Recursive function of Fibonacci Function:

```
int fibRec (int n) {
        //Base Case
         if(n<=0) return 0;
         if(n==1) return 1;
         //Recursive Representation
        return fibRec(n-1) +fibRec(n-2);
}
```

The function calls itself 2 times for each non-base case. This creates a recursion tree where:

- Each node branches into 2 more recursive calls
- The depth of the tree is n
- The total number of function calls is approximately 2^n

So, the Big O notation for this is O(2^n). The time complexity is O(2^n) for recursive function

Non-Recursive Function for Fibonacci Function:

```
int fibLoop (int n){
   //Base Case
   if(n<=0) return 0;
   if(n==1) return 1;
   int fim1=1, fim2=0, fi=fim1+fim2;
   for (int i=2; i<n; i++) {
      fim2=fim1;
      fim1=fi;
      fi=fim1+fim2;
   }
   return fi;
}
```

This function uses one loop that runs n-2 times. Each step inside the loop takes O (1) time. This means it takes the same amount of time regardless of n. The other operations outside the loop also take O (1) time.
Time complexity = O (n) + O (1) = O (n)
We only keep O(n) in the final answer because when n gets large, the O (1) part becomes insignificant compared to O(n). Therefore, the non-recursive Fibonacci function has a time complexity of O(n).