Problem 6 | Derive O() for recursive fibonacci function

```
int fibRec(int n) {
    if (n<=0) return 0;
    if (n==1) return 1;
    return fibRec(n-1) + fibRec(n-2);
}
```

The above recursive function of the Fibonacci series is defined by

$$f(n) = \begin{cases} 0 & \text{if } n=0 \\ 1 & \text{if } n=1 \\ f(n-1)+f(n-2) & \text{if } n>1 \end{cases}$$

The recurrance relation for the fibonacci series

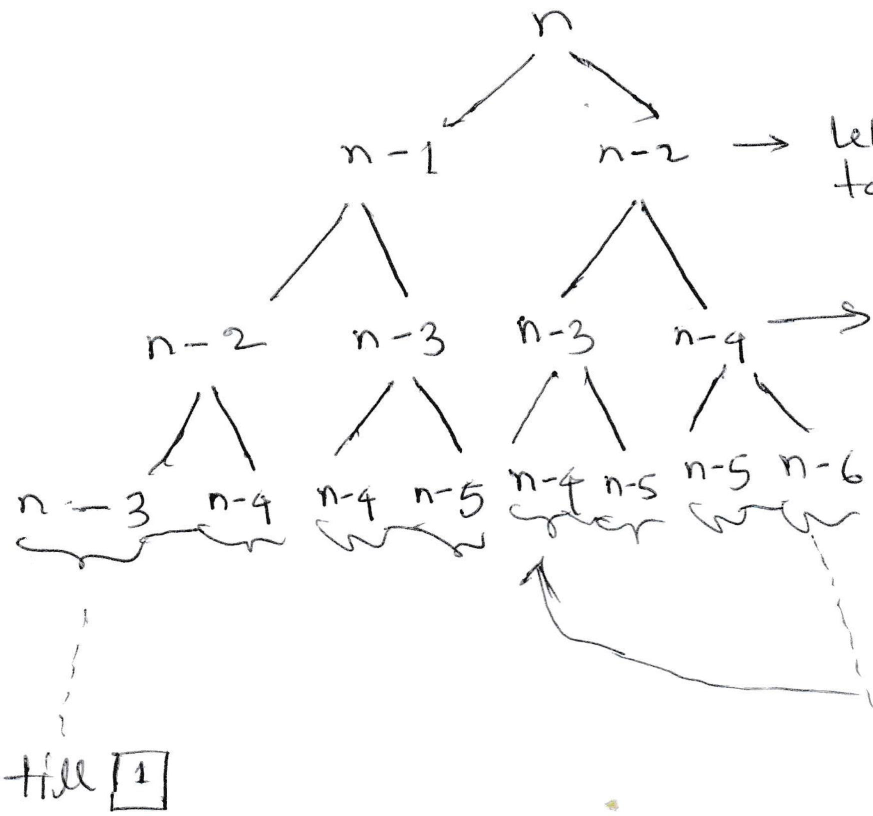
$$T(n) = T(n-1) + T(n-2) + c$$

Where $T(n)$ total number calls made to compute the $f(n)$. c is some constant operations before the recursive call of the function

Now, let's solve the recurrance relation for Fibonacci series,

$$T(n) = T(n-1) + T(n-2) + c$$

let's make the recursion tree; first the problem of size n is broken into $n-1$ and $n-2$



→ Let combine of (n-1) and (n-2); take k operations

→ In this level, (n-2) and (n-3); total k operations (n-3) and (n-4) another k operations. It needs 2*k operation

total 4*k operation
↳ or $2^2 * k$ operati

$2^{n-1} * k$

So, we can write the total operations for O( ),

$$T(n) = k + 2k + 2^2k + 2^3k \cdots + 2^{n-1}k$$

$$\Rightarrow T(n) = k\{1 + 2 + 2^2 + 2^3 + \cdots + 2^{n-1}\}$$

$$\Rightarrow T(n) = k\{2^n\}$$

$$\boxed{T(n) = c'2^n}$$ where $c' = k$

↳ So, the time complexity $O(2^n)$.

Derive O() for non-recursive fibonacci function:

```
int fibArray (int a) {
    int arr[n+1];
    arr[0] = 0;
    arr[1] = 1;
  for( int i=2; i<=n; i++) {
        arr[i] = arr[i-1] + arr[i-2];
  }
    return arr[n];
}
```

Let's consider, operations before the for loop $= O_b$

operations inside the for loop $= O_i$

operations after the for loop $= O_a$

We can write from the above function,

$$O_b + \sum_{i=2}^{n} (O_i) + O_a \quad \text{---(1)}$$

O, operation which equates T(0) → clock cycles

We know that, $\sum_{J=X}^{Y} 1 = (Y-X)+1$

So, from equation (1), we can write,

$$O_b + ((n-2)+1)O_i + O_a$$

$$= O_b + O_i n - O_i + O_a;$$

↳ f(n) is a first order polynomial

$= \boxed{c'n + c}$ Where $c' = O_i$; $c = O_b + O_a - O_i$

↖ it is O(n)