Time complexity analysis of recursive fibonacci function

Implementation 1

The equation for the function is $T(n) = T(n-1) + T(n-2) + c$

For the best case scenario,

we can consider $T(n-1) \approx T(n-2)$

So, $T(n) = 2T(n-2) + c$

$= 2^2 T(n-4) + 3c$

$= 2^3 T(n-6) + 7c$

$\vdots$

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

$\therefore n - 2k = 0$

$k = \frac{n}{2}$

Therefore,

$= 2^{n/2} T(0) + (2^{n/2} - 1)c$

Therefore the lowerbound is $\Omega(2^{n/2})$

For the worst case scenario,
we can consider $T(n-2) \approx T(n-1)$

$$\therefore T(n) = 2T(n-1) + c$$
$$= 2^2 T(n-2) + 3c$$
$$= 2^3 T(n-3) + 7c$$
$$\vdots$$
$$= 2^k T(n-k) + (2^k - 1)c$$

$\therefore n-k=0$

$n=k$

Therefore,
$$= 2^n T(0) + (2^n - 1)c$$

Therefore the upperbound is $O(2^n)$

As, we consider the upperbound as the time complexity.

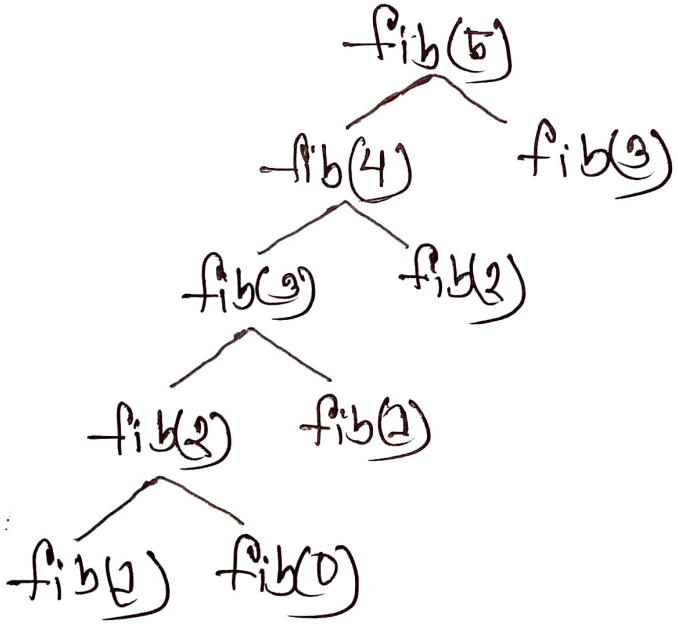
$\therefore$ The time complexity will be $O(2^n)$

Implementation 2

Here,

Every time we get a new value of fibonacci series. If we save it, then we do not have to find that number again.

Let see the fibonacci of the 5th term



Here it does not call the function for which we have fibonacci number for n times. It only takes fibonacci term only for 1 time for time save.

∴ Therefore, the time complexity will be O(n)