Name: Kakulavarapu Lakshmi Gayathri

Reg. No.: 18BEC019

Subject: Introduction To Algorithms (EC351)

Problem Statement:

Fib(n) = Fib(0), Fib(1),….. Fib(n), where Fib(n) = Fib(n-1) + Fib(n-2)

- Draw the Flowchart, Algorithms in pseudo code for solving.
- Write two types of algorithm (recursive and non-recursive) for fib(5) and fib(500) series.
- Find out the Total Memory or Space required to perform these Fibonacci series computational operations.
- Find out Worst Case and Best Case scenario from the above identified approaches.
- Write a program and compare the actual memory consumed by all the approaches.

Algorithm:

- Without Recursion:

Step 1: Start

Step 2: Declare variables number, n1, n2, n3.

Step 3: Initialize the variables: n1 = 0, n2 = 1 and n3 = 0.

Step 4: Enter the number of elements of Fibonacci Series to be printed.

Step 5: Print first two elements of Fibonacci Series.

Step 6: Use loop for the following steps:-

n3 = n1 + n2
n1 = n2
n2 = n3
Increase the value of number each time by 1.
Print value of n3.

Step 7: Stop

- With Recursion:

Step 1: Start

Step 2: Declare the value of N.

Step 3: Initialize the variables A = 0, B = 1, Count = 2.

Step 4: Print A, B.

Step 5: If (Count > N) then go to Step 10.

Step 6: Initialize the variable Next = A + B.

Step 7: Print Next.

Step 8: Use loop for following steps:-

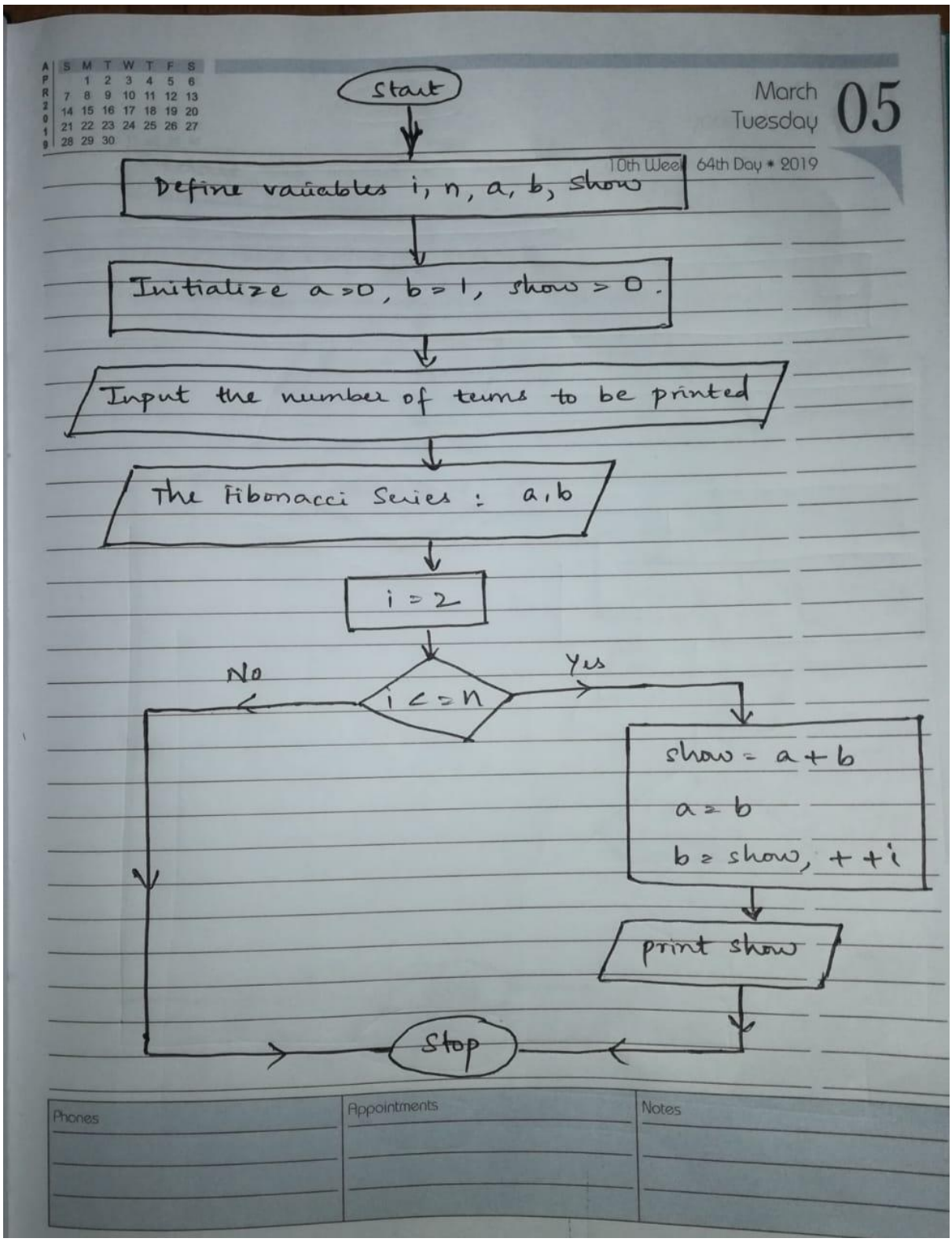
A = B.
B = Next.
Increase the value of Count each time by 1.

Step 9: Go to Step 4.

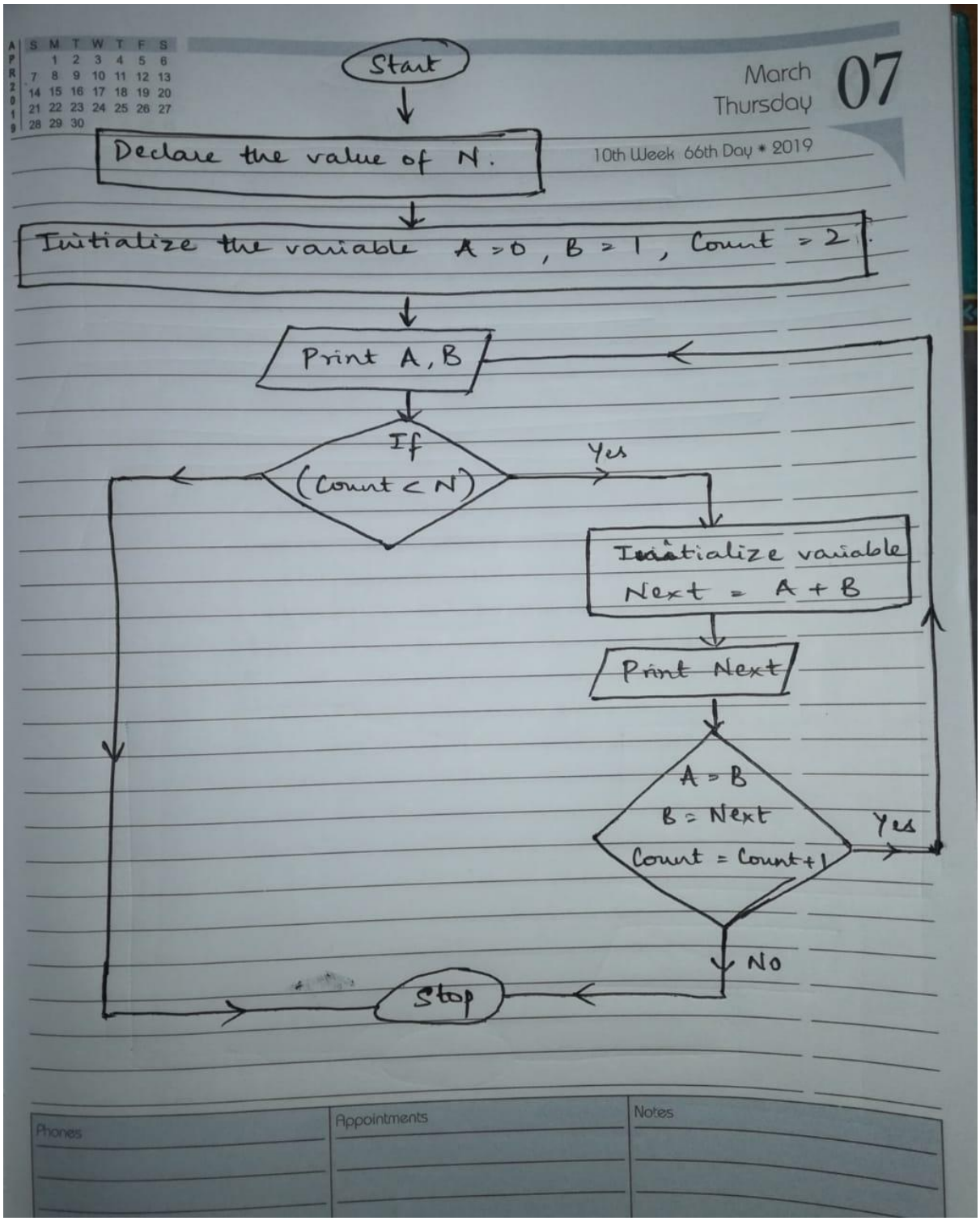
Step 10: Stop.

Flowchart:

- Without Recursion:



- With Recursion:



Two types of Algorithms for fib(5) and fib(500):

- Without Recursion:

For fib(5):

Step 1: Start

Step 2: Declare variables number, n1, n2, n3.

Step 3: Initialize the variables: n1 = 0, n2 = 1 and n3 = 0.

Step 4: Enter the number of elements of Fibonacci Series to be printed (number = 5).

Step 5: Print first two elements of Fibonacci Series (0, 1).

Step 6: Use loop for the following steps:-

n3 = n1 + n2
n1 = n2
n2 = n3
Increase the value of number each time by 1.
Print value of n3.

Step 7: Stop

Output: 0 1 1 2 3

For fib(500):

Step 1: Start

Step 2: Declare variables number, n1, n2, n3.

Step 3: Initialize the variables: n1 = 0, n2 = 1 and n3 = 0.

Step 4: Enter the number of elements of Fibonacci Series to be printed (number = 500).

Step 5: Print first two elements of Fibonacci Series (0, 1).

Step 6: Use loop for the following steps:-

n3 = n1 + n2
n1 = n2
n2 = n3
Increase the value of number each time by 1.
Print value of n3.

Step 7: Stop

Output: 0 1 1 ……

- With Recursion:

For fib(5):

Step 1: Start

Step 2: Declare the value of N. (N = 5)

Step 3: Initialize the variables A = 0, B = 1, Count = 2.

Step 4: Print A, B.

Step 5: If (Count > N) then go to Step 10.

Step 6: Initialize the variable Next = A + B.

Step 7: Print Next.

Step 8: Use loop for following steps:-

A = B.
B = Next.
Increase the value of Count each time by 1.

Step 9: Go to Step 4.

Step 10: Stop.

Output: 0 1 1 2 3

For fib(500):

Step 1: Start

Step 2: Declare the value of N. (N = 5)

Step 3: Initialize the variables A = 0, B = 1, Count = 2.

Step 4: Print A, B.

Step 5: If (Count > N) then go to Step 10.

Step 6: Initialize the variable Next = A + B.

Step 7: Print Next.

Step 8: Use loop for following steps:-

A = B.
B = Next.
Increase the value of Count each time by 1.

Step 9: Go to Step 4.

Step 10: Stop.

Output: 0 1 1….

Total Memory or Space Used:

- Without Recursion:

Total Memory = 4 bytes * 5 variables

= 20 bytes

Therefore, Space Complexity is O(1).

- With Recursion:

Total Memory = 4 bytes * 4 variables + O(n) (n recursive calls, n stacks used. So, O(n).)

= 20 bytes + O(n)

= O(n)

Therefore, Space complexity is O(n).

Worst Case and Best Case Scenario:

Iteration Fibonacci Algorithm (Non Recursive Fibonacci Algorithm) is best case since, its space complexity is O(1). For Fib(500), it iterates 500 times but does not call for a function or comes out of main function. In other words, total memory consumption does not depend on 'n'.

Recursive Fibonacci Algorithm is worst case since, its space complexity is O(n). As number of recursive calls increases, the number of stacks used increase which means the program occupies more space. In other words, total memory consumption depends on 'n'.

C Program:

- Without Recursion:

```
#include<studio.h>

int main()

{

        int n1 = 0, n2 = 1, n3, i, number;

        printf("Enter the number of Elements:  ");

        scanf("%d", &number);

        printf("\n%d %d", n1, n2);

        for (i = 2; i < number; ++i)
```

```
        {
                n3 = n1 + n2;
                printf("%d", n3);
                n1 = n2;
                n2 = n3;
        }
        return 0;
}
```

- With Recursion:

```
#include<stdio.h>
void printFibonacci(int n)
{
        static int n1 = 0, n2 = 1, n3;
        if (n>0)
        {
                n3 = n1 + n2;
                n1 = n2;
                n2 = n3;
                printf("%d", n3);
                printFibonacci(n-1);
        }
}
int main()
{
        int n;
        printf("Enter the number of Elements:  ");
        scanf("%d", &n);
        printf("Fibonacci Series:  ");
        printf("%d %d", 0, 1);
```

```
        printFibonacci(n-2);


        return 0;
}
```