**Day2 : Assignment 2**

Recursive Function and Efficiency Analysis - Write a recursive function pseudocode and calculate the nth Fibonacci number and use Big O notation to analyze its efficiency. Compare this with an iterative approach and discuss the pros and cons in terms of space and time complexity.

**Recursive Function**

function fibonacci\_recursive(n)

if n == 0

return 0

elif n == 1

return 1

else

return fibonacci\_recursive(n-1) + fibonacci\_recursive(n-2)

**Explanation:**

1. The function takes an integer n as input, representing the desired position in the Fibonacci sequence.
2. It has two base cases:

If n is 0, the function returns 0 (the first Fibonacci number).

If n is 1, the function returns 1 (the second Fibonacci number).

1. For any other value of n, the function makes two recursive calls:

One call with n-1 to get the (n-1)th Fibonacci number.

Another call with n-2 to get the (n-2)th Fibonacci number.

1. Finally, it returns the sum of these two recursive calls, which represents the nth Fibonacci number according to the property of the Fibonacci sequence (F(n) = F(n-1) + F(n-2)).

**Efficiency Analysis (Recursive):**

1. Time Complexity: O(2^n). This is because for each n, the function makes two recursive calls, leading to exponential growth in the number of calculations required as n increases.
2. Space Complexity: O(n). The space complexity arises from the function call stack, which grows with the depth of recursion (which is proportional to n in this case).

**Iterative Approach**

function fibonacci\_iterative(n)

a, b = 0, 1

for i in range(n):

a, b = b, a + b

return a

**Explanation:**

1. The function takes an integer n as input.
2. It initializes two variables, a and b, with the first two Fibonacci numbers (0 and 1).
3. It uses a loop that iterates n times.
4. Inside the loop, it performs the following steps in each iteration:

Stores the current value of b in a.

Updates b with the sum of a (previous value) and b (current value), effectively calculating the next Fibonacci number.

1. After the loop completes, the function returns the value stored in a, which holds the nth Fibonacci number.

**Efficiency Analysis (Iterative):**

Time Complexity: O(n). The loop iterates exactly n times, resulting in linear growth in the number of calculations required as n increases.

Space Complexity: O(1). The space complexity is constant as it uses only a fixed number of variables regardless of the input n.

**Comparison:**

**Recursion:**

1. Pros: Simpler to understand and reflects the recursive nature of the Fibonacci sequence definition.
2. Cons: Exponentially slower for larger n due to redundant calculations and function call stack overhead.

**Iteration:**

1. Pros: More efficient for larger n as it avoids redundant calculations and has constant space complexity.
2. Cons: Might be less intuitive for beginners compared to the recursive approach.

*In conclusion, while recursion offers a conceptually elegant solution, the iterative approach is generally preferred for calculating Fibonacci numbers due to its superior efficiency, especially for larger inputs.*