GROUP A : ASSIGNMENT No. 1

Page No.
Date

Title: Fibonacci Numbers

Objective: To find time & space complexity of Fibonacci Series Algorithm.

Problem Statement:
Write a non recursive & recursive program to calculate Fibonacci Numbers & Analyze their time & space complexity.

Software & Hardware Requirement:
1> Desktop / Laptop
2> Any Operating System
3> IDE for writing & running code.

Theory:

Algorithm:
An algorithm can be defined as a finite set of steps, which has to be followed while carrying out a particular problem. It is nothing but a process of executing actions step by step.
An algorithm is a distinct computational procedure that takes input as a set of values & results in the output as a set of values by solving the problem. More precisely, an algorithm is correct, if for each input instance, it gets the correct output & gets terminated.

Asymptotic Notations:

Asymptotic notations are the mathematical notations used to describe the running time of an algorithm when the input tends towards a particular value or a limiting value.

e.g. In bubble sort, when the input array is sorted the time taken by the algorithm is linear i.e. the best case. When in reverse order the time taken is quadratic. When neither sorted, nor reverse it is average time.

There are mainly 3 Asymptotic Notations:

1) Big-O Notation: (O)

Big-O notation represents the upper bound of the running time of an algorithm. Thus, it gives the worst case complexity of an algorithm.

2) Omega Notation: (Ω)

Omega notation represents the lower bound of the running time of an algorithm. Thus, it provides the best case complexity of an algorithm.

3) Theta Notation: (Θ)

Theta notation encloses the function from above & below. Since it represents the upper & lower bound of the running time of an algorithm, it is used for analyzing the average-case complexity of an algorithm.

Page No.
Date

Algorithm:

1> Iterative:

```
Procedure Iterative_Fibonacci(n):
    int f0, f1, fib
    f0 = 0
    f1 = 1
    display f0,f1
    for int i := 1 to n:
        fib := f0 + f1
        f0 := f1
        f1 = fib
        display fib
    END for loop
END Iterative_fibonacci
```

2> Recursive:

```
Procedure Recursive_Fibonacci(n)
    int f0, f1
    f0 = 0
    f1 = 1
    if (n<=1):
        return 1
    return Recursive_Fibonacci(n-1) +
           Recursive_Fibonacci(n-2)
END Recursive_Fibonacci
```

Time Complexity: The time complexity of an algorithm quantifies the amount of time taken by an algorithm to run as a function of length of the input.

Page No.
Date

Space Complexity: The space complexity of an algorithm quantifies the amount of space taken by an algorithm to run as a function of the length of the input.

For Iterative Method:
- Time complexity is T(N) i.e. linear
- Space complexity is O(1).

For recursive Method:
- Time complexity is T(2^N) i.e. exponential.
- Space complexity is O(N).

Conclusion: Successfully implemented program for Fibonacci series using iterative & recursive approach & analyzed the space & time complexity for both approaches.

A+

30/8/22

Code:

Non Recursive Fibonacci

```
#include <iostream>
using namespace std;

int main(){

    int n;
    cout<<"Enter no of elements in series (>2): "<<endl;
    cin>>n;

    int a = 0;
    int b = 1;
    int c;
    cout<<a<<" "<<b<<" ";
    for (int i = 0; i < n-2; i++)
    {
        c = a+b;
        a = b;
        b = c;
    }

    cout<<c<<endl;
    cout<<"Count: "<<endl;

    return 0;
}
```

Output:

```
Enter no of elements in series (>2):
5
0 1 3
Count: 3
```

Recursive Fibonacci

```cpp
#include <iostream>
using namespace std;


int fib(int x,int &count){
    count++;
    if (x == 0 || x == 1)
    {
        return x;
    }
    else
    {
        return (fib(x-1,count) + fib(x-2,count));
    }
}

int main(){
    int n;
    int count = 0;
    cout<<"Enter no of elements in Fibonacci Series: "<<endl;
    cin>>n;
    cout << fib(n-1, count)<<endl;
    cout << "Count: "<<count<<endl;
    return 0;
}
```

Output:

```
Enter no of elements in Fibonacci Series:
5
3
Count: 9
```