Recursive Function

```
FUNCTION Fibonacci(n)
    IF n <= 0 THEN
        RETURN "Input should be positive integer"
    ELSE IF n IS 1 THEN
        RETURN 0
    ELSE IF n IS 2 THEN
        RETURN 1
    ELSE
        RETURN Fibonacci(n-1) + Fibonacci(n-2)
    END IF
END FUNCTION
```

```
FUNCTION Fibonacci(n)
    IF n <= 0 THEN
        RETURN "Input should be positive integer"
    ELSE IF n IS 1 THEN
        RETURN 0
    ELSE IF n IS 2 THEN
        RETURN 1
    ELSE
        DECLARE a = 0
        DECLARE b = 1
        DECLARE temp
        FOR i FROM 2 TO n
```

```
            temp = a + b
            a = b
            b = temp
        END FOR
        RETURN b
    END IF
END FUNCTION
```

Recursive Approach: The time complexity of the recursive approach is O(2^n). This is because each function call branches into two new calls in the recursion tree. However, this approach involves a lot of repeated calculations which makes it inefficient for large inputs.

Iterative Approach: The time complexity of the iterative approach is O(n). This is because it uses a simple loop to calculate the nth Fibonacci number, and does not involve any repeated calculations. Therefore, it is much more efficient for large inputs.

In terms of space complexity, the recursive approach has a space complexity of O(n) due to the maximum depth of the recursion tree, while the iterative approach has a constant space complexity of O(1), as it only requires a fixed amount of space to store the variables.

So, in conclusion, the iterative approach is more efficient than the recursive approach for calculating the nth Fibonacci number, both in terms of time and space complexity. It’s always important to consider these factors when choosing an algorithm to solve a problem.