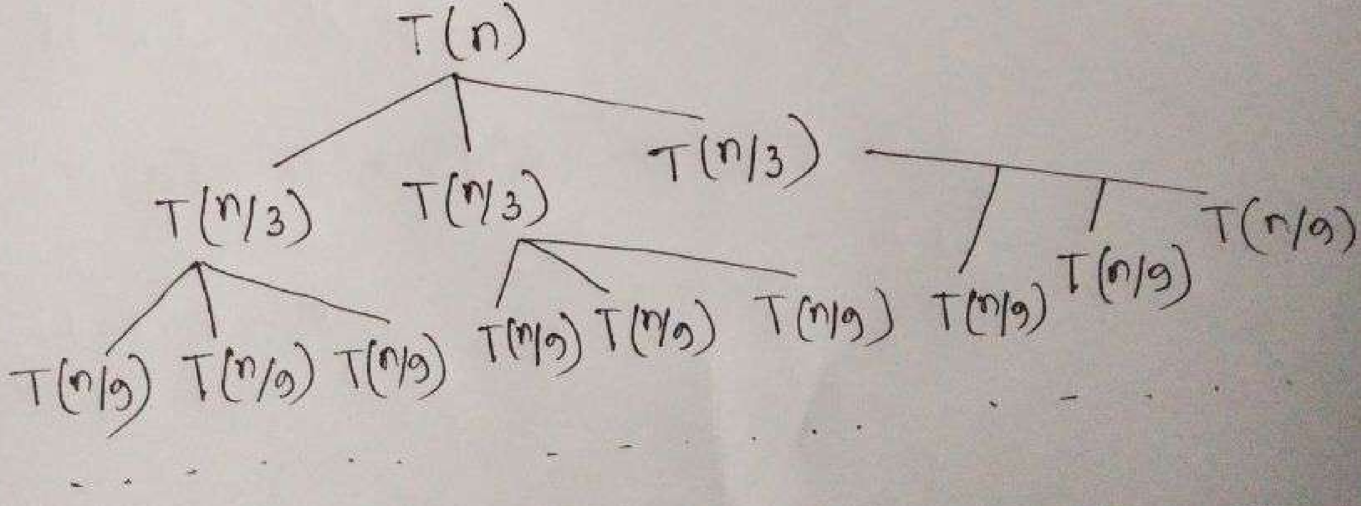
$$= \frac{(n-1)(n-1+1)}{2}$$

$$= \frac{(n-1)n}{2}$$

$$= \frac{n^2 - n}{2}$$

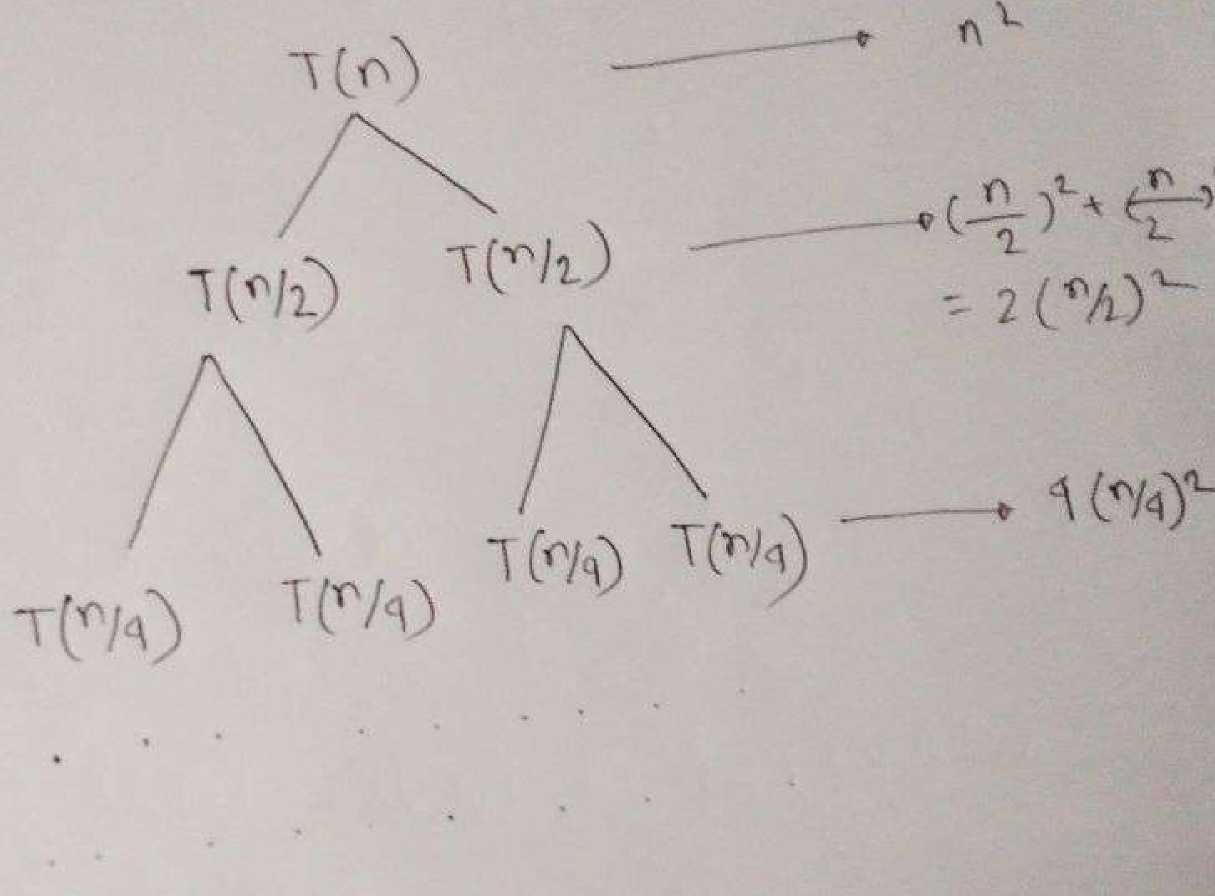
$$= O(n^2)$$

$\boxed{3}$ $T(n) = T(n/3) + 2T(n/3) + n = 3T(n/3) + n$



$\boxed{4}$ $T(n) = 2T(n/2) + n^2$; prove $\cancel{\theta(n^2)}$ $T(n) = O(n^2)$

Using recursive tree,



So, $T(n) = n^2 + 2(n/2)^2 + 4(n/4)^2 + \cdots$

$= n^2 + 2 \cdot \frac{n^2}{4} + 4 \cdot \frac{n^2}{16} + \cdots$

$= n^2 + \frac{n^2}{2} + \frac{n^2}{4} + \cdots$

$= n^2 (1 + \frac{1}{2} + \frac{1}{4} + \cdots) = n^2 \cdot 1 = n^2$

$\therefore T(n) = n^2$ [Proved]