**DAY – 2**

**Assignment 2 :-**

**Recursive Function and Efficiency Analysis :-** Write a recursive function pseudocode and calculate the nth Fibonacci number and use Big O notation to analyze its efficiency. Compare this with an iterative approach and discuss the pros and cons in terms of space and time complexity.

**Solution :-**

**Recursive Fibonacci Function Pseudocode :-**

Function fibonacci(n)

if n is 0

return 0

else if n is 1

return 1

else

returnfibonacci(n - 1) + fibonacci(n - 2)

**Efficiency Analysis using Big O Notation :-**

The time complexity of the recursive Fibonacci function can be analyzed as follows:

- Each call to the fibonacci function results in two recursive calls (except for the base cases).

- Therefore, the number of recursive calls grows exponentially with 'n'.

- The time complexity of the recursive Fibonacci function is approximately O(2^n).

**Iterative Fibonacci Function Pseudocode :-**

Function fibonacci(n)

if n is 0

return 0

else if n is 1

return 1

else

Initialize variables a and b to store the Fibonacci numbers for n=0 and n=1 respectively

fori from 2 to n

c = a + b

a=b

b = next

return b

**Efficiency Analysis using Big O Notation :-**

The time complexity of the iterative Fibonacci function can be analyzed as follows:

- The iterative approach iterates 'n' times to calculate the nth Fibonacci number.

- Each iteration involves simple arithmetic operations (addition and assignment) which are constant time operations.

- Therefore, the time complexity of the iterative Fibonacci function is O(n).

**Comparison :-**

1. Time Complexity:

- Recursive Fibonacci: O(2^n)

- Iterative Fibonacci: O(n)

The iterative approach is much more efficient than the recursive approach in terms of time complexity. The recursive approach has exponential time complexity, making it highly inefficient for large values of 'n'.

2. Space Complexity:

- Recursive Fibonacci: O(n) due to the recursive call stack.

- Iterative Fibonacci: O(1) as it uses only a constant amount of extra space for variables.

The recursive approach consumes more space due to the call stack, potentially leading to stack overflow errors for large values of n. The iterative approach is more space-efficient as it does not use additional stack space.

**Pros and Cons :-**

- **Recursive Approach :-**

- Pros:

- Simple and easy to understand implementation.

- Directly follows the definition of the Fibonacci sequence.

- Cons:

- Exponential time complexity makes it highly inefficient for large 'n'.

- Consumes more memory due to the recursive call stack.

- **Iterative Approach :-**

- Pros:

- More efficient in terms of time complexity, especially for large 'n'.

- Requires less memory as it does not use additional stack space.

- Cons:

- Implementation may be slightly more complex compared to the recursive approach.

- Does not directly reflect the definition of the Fibonacci sequence, which may make it less intuitive for some developers.