# Assignment 2 :

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

**pseudocode for a recursive Fibonacci function**

function fibonacci(n):

a = 0

b = 1

if n <= 1:

return n

for i = 2 to n:

temp = a + b

a = b

b = temp return b

Big O notation. In the recursive Fibonacci function, each call to fibonacci(n) results in two more calls to fibonacci(n-1) and fibonacci(n-2), and this branching continues until reaching the base cases. Therefore, the time complexity can be expressed as:

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

# Algorithm :

1. Start
2. Initialize two variables a and b with values 0 and 1, respectively.
3. Input value n is less than or equal to 1. If it is, return n itself since Fibonacci numbers for n=0 and n=1 are 0 and 1
4. If n is greater than 1, enter a loop that iterates from 2 to n. This loop is used to calculate the Fibonacci number for values greater than 1.
5. Inside the loop: a. Calculate the sum of a and b and store it in a temporary variable temp. b. Update the value of a to be the current value of b. c. Update the value of b to be the current value of temp.
6. After the loop completes, return the value of b, which represents the nth Fibonacci number.

Comparing the two approaches:

**Recursive approach:**

Pros: Simple implementation.

Cons: Exponential time complexity, high memory overhead due to recursive function calls, and potential for stack overflow with large input values.

Iterative approach:

Pros: Linear time complexity, no risk of stack overflow.

Cons: Slightly more complex implementation compared to the recursive approach.

In terms of space complexity, both approaches use O(1) space for storing temporary variables, but the recursive approach has an additional overhead due to the recursive function calls, potentially leading to a stack overflow for large inputs. Therefore, the iterative approach is generally preferred for calculating Fibonacci numbers efficiently.

**CODE:**

#include <stdio.h>

int fibonacci\_iterative(int n) { int a = 0, b = 1, temp, i;

if (n <= 1) return n;

for (i = 2; i <= n; i++) { temp = a + b;

a = b;

b = temp;

}

return b;

}

int main() { int n;

printf("Enter the value of n to calculate the nth Fibonacci number: "); scanf("%d", &n);

if (n < 0) {

printf("Invalid input! n must be a non-negative integer.\n"); return 1;

}

int result = fibonacci\_iterative(n);

printf("The %dth Fibonacci number is: %d\n", n, result); return 0;

}

FLOW CHART:

