Ans to the ques no- 2

Implementation-1

```
def fibonacci_1(n):  ---> O(1)
    if n.....  ---> O(1)
        print.....  ---> O(1)
    elif n<=2:  ---> O(1)
        return....  ---> O(1)

    else:
        return fibonacci_1(n-1)+fibonacci_1(n-2)
                    ↓                   ↓
                  T(n-1)              T(n-2)
```

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

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

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

$$= 4T(n-2) + 2C + C$$

$$= 2^2 T(n-2) + 2^1 C + 2^0 C$$

For base condition,

$$T(2) = O(1)$$

So, $2^{n+2}\, T(n-(n+2)) + \ldots\ldots 2^1 C + 2^0 C$

$= 2^{n+2} \cdot 1 + \ldots\ldots 2^1 C + 2^0 C$

$= (2^{n+1} - 1)$

$= 2^n$

$\therefore$ Time complexity $= 2^n$

Implementation-2 :-

```
def fibonacci_2 (n):
    fibonacci_array . . . . . -> O[(1)
    if n<0:          ---> O(1)
        print . . . . . .
    elif . . . .     ---> O(1)
      return fibonacci_array [n-1]

    else:
        for i in . . . . . ---> (2,n) or O(n)
            fibonacci_array . . . . . .
        return fibonacci_array [-1]
```

So, $O(1) + O(1) + O(1) + O(n)$

$= O(n)$

∴ Time complexity $= O(n)$

Ans to the quer no-4

```
def matrixMultiplication(A,B):
    C = [[0] * (len(A)) for k in range (len(B[0]))]      → O(n)
    for i in range (len(A)):          → O(n)
        for j in range (len(B[0])):   → O(n)
            for k in range(len(B)):   → O(n)
                C[i][j] += A[i][k] * B[k][j]   → O(1)
    return C
```

Time complexity :-

$O(1) + O(n) + [O(n) \times O(n) \times O(n)]$

$= O(n^3)$

So, time complexity is $O(n^3)$