**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.**

Algorithm for recursion function:

fibonacci(n):

1. Start

2. If n is 0, return 0

3. If n is 1, return 1

4. Otherwise, return fibonacci(n - 1) + fibonacci(n - 2)

5. End

Psedocode:

fibonacci(n):

Begin

If n is 0:

Return 0

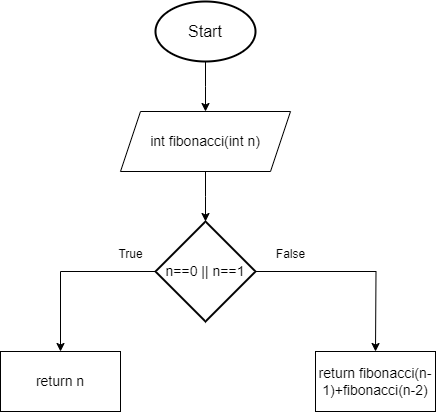
If n is 1:

Return 1

Return RecursiveFibonacci(n - 1) + RecursiveFibonacci(n - 2)

End

Flowchart:



Comparision with an iterative approach along with Big O notation analysis:

int fibonacci\_recursive(int n) {

if (n == 0)

return 0;

else if (n == 1)

return 1;

// Recursive call

else

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

}

The time complexity of the recursive Fibonacci function is O(2^n) because, for each value of 'n', the function makes two recursive calls (one for n - 1 and another for n - 2). This results in an exponential growth in the number of function calls as 'n' increases.

On the other hand, the space complexity of the recursive Fibonacci function is O(n) due to the recursive calls consuming stack space proportional to the depth of recursion, which can grow linearly with 'n'.

Comparision with an iterative approach:

// Iterative function to calculate the nth Fibonacci number

int fibonacci\_iterative(int n) {

int a = 0, b = 1, c;

if (n == 0)

return a;

for (int i = 2; i <= n; i++) {

c = a + b;

a = b;

b = c;

}

return b;

}

The time complexity of the iterative Fibonacci function is O(n) because it iterates 'n' times to calculate the nth Fibonacci number.

In terms of space complexity, the iterative approach has a constant space complexity of O(1) because it only uses a constant amount of space regardless of the input size.

Comparing the two approaches:

- Recursive approach:

- Pros: Simple implementation.

- Cons: Exponential time complexity and higher space complexity.

- Iterative approach:

- Pros: Linear time complexity and constant space complexity.

- Cons: More complex implementation.

In summary, while the recursive approach may be simpler to implement, it suffers from poor performance for large values of 'n' due to its exponential time complexity for this function. The iterative approach, on the other hand, offers better performance with linear time complexity and constant space complexity.