

INTRODUCTION TO ALGORITHMS(EC351)

ASSIGNMENT-I

Consider the Fibonacci series and solve the following.

Where, $\text{fib}(k) = \text{fib}(k-1) + \text{fib}(k-2)$

- Draw the flow chart, algorithms in pseudo code for solving $\text{fib}(5)$, $\text{fib}(10)$, $\text{fib}(100)$ and $\text{fib}(500)$.
- Find out total memory or space required to perform these Fibonacci series computational operations.
- Find out worst case and best case scenarios from the above identified approaches.
- Write a program to compare the actual memory consumed by all the approaches.

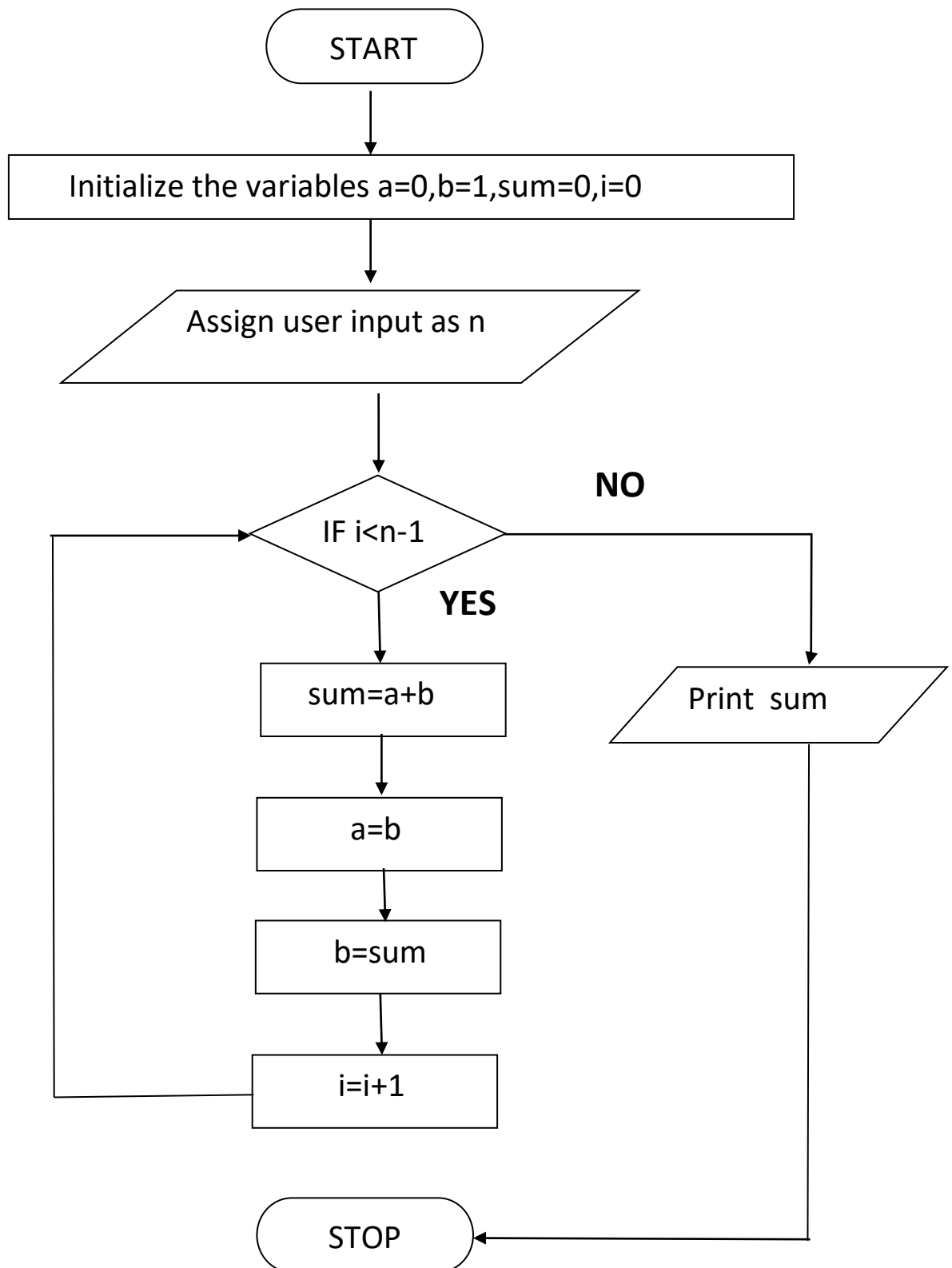
Solution:

Let us consider the input as 'n' for which we need to find the element at nth index in fibonacci series.

We can find the value of nth element in the Fibonacci series ($\text{fib}(n)$) in two ways.

1.Non-recursive method:

Flow chart:



Algorithm:

Step 1: START

Step 2: Initialize three variables $a=0$, $b=1$ and $sum=0$

Step 3: Assign the user input to n

Step 4: Initialize $i=0$ which acts as a counter variable

Step 5: If i is greater than $n-2$ then, go to Step 10

Step 6: $sum=a+b$

Step 7: $a=b$

Step 8: $b=sum$

Step 9: $i=i+1$, go to Step 5

Step 10: Print the value of sum

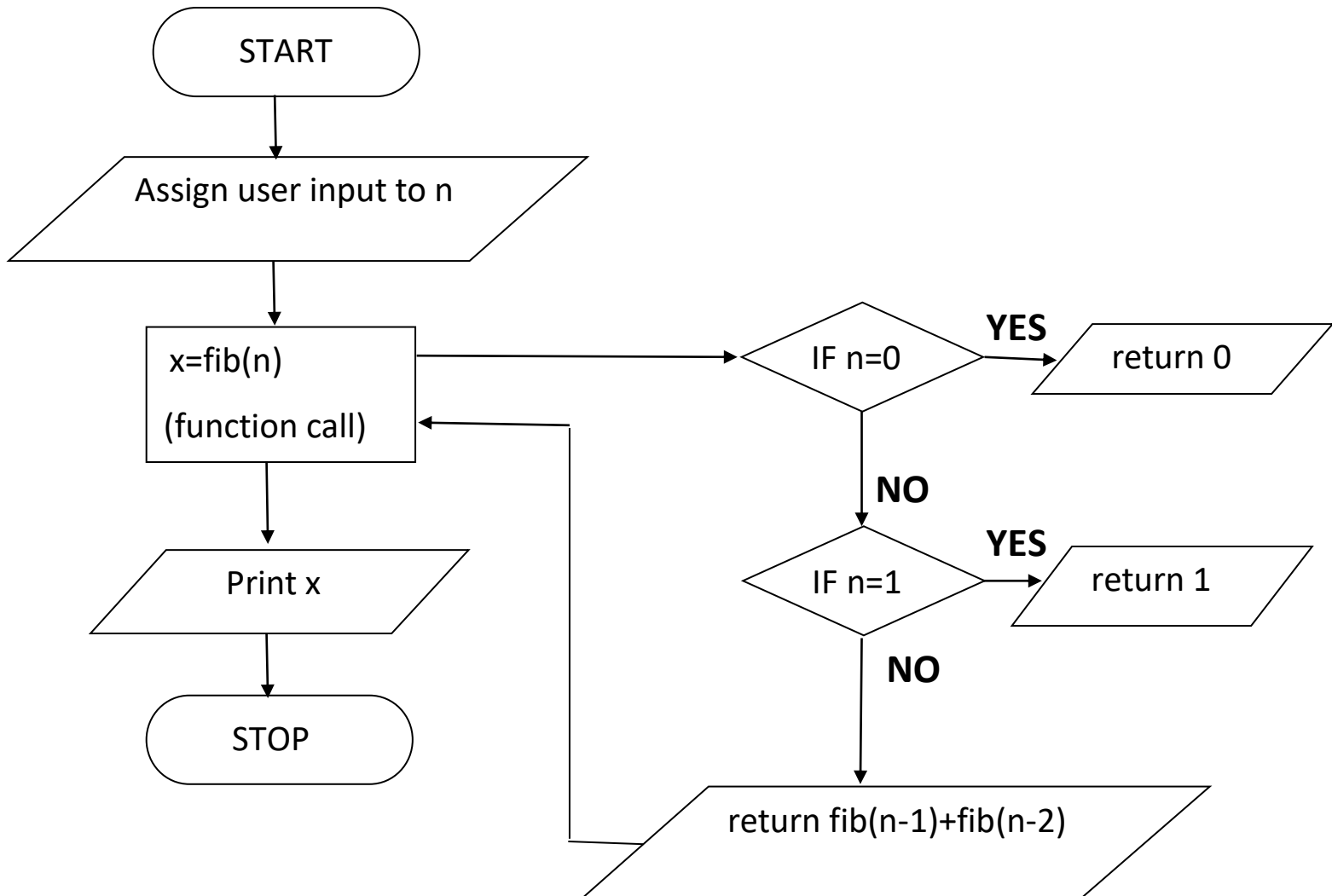
Step 11: STOP

Space Complexity:

As per this algorithm(non-recursive),there is a constant space occupied for the variables a , b , sum and i at the beginning. Even if there is iteration(loop) in the algorithm,the values stored in the variables are just overwriting for each iteration but there is no extra space occupied.Therefore,the space complexity is $O(1)$.

2. Recursive method:

Flow chart:



Algorithm:

Algorithm for recursive function fib(k):

Step 1: Take the input value k

Step 2: IF k is equal to 0

THEN

return 0

Step 3: IF k is equal to 1

THEN

return 1

Step 4: ELSE

return fib(k-1)+fib(k-2)

ENDIF

Main function:

Step 1: START

Step 2: Assign user input to n

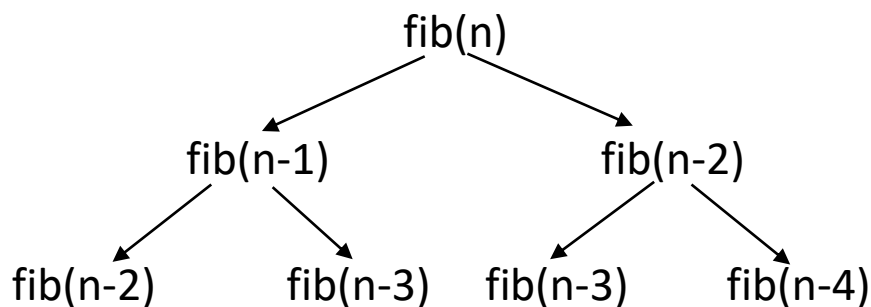
Step 3: Call the function fib(n) and assign the value to a variable x

Step 4: Print the value of x

Step 5: STOP

Space Complexity:

Here, in this algorithm we are calling the function $\text{fib}(k)$ multiple times to get the value of $\text{fib}(n)$. At each function call, extra memory will be allocated to store the return values in the stack. Hence space complexity is decided by that space. The function executes in the following way-



and so on till there is termination i.e., $\text{fib}(0)$ or $\text{fib}(1)$.

So while executing the function $\text{fib}(n)$ the space is occupied by the function in the stack till it find its value(i.e., it calls itself again and again till it terminates.).Therefore, the space complexity is $O(n) \times \text{space occupied by the each function } \text{fib}(k)$.

$$\text{Space complexity} = O(n) \times O(1) = O(n)$$

(Let's say the space occupied by the function is constant i.e., $O(1)$)

❖ Based on these two algorithms, we can say that the non-recursive algorithm is the best case scenario and the recursive algorithm is the worst case as its space complexity is more.