Group A

Assignment No: 1

Title of the Assignment: Write a program non-recursive and recursive program to calculate Fibonacci numbers and analyze their time and space complexity.

Objective of the Assignment: Students should be able to perform non-recursive and recursive programs to calculate Fibonacci numbers and analyze their time and space complexity.

Prerequisite:

1. Basic of Python or Java Programming
2. Concept of Recursive and Nonrecursive functions
3. Execution flow of calculate Fibonacci numbers
4. Basic of Time and Space complexity

Contents for Theory:

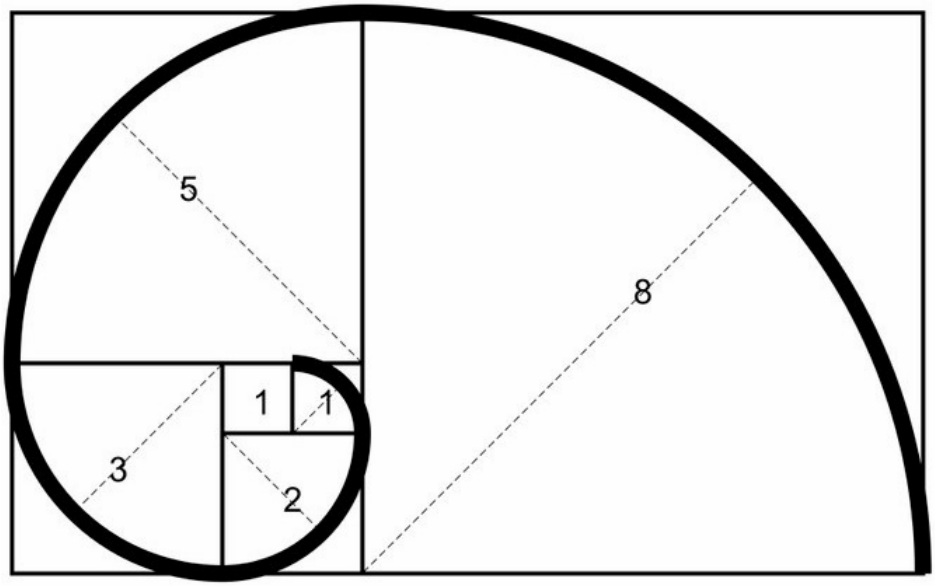
1. **Introduction to Fibonacci numbers**
2. **Time and Space complexity**

Introduction to Fibonacci numbers

- The Fibonacci series, named after Italian mathematician Leonardo Pisano Bogollo, later known as Fibonacci, is a series (sum) formed by Fibonacci numbers denoted as Fn. The numbers in Fibonacci sequence are given as: 0, 1, 1, 2, 3, 5, 8, 13, 21, 38, . . .
- In a Fibonacci series, every term is the sum of the preceding two terms, starting from 0 and 1 as first and second terms. In some old references, the term '0' might be omitted.

What is the Fibonacci Series?

- The Fibonacci series is the sequence of numbers (also called Fibonacci numbers), where every number is the sum of the preceding two numbers, such that the first two terms are '0' and '1'.
- In some older versions of the series, the term '0' might be omitted. A Fibonacci series can thus be given as, 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, . . . It can be thus be observed that every term can be calculated by adding the two terms before it.



- Given the first term, F0 and second term, F1 as '0' and '1', the third term here can be given as, F2 = 0 + 1 = 1

Similarly,

F3 = 1 + 1 = 2

F4 = 2 + 1 = 3

Given a number n, print n-th Fibonacci Number.

Fibonacci Sequence Formula

The Fibonacci sequence of numbers “Fn” is defined using the recursive relation with the seed values F0=0 and F1=1:

Fn = Fn-1+Fn-2

Here, the sequence is defined using two different parts, such as kick-off and recursive relation.

The kick-off part is F0=0 and F1=1.

The recursive relation part is Fn = Fn-1+Fn-2.

It is noted that the sequence starts with 0 rather than 1. So, F5 should be the 6th term of the sequence.

Examples:

Input : n = 2

Output : 1

Input : n = 9

Output : 34

The list of Fibonacci numbers are calculated as follows:

F_n	Fibonacci Number
0	0
1	1
2	1
3	2
4	3
5	5

6	8
7	13
8	21
9	34
… and so on.	… and so on.

Method 1 (Use Non-recursion)

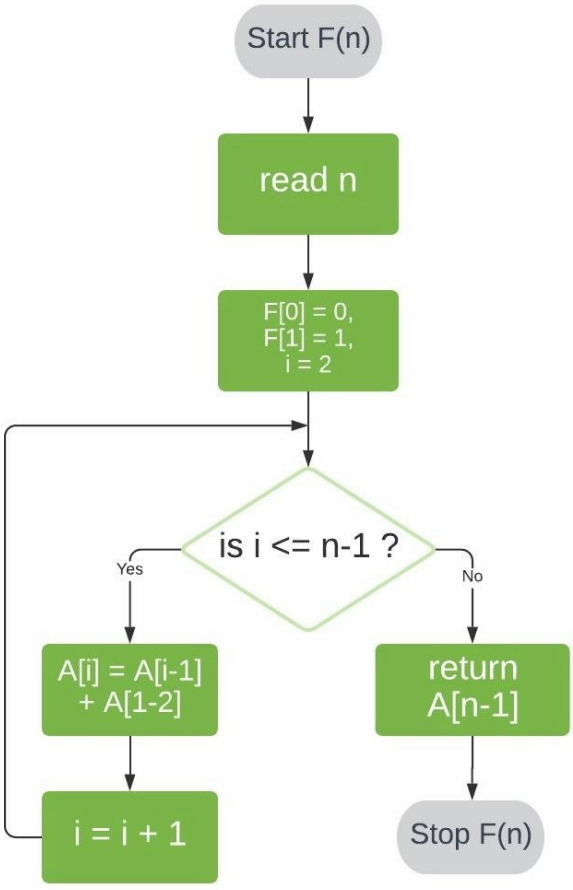
A simple method that is a direct recursive implementation of mathematical recurrence relation is given above.

First, we'll store 0 and 1 in $F[0]$ and $F[1]$, respectively.

Next, we'll iterate through array positions 2 to $n-1$. At each position i, we store the sum of the two preceding array values in $F[i]$.

Finally, we return the value of $F[n-1]$, giving us the number at position n in the sequence.

Here's a visual representation of this process:




```
# Program to display the Fibonacci sequence up to n-th term
nterms = int(input("How many terms? "))
# first two terms
n1, n2 = 0, 1
count = 0
# check if the number of terms is valid
if nterms <= 0:
   print("Please enter a positive integer")
# if there is only one term, return n1
elif nterms == 1:
   print("Fibonacci sequence upto",nterms,":")
   print(n1)
# generate fibonacci sequence
```

```
else:
  print("Fibonacci sequence:")
  while count < nterms:
     print(n1)
     nth = n1 + n2
     # update values
     n1 = n2
     n2 = nth
     count += 1
```

Output

How many terms? 7

Fibonacci sequence:
0

1

1

2

3

5

8

Time and Space Complexity of Space Optimized Method

- The time complexity of the Fibonacci series is **O(N) i.e, linear**. We have to find the sum of two terms and it is repeated n times depending on the value of n.
- The space complexity of the Fibonacci series using dynamic programming is **O(1)**.

Time Complexity and Space Complexity of Dynamic Programming

- The time complexity of the above code is **O(N) i.e, linear**. We have to find the sum of two terms and it is repeated n times depending on the value of n.

- The space complexity of the above code is **O(N)**.

Method 2 (Use Recursion)

Let's start by defining F(n) as the function that returns the value of Fn.

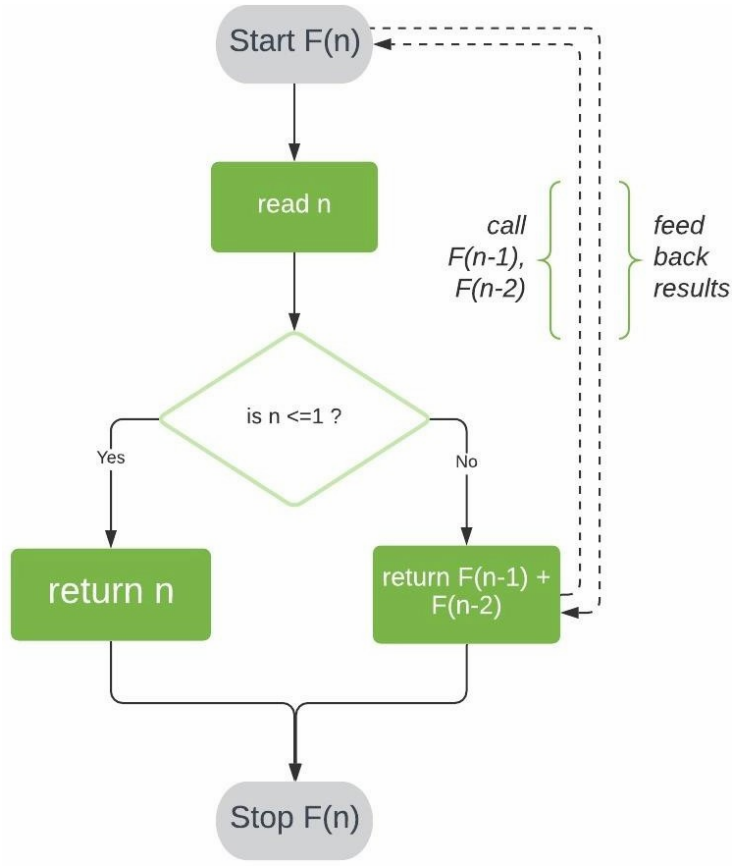
To evaluate *F*(*n*) for *n* > 1, we can reduce our problem into two smaller problems of the same kind: *F*(*n*-1) and *F*(*n*-2). We can further reduce *F*(*n*-1) and *F*(*n*-2) to *F*((*n*-1)-1) and *F*((*n*-1)-2); and *F*((*n*-2)-1) and *F*((*n*-2)-2), respectively.

If we repeat this reduction, we'll eventually reach our known base cases and, thereby, obtain a solution to *F*(*n*).

Employing this logic, our algorithm for *F*(*n*) will have two steps:

1. Check if $n \leq 1$. If so, return *n*.
2. Check if $n > 1$. If so, call our function *F* with inputs *n*-1 and *n*-2, and return the sum of the two results.

Here's a visual representation of this algorithm:



```
# Python program to display the Fibonacci sequence

def recur_fibo(n):
   if n <= 1:
      return n
   else:
      return(recur_fibo(n-1) + recur_fibo(n-2))

nterms = 7

# check if the number of terms is valid
if nterms <= 0:
   print("Plese enter a positive integer")
else:
   print("Fibonacci sequence:")
   for i in range(nterms):
      print(recur_fibo(i))
```

Output

```
Fibonacci sequence:
0
1
1
2
3
5
8
```

Time and Space Complexity

- The time complexity of the above code is **T(2^N) i.e, exponential**.

Date :

$\therefore T(n) = 2^n * T(n-n) + (2^n - 1) * C.$

$T(n) = 2^n T(0) + 2^n * C - C$

$T(n) = 2^n (1 + C) - C$

$T(n) = 2^n.$

Recursive algorithm:-

Algorithm r Fibonacci(n).

{ if (n <= 1)

→return n;

else.

return r Fibonacci (n-1) +

r Fibonacci (n-2);}

plus some constant Value.

Date : / /

Analysis.

$T(n) = T(n-1) + T(n-2) + C.$

$\boxed{T(n) = 2T(n-1) + C}$ ... // from approx. $T(n-1) \approx T(n-2)$.

$= 2 * (2T(n-2) + C) + C.$

$= 4T(n-2) + 2C + C.$

$= 4T(n-2) + 3C.$

$\Rightarrow 4(2T(n-2) + C) + 3C.$

$\Rightarrow 8T(n-3) + 4C + 3C.$

$\Rightarrow 8T(n-3) + 7C.$

Lets assume.

$T(n) \Rightarrow 2^K * T(n-K) + (2^K - 1) * C$

Lets find the value of K.

for which : $n - K = 0$

where $\boxed{n = K.}$

- The Space complexity of the above code is **O(N) for a recursive series.**

Method	Time complexity	Space complexity
Using recursion	**T(n) = T(n-1) + T(n-2)**	**O(n)**
Using DP	**O(n)**	**O(1)**
Space optimization of DP	**O(n)**	**O(1)**
Using the power of matrix method	**O(n)**	**O(1)**
Optimized matrix method	**O(log n)**	**O(log n)**
Recursive method in O(log n) time	**O(log n)**	**O(n)**
Using direct formula	**O(log n)**	**O(1)**
DP using memoization	**O(n)**	**O(1)**

Applications of Fibonacci Series

The Fibonacci series finds application in different fields in our day-to-day lives. The different patterns found in a varied number of fields from nature, to music, and to the human body follow the Fibonacci series. Some of the applications of the series are given as,

- It is used in the grouping of numbers and used to study different other special mathematical sequences.
- It finds application in Coding (computer algorithms, distributed systems, etc). For example, Fibonacci series are important in the computational run-time analysis of Euclid's algorithm, used for determining the GCF of two integers.
- It is applied in numerous fields of science like quantum mechanics, cryptography, etc.
- In finance market trading, Fibonacci retracement levels are widely used in technical analysis.

Conclusion- In this way we have explored Concept of Fibonacci series using recursive and non recursive method and also learn time and space complexity.