Name : Ashakuzzaman Odnee

ID : 20301268

Section : 11

Ans to the Q No-2

Implementation - 1:

```
def fibonacci_1(n):
    if n <= 0:
        print("Invalid input")
    elif n <= 2:
        return n-1
    else:
        retu fibonacci_1(n-1) + fibonacci_1(n-2)

n = int(input("Enter a number: "))
n_th fib = fibonacci_1(n)
print("The %d-th fibonacci number is %d" %(n, nth_fib))
```

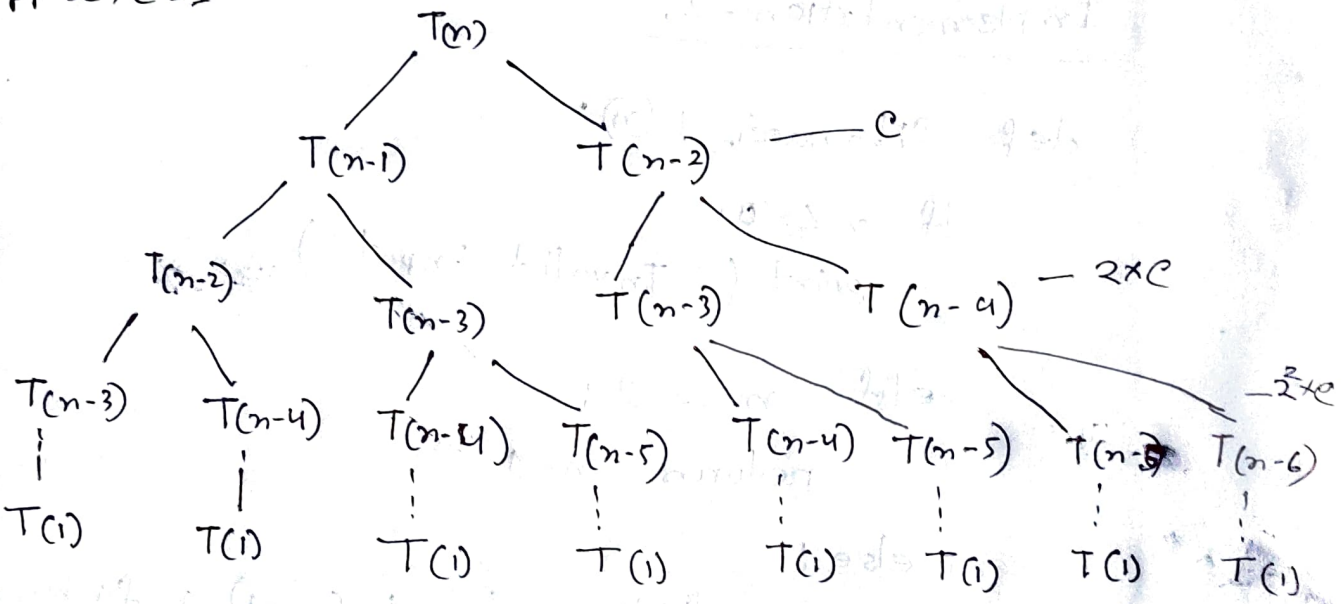
Now,

The running time equation we have,

$$T(n) = T(n-1) + T(n-2) + 1 \; ; \; T(0) = 1 \; , \; T(1) = 1$$

Recursion Tree:

Suppose, $c = 1$



The last term will be $\leq c \times 2^{n-1}$

$$T(n) \leq c + 2c + 2^2 c + 2^3 c + \cdots 2^{n-1} c$$

$$\Rightarrow T(n) \leq c \times \{ 1 + 2 + 2^2 + \cdots 2^{n-1} \}$$

$$\Rightarrow T(n) \leq c \times \{ 2^n - 1 \}$$

Therefore, $T(n) = O(2^n)$

Implementation - 2:

```
def fibonacci_2(n):
    fibonacci_array = [0, 1]
    if n<0:                                   <-- O(1)
        print("Invalid input!")
    elif n<=2:                                <-- O(1)
        return fibonacci_array[n-1]
    else:
        for i in range(2,n):                  <-- O(n)
            fibonacci_array.append(fibonacci_array[i-1]
                                   + fibonacci_array[i-2])
        return fibonacci_array[-1]
```

So, time complexity = $O(1) + O(1) + O(n)$

$= O(n)$

The above implementation is fasten than implementation - 1 ($O(n) < O(n2^n)$). Basically, in case of implementation - 2, we intenate

'n' number of times to find the n-th fibonacci number. That's way time complexity is O(n).

On the other hand, in the implementation I if $n > 2$ then $T(n) = T(n-1) + T(n-2) + 1$.

Because each recursion would call two ~~then~~ recursions. This aspect increases the time complexity exponentially. Hence, we got $O(2^n)$ as the time complexity of implementation-1.

A.

Ans to the Q No- 4

```
Procedure Multiply_matrix (A, B)
    Input  A, B   n x n matrix
    Output C      n x n matrix
begin,
    Initialize C as a n x n matrix
    for i=0 to n-1                          ---------- n
        for j=0 to n-1                      ---------- n x n
            for k=0 to n-1                  ---------- n x n x n
                C[i, j] += A[i, k] * B[k, j]
            end for
        end for
    end for
end Multiply_matrix
```

In the above algorithm, the inner, middle and outer loop execute n times, Basically, there are three nested loops.

∴ The time complexity is $O(n^3)$

Ans to the Q No-5

(1)

$T(n) = T(n/2) + n - 1 \quad \because T(1) = 0$

Using Master Teorem for $T(n/2) + n$ part,

$T(n)' = T(n/2) + n$

After comparing $T(n)'$ with $T(n) = a\,T(n/b) + f(n)$

$f(n) = \Theta(n^k \log^p n)$

We got, $p = 0, \quad a = 1, \quad b = 2 \quad , k = 1$

There fore, $T(n)' = n$ i [$\because \log_b^a < k$ and $p \geq 0$]

$\therefore T(n) = n - 1$

$\therefore$ Time complexity is $O(n)$.

A

$T(n) = T(n-1) + n - 1 \quad , T(1) = 0$

Here,
$T(n-1) = T(n-2) + n - 2$
$T(n-2) = T(n-3) + n - 3$

$\Rightarrow T(n) = T(n-2) + (n-2) + (n-1)$

$\Rightarrow T(n) = T(n-3) + (n-3) + (n-2) + (n-1)$

$\Rightarrow T(n) = T(n-k) + (n-k) + (n-(k-1)) + \cdots + (n-2) + (n-1)$

Assume,

$n - k = 1$

$\Rightarrow n = k + 1$

$\therefore T(n) = T(1) + 1 + 2 + \cdots + (n-2) + (n-1)$

$= 0 + \frac{n(n-1)}{2}$

$= \frac{n^2}{2} - \frac{n}{2}$

Thene fone, time complexity is $O(n^2)$.

Ans

(3)

$T(n) = T(n/3) + 2T(n/3) + n$

Using Masten Theorem in $2T(n/3) + n$ part,

$T(n)' = 3T(n/3) + n$

After comparing $T(n)'$ with $T(n) = aT(n/b) + f(n)$

$f(n) = \Theta(n^k \log^p n)$

we get,

$a=2, \quad b=3, \quad p=0, \quad k=1$

$\therefore T(n) = O(n)$: [Since $\log_b a < k$ and $p \geq 0$]

so, time complexity $= O(n)$

Again, using Masten Theorem,

$T(n) = T(n/3) + n$

we get, $a=1, \quad b=3, \quad p=0, \quad k=1$

$\therefore T(n) = O(n)$; [Since $\log_b a < k$ and $p \geq 0$]

$\therefore$ Time complexity is $O(n)$.

Given that,

$$T(n) = 2T(n/2) + n^2$$

Using Masten Theonem,

$a = 2,\ b = 2,\ k = 2,\ p = 0$ $\quad [\because f(n) = n^k \log^p n]$

$\therefore T(n) = O(n^2)$; $[\because \log_b^a < k$ and $p \geq 0]$

Thenefone, the wonst case complexity will be $O(n^2)$.

[proved]