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.

1. Fibonacci Number using recursive function

function fibonacci(n)

if n <= 1

return n

else

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

Time Complexity:

In the worst case, each call to fibonacci(n) results in two recursive calls: fibonacci(n-1) and fibonacci(n-2).

The time complexity can be expressed as O(2^n), where n is the input number.

This is because the number of function calls grows exponentially with the input size.

Space Complexity:

The space complexity of the recursive approach is O(n) due to the recursive calls stored in the call stack.

In each recursive call, space is allocated in the stack to store the function arguments and local variables.

def fibonacci\_iterative(n):

if n <= 1:

return n

else:

a, b = 0, 1

for \_ in range(2, n + 1):

a, b = b, a + b

return b

Time Complexity:

The time complexity of the iterative approach is O(n).

It involves iterating through the sequence once to compute the nth Fibonacci number.

Space Complexity:

The space complexity of the iterative approach is O(1).

It only requires a constant amount of space for storing the variables a and b.

Pros and Cons:

Recursive Approach:

Pros:

Simple and intuitive implementation.

Cons:

Exponential time complexity makes it inefficient for large values of n.

Consumes more memory due to the overhead of function calls.

Iterative Approach:

Pros:

Linear time complexity makes it efficient, especially for large values of n.

Consumes less memory due to its iterative nature.

Cons:

May be less intuitive for some programmers compared to the recursive approach.