Algorithm Task

Project Idea: 1 (Climbing Stair)

Team Number: 12

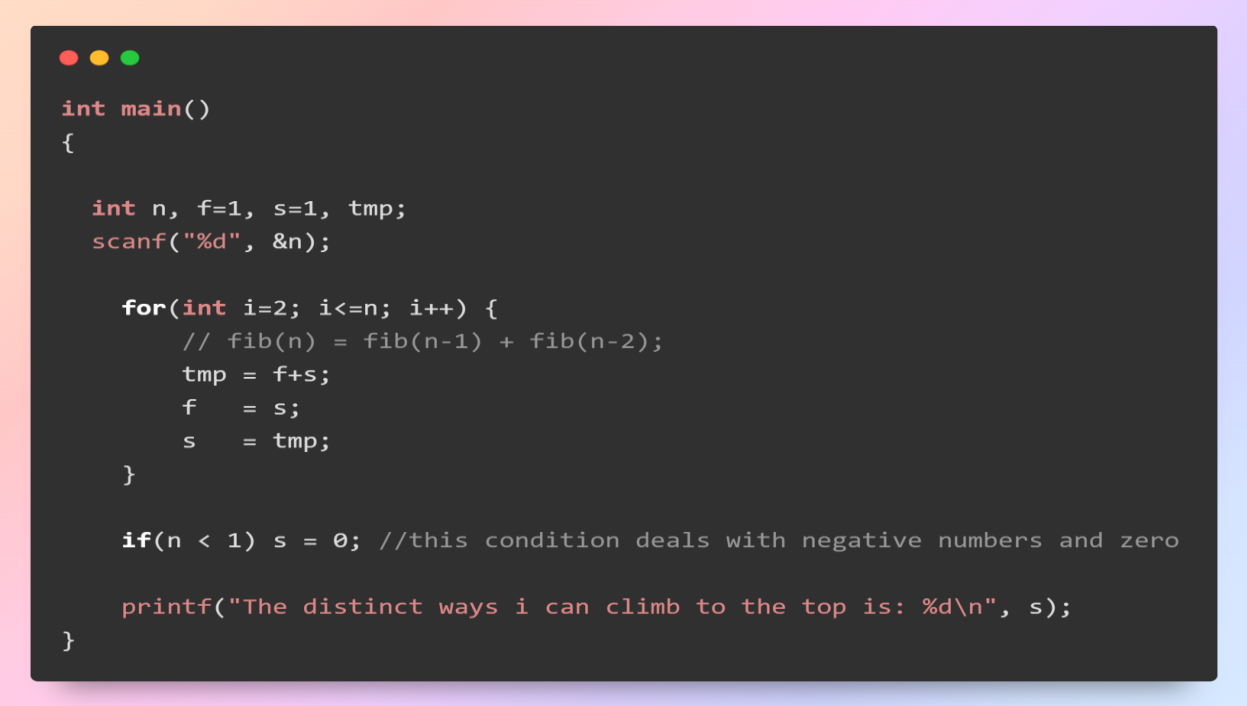
1- Climbing Stair

You are climbing a stair case. It takes n steps to reach to the top. Each time you can either climb 1 or 2 steps. Design an algorithm that calculate how many distinct ways can you climb to the top? Note: Given n will be a positive integer.

GitHub Link: AhmedMetwaly12/algorithmtask (github.com)

ID	Name
20210108	أحمد محمد متولي بيومي
20210102	أحمد محمد عبدالسلام محمد
20210115	أحمد محمود رمضان محمد
20210085	أحمد عمر حسين محمد
20210082	أحمد علي سالمان أحمد
20210026	أحمد السيد السيد محمود

Code Number 1:



```c
int main()
{

  int n, f=1, s=1, tmp;
  scanf("%d", &n);

    for(int i=2; i<=n; i++) {
        // fib(n) = fib(n-1) + fib(n-2);
        tmp = f+s;
        f   = s;
        s   = tmp;
    }

    if(n < 1) s = 0; //this condition deals with negative numbers and zero

    printf("The distinct ways i can climb to the top is: %d\n", s);
}
```

Pseudocode of Code Number 1

```
f := 1
s := 1
Read integer n from the user
for I := 2 to n do {
        tmp := f+s
        f := s
        s := tmp
}
If n < 1 then
        s := 0
print s
```

Analysis of Code Number 1

$$C(n) = \sum_{i=1}^{n} 1 = n\text{-}2 = O(n)$$

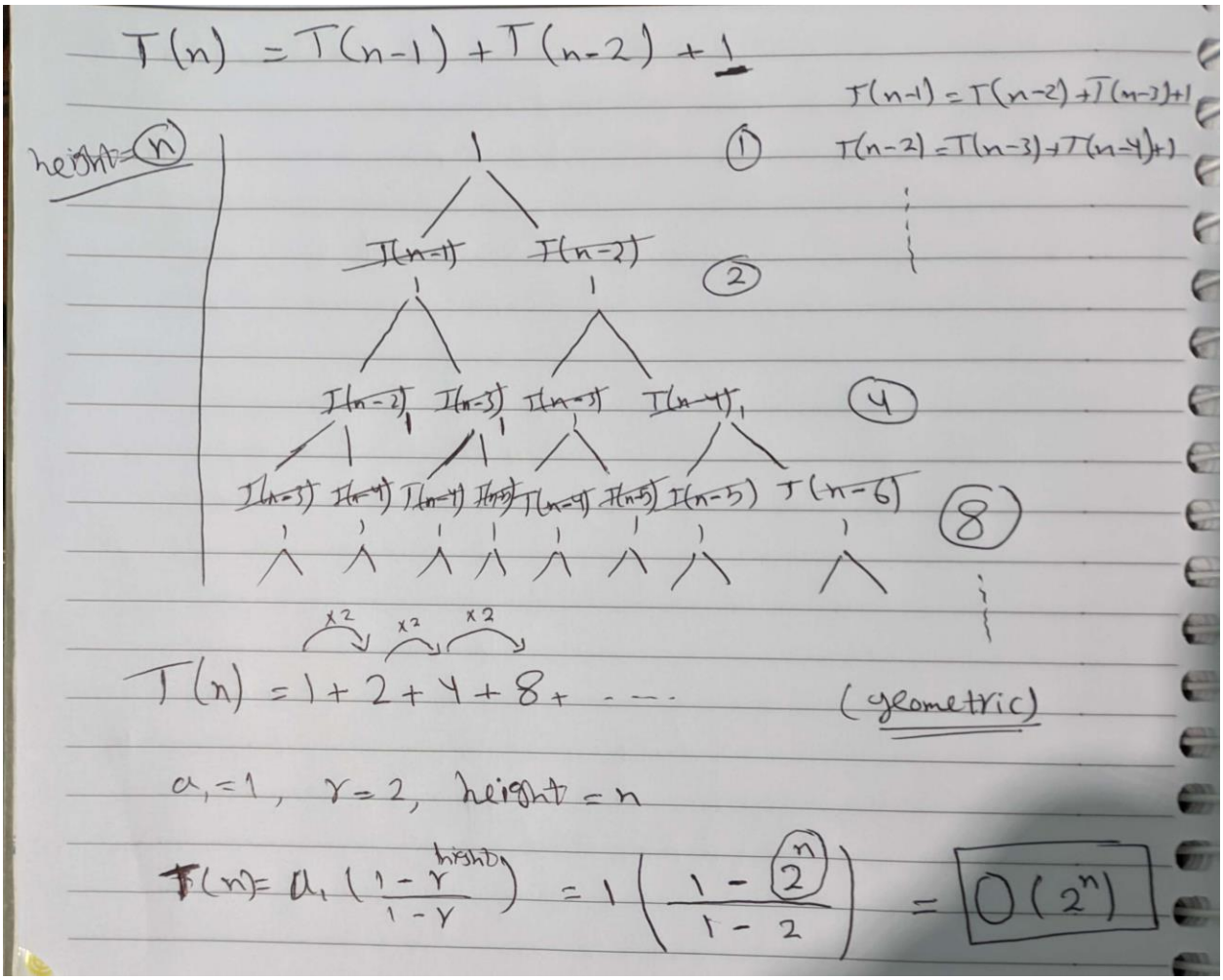
Code Number 2

```c
int fibonacci(int n) {
    if(n < 1) return 0; //this condition deals with negative numbers and zero

    if (n <= 2) return n; // 2=>2, 1=>1

    return fibonacci(n-1) + fibonacci(n-2);
}
```

Pseudocode of Code Number 2

```
function fibonacci(n)
   if n < 1 then
        return 0
   if n <= 2 then
      return n
   return fibonacci(n-1) + fibonacci(n-2)
```

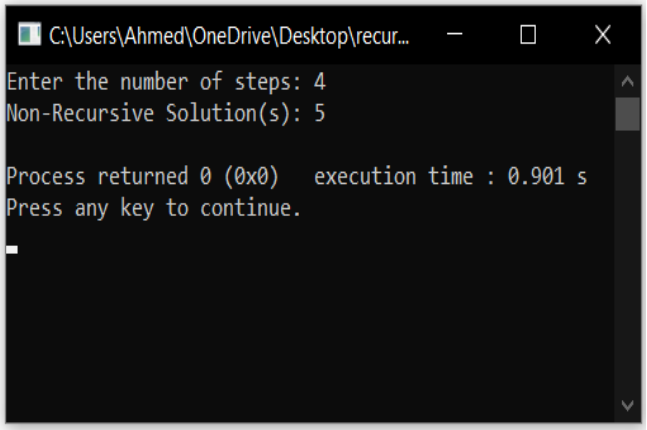
Analysis of Code Number 2



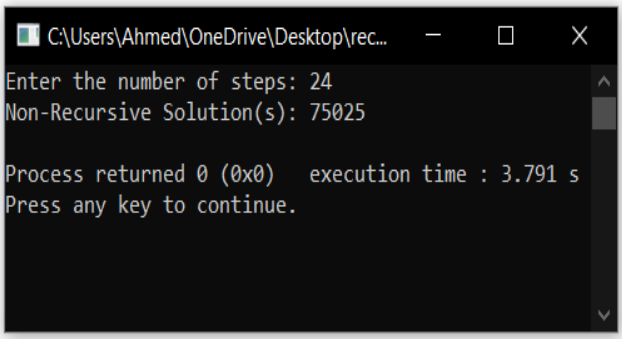
Test Run of Codes

Code 1

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{

    int n, f=1, s=1, tmp;
    printf("Enter the number of steps: ");
    scanf("%d", &n);

    for(int i=2; i<=n; i++) {
        // fib(n) = fib(n-1) + fib(n-2);
        tmp = f+s;
        f   = s;
        s   = tmp;
    }
    if(n < 1) s = 0; //this condition deals with negative numbers and zero
    printf("Non-Recursive Solution(s): %d\n", s);
}
```

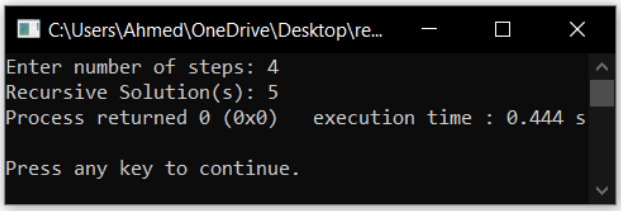



```c
#include <stdio.h>
#include <stdlib.h>

int main(){

    int n, f=1, s=1, tmp;
    printf("Enter the number of steps: ");
    scanf("%d", &n);

    for(int i=2; i<=n; i++) {
        // fib(n) = fib(n-1) + fib(n-2);
        tmp = f+s;
        f   = s;
        s   = tmp;
    }
    if(n < 1) s = 0; //this condition deals with negative numbers and zero
    printf("Non-Recursive Solution(s): %d\n", s);
}
```

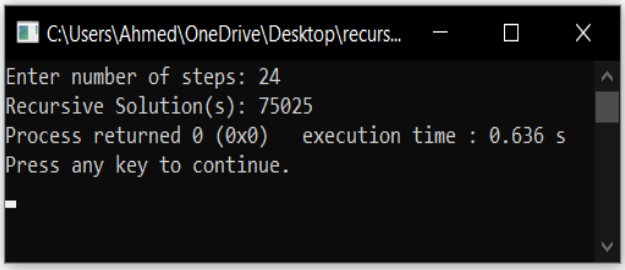


Code 2

```c
#include <stdio.h>
#include <stdlib.h>

int fibonacci(int n) {
     if(n < 1) return 0;
     if (n <= 2) return n; // 2=>2, 1=>1
     return fibonacci(n-1) + fibonacci(n-2);
}

int main()
{

     int n;
     printf("Enter number of steps: ");
     scanf("%d", &n);
     printf("Recursive Solution(s): %d",fibonacci(n));
}
```




```c
#include <stdio.h>
#include <stdlib.h>

int fibonacci(int n) {
     if(n < 1) return 0;
     if (n <= 2) return n; // 2=>2, 1=>1
     return fibonacci(n-1) + fibonacci(n-2);
}

int main()
{

     int n;
     printf("Enter number of steps: ");
     scanf("%d", &n);
     printf("Recursive Solution(s): %d",fibonacci(n));
}
```



Comparison

Code No	Best Case	Worst Case
1	$\Omega(1)$	$O(n)$
2	$\Omega(1)$	$O(2^n)$

Therefore: Code number 1 is better, more time efficient.