Assignment No.	**1**
Title	Recursive and Iterative algorithms
PROBLEM STATEMENT/ DEFINITION	Write a non-recursive and recursive program to calculate Fibonacci numbers and analyze their time and space complexity.
Objectives	• Learn to implement procedures and to pass parameters. • Understand how to use recursion to define concepts. • Learn to implement recursive functions and handle a stack using low level instructions. • Learn how to write iterative programs • Analyze algorithm in tern of time and space Complexity
Software packages and hardware apparatus used	PC with the configuration as Latest Version of 64 bit Operating Systems, Open Source Fedora-GHz. 8 G.B. RAM, 500 G.B. HDD, 15";Color Monitor, Keyboard, Mouse with eclipse installed
References	http://en.wikipedia.org/wiki/Fibonacci_number http://www.ics.uci.edu/~eppstein/161/960109.html
STEPS	Refer details
Instructions for writing journal	• Date • Title • Problem Definition • Learning Objective • Learning Outcome • Theory-Related concept,Architecture,Syntax etc • Class Diagram/ER diagram • Test cases • Program Listing • Output • Conclusion

AssignmentNo.1

Title : Recursive an Iterative algorithms

Objectives:

- Learn to implement procedures and to pass parameters.
- Understand how to use recursion to define concepts.
- Learn to implement recursive functions and handle a stack using low level instructions.
- Learn how to write iterative program
- Analyze algorithm in tern of time and space Complexity

Theory:

Aim: Write recursive & iterative programme which computes the nth Fibonacci number, for appropriate values of n. Analyze behavior of the programme in their time and space complexity.

Recursive function

Simply put, a recursive function is one which calls itself. The typical example presented when recursion is first encountered is the factorial function. The factorial of n is defined in mathematics as the product of the integers from 1 to n .

$$n! = 1 \times 2 \times 3 \times \ldots \times (n-2) \times (n-1) \times n$$

For example:

$$\begin{aligned} 3! &= 1 \times 2 \times 3 \\ 4! &= 1 \times 2 \times 3 \times 4 \\ 5! &= 1 \times 2 \times 3 \times 4 \times 5 \end{aligned}$$

Factorials are useful in counting ordered outcomes. For example, consider a race run by five people. Assuming no ties, there are five possibilities as to who will cross the finish line first; there are subsequently four remaining possibilities for second place, three for third, two for second and finally only one possibility for last place. The total number of possible outcomes for the race is $5 \times 4 \times 3 \times 2 \times 1$, or $5!$.

A Recursive Factorial Function

Any iterative function can be implemented recursively and vice versa. The recursive function will always be slower due to the added overhead needed by function calls, but it is often times easier to state the solution of a problem recursively. The factorial function can be defined recursively by the following.

$$\begin{aligned} 5! &= 5 \times 4 \times 3 \times 2 \times 1 \\ 4! &= 4 \times 3 \times 2 \times 1 \\ 5! &= 5 \times 4! \\ n! &= n \times (n-1)! \end{aligned}$$

As is the case for every recursive definition, there must be some stopping point at which the function will no longer call itself. For factorial, this point is when zero is reached; by definition, 0!=1. This definition can be interpreted using the earlier race analogy to mean that there is only one way for a race with zero participants to end. The full recursive definition for factorial follows.

$$n! = \begin{cases} 1 & \text{if} \quad n = 0 \\ n \times (n-1)! & \text{if} \quad n > 0 \end{cases}$$

Add the function recursive Factorial, which implements this definition, to the program

```
1. int recursiveFactorial (int n)
2. {
3. int result;
4. if (n == 0)
5. result = 1;
6. else // n != 0
7. result = n * recursiveFactorial (n-1);
8. return result;
9. }
```

An Iterative Function

It is easy enough to write a function which uses a counting loop to multiply successive numbers together in order to obtain a factorial, which accepts a value parameter n and returns an integer; note how the work of the function is done by its loop.

```
1. int iterativeFactorial (int n)
2. {
3. int product = 1;
4. for (int i = 2; i <= n; i++)
5. product = product * i;
6. return product;
7. }
```

Fibonacci numbers

The Fibonacci numbers or Fibonacci series are the numbers in the following integer sequence: 0,1,1,2,3,5,8,13,21,…. .
By definition, the first two numbers in the Fibonacci sequence are 0 and 1, and each subsequent number is the sum of the previous two.
In mathematical terms, the sequence Fn of Fibonacci numbers is defined by the recurrence relation Fn= Fn-1 + Fn-2

Algorithm

Input: Read n value
Output: Prints the nth Fibonacci term

Step1: Start
Step2: Read n value for computing nth term in Fibonacci series
Step3: call Fibonacci (n)
Step4: Print the nth term
Step5: End Fibonacci(n)

Algorithm: Recursive Fibonacci computes the nth Fibonacci number, for appropriate values of n.

The sequence is conventionally defined as follows:

$$Fib_n = \begin{cases} 1 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ Fib_{n-1} + Fib_{n-2} & \text{if } n > 1 \end{cases}$$

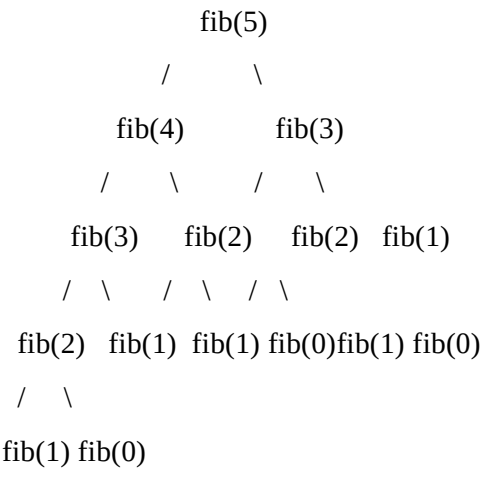
Step1: If n = 0 then go to step2 else go to step3
Step2: return 0
Step3: If n = 1 then go to step4
else go to step5
Step4: return 1
Step5: return(Fibonacci (n-1) + Fibonacci (n-2))

Analysis

Time Complexity: Exponential, as every function calls two other functions.

If the original recursion tree were to be implemented then this would have been the tree but now for n times the recursion function is called

Original tree for recursion



Space Complexity: O(n) if we consider the function call stack size, otherwise O(1).

An Iterative Fibonacci Function

In order to reduce the execution time for Fibonacci numbers, an iterative algorithm can be developed. The iterative factorial-for-n algorithm provides a good starting point. Time permitting, develop this algorithm and execute it with an extension to function main.

Step1: If n = 0 then go to step2 else go to step3
Step2: return 0
Step3: If n = 1 then go to step4
else go to step5
Step4: return 1
Step 5: for(i=3;i<=n;i++)

```
{ c=a+b
   a=b;
   b=c;
  return c;
 }
```

Time Complexity:O(n)
Extra Space: O(1)

Sample Input: n= 8 (Fibonacci Number Computation For which term you want to compute the Fibonacci number)

Observed Output:
Fibonacci Number of 9 is 21

Conclusion: We have successfully seen how recursive and iterative Fibonacci numbers are implemented and how analysis of same is done

FAQ

1. What is Fibonacci series?
2. What is the Logic of Fibonacci series?
3. Give example of Fibonacci series:
4. What is recursion?
5. What are the advantages of using functions?
6. What is data structure used in recusive fuction of Fibonacci series?
7. What is the complexity(space and time of recursive & non recursive Fibonacci numbers algorithm