**Practical – 01**

**Aim :**

Write a program non-recursive and recursive program to calculate Fibonacci numbers and analyze their time and space complexity.

**Program:**

import java.util.Scanner;

import java.util.ArrayList;

public class Fibonacci {

// Non-Recursive (Iterative) Fibonacci Calculation

public static int fibonacciNonRecursive(int n) {

if (n <= 0) {

return 0;

} else if (n == 1) {

return 1;

}

int[] fib = new int[n + 1];

fib[1] = 1;

for (int i = 2; i <= n; i++) {

fib[i] = fib[i - 1] + fib[i - 2];

}

return fib[n];

}

// Recursive Fibonacci Calculation

public static int fibonacciRecursive(int n) {

if (n <= 0) {

return 0;

} else if (n == 1) {

return 1;

} else {

return fibonacciRecursive(n - 1) + fibonacciRecursive(n - 2);

}

}

public static void main(String[] args) {

Scanner sb=new Scanner(System.in);

int n=sb.nextInt();

// int n = 10; // Change this to the desired value of n

// Calculate Fibonacci using non-recursive method

int nonRecursiveResult = fibonacciNonRecursive(n);

System.out.println("Fibonacci(" + n + ") (Non-Recursive) = " + nonRecursiveResult);

// Calculate Fibonacci using recursive method

int recursiveResult = fibonacciRecursive(n);

System.out.println("Fibonacci(" + n + ") (Recursive) = " + recursiveResult);

}

}

/\*

Recursive fibbonacci:

Time Complexity: O(n\*2n)

Auxiliary Space: O(n), For recursion call stack.

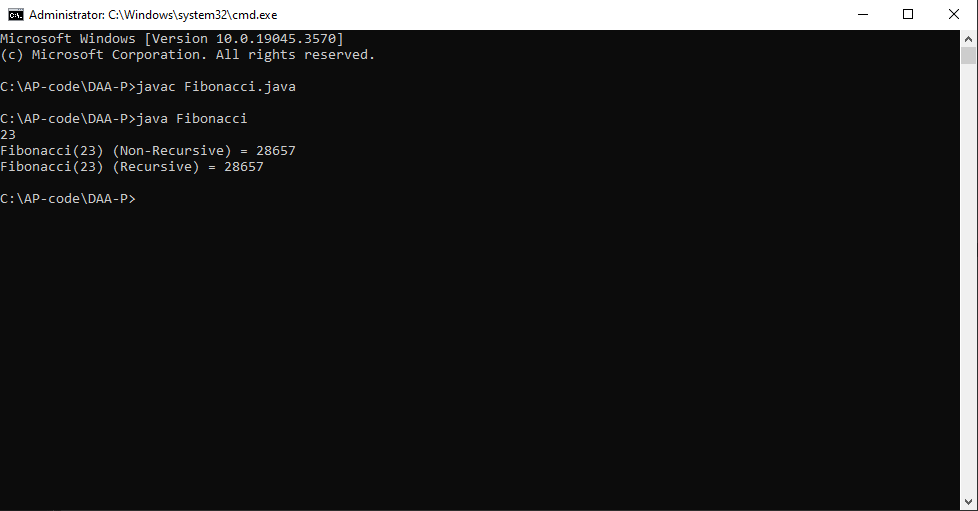
Iterative fibbonacci:

Time Complexity: O(n)

Auxiliary Space: O(1)

\*/

**Output:**

****