Answer to the question No - 2

Implementation - 1:

```
def fibonacci_1 (n):                                   → T(n)
    if n <= 0 :
        print ("Invalid input!")                       → 1
    elif n <= 2:
        return n-1                                     → 1
    else :
        return fibonacc_1(n-1) + fibonacci_1(n-2)      → T(n-1) + T(n-2)

n = int (input ("Enter:"))
nth_fib = fibonacci_1 (n)
print (" ...... ")
```

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

Assume: $T(2) = T(1) = T(0) = 1$

Here, $T(n-1) \approx T(n-2)$

Scanned with CamScanner

$\therefore$ Rewriting the equation we get:

$$T(n) = T(n-1) + T(n-1) + c \quad [\text{taking constant, } c]$$

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

$$\Rightarrow T(n) = 2\{2T(n-2) + c\} + c$$

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

$$= 2^2\{2T(n-3) + c\} + 3c$$

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

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

$\vdots$ continue for k times

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

Assuming: $n - k = 0$

$$\Rightarrow n = k$$

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

$$= 2^k T(0) + (2^k - 1)c$$

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

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

$= 2^n + 2^n c - c$

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

$\therefore$ Time complexity $O(2^n)$

Implementation - 2:

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

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

Scanned with CamScanner

From implementation - 1 we got $O(2^n)$ time complexity and from implementation - 2 we got $O(n)$ time complexity. We know, $O(2^n) > O(n)$. Thus, Implementation - 2 is more efficient than implementation - 1.

Answer to the question No - 4

```
def multiply_matrix (A, B):
    n = len (a)
    c = [ ]

    for i in range (len(a)):
        m = [ ]
        for j in range(len(a)):
            m. append (0)
        c. append(m)
```

$n \times n = n^2$

$n \times n \times n = n^3$

```
for i in range (n):
    for j in range (n):
        for k in range (n):
            c[i] [j] += A[i] [k] * B[k] [j]

output = open ("output_problem4.txt",'w').
```

$n \times n = n^2$

```
for i in c:
    for j in i:
        output.write (str(j) + " ")

output.close( )
```

$\therefore f(n) = n^3 + n^2 + n^2$
$= n^3 + 2n^2$.

$\therefore$ Time complexity $= O(n^3)$