# 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.

Function fibonacci(n):

1. If n is 0, return 0.

2. If n is 1, return 1.

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

```

Now, let's analyze its efficiency using Big O notation:

- Time Complexity: The time complexity of the recursive Fibonacci function is O(2^n). This is because each call branches into two more calls, resulting in an exponential number of function calls. Therefore, the time complexity grows exponentially with the input size.

- Space Complexity: The space complexity of the recursive Fibonacci function is O(n). This is because it requires memory to store the recursive function calls on the call stack, up to a depth of n. Therefore, the space complexity grows linearly with the input size.

Now, let's compare it with an iterative approach:

```

Function fibonacciIterative(n):

1. Initialize variables "a" and "b" to 0 and 1, respectively.

2. Loop from i = 2 to n:

2.1. Set "temp" to "a + b".

2.2. Set "a" to "b".

2.3. Set "b" to "temp".

3. Return "b".

```

Now, let's analyze the efficiency of the iterative approach:

- Time Complexity: The time complexity of the iterative Fibonacci function is O(n). This is because it iterates n times through a loop, performing constant-time operations in each iteration. Therefore, the time complexity grows linearly with the input size.

- Space Complexity: The space complexity of the iterative Fibonacci function is O(1). This is because it only requires a constant amount of additional memory to store the variables "a", "b", and "temp". Therefore, the space complexity is constant regardless of the input size.

Pros and Cons:

Recursive Approach:

- Pros:

- It's simple and easy to understand.

- Cons:

- It has exponential time complexity, making it inefficient for large values of n.

- It has linear space complexity, but the large number of function calls can lead to stack overflow errors for large values of n.

Iterative Approach:

- Pros:

- It has linear time complexity, making it efficient for large values of n.

- It has constant space complexity, making it more memory-efficient compared to the recursive approach.

- Cons:

- It may be less intuitive to understand for some programmers compared to the recursive approach.

`

the recursive approach is simpler, it suffers from poor time complexity and may encounter stack overflow errors for large inputs. On the other hand, the iterative approach is more efficient in terms of both time and space complexity, making it a better choice for calculating Fibonacci numbers for large values of n.