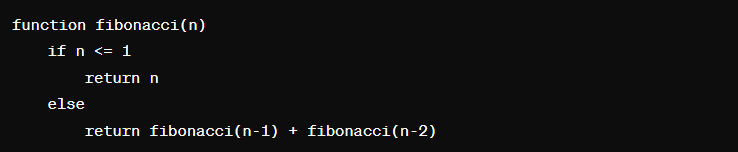
**Assignment 2.2**

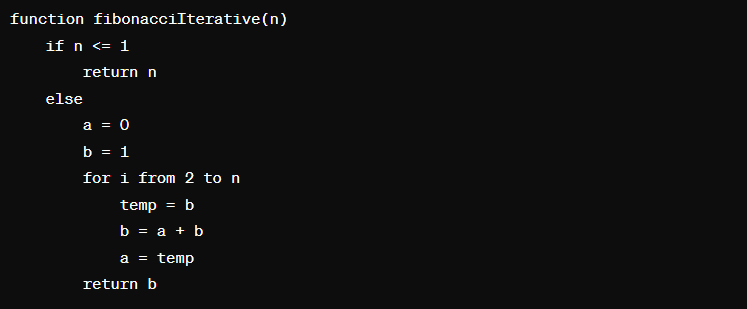
**Pseudocode:**



**Explanation:**

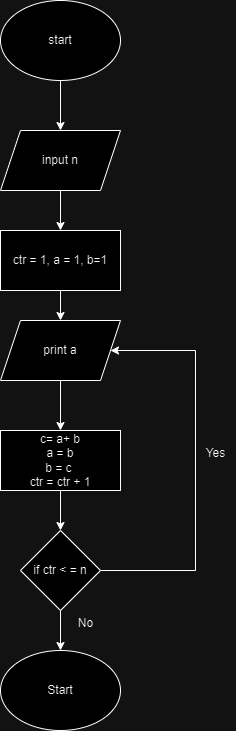
let's analyze its efficiency using Big O notation. The time complexity of this recursive approach is O(2^n), where n is the input number. This is because each call to fibonacci(n) results in two recursive calls (fibonacci(n-1) and fibonacci(n-2)), leading to an exponential growth in the number of function calls as n increases.

**Comparing with iterative approach:**

****

The time complexity of the iterative approach is O(n) because it involves a single loop that iterates n times to calculate the Fibonacci number, making it much more efficient than the recursive approach, especially for large values of n.

In terms of space complexity, the recursive approach requires O(n) space on the function call stack due to the recursive calls, while the iterative approach only requires O(1) space as it uses a constant amount of memory regardless of the input value.

**Flowchart:**