Task - 2

Implementation - 1

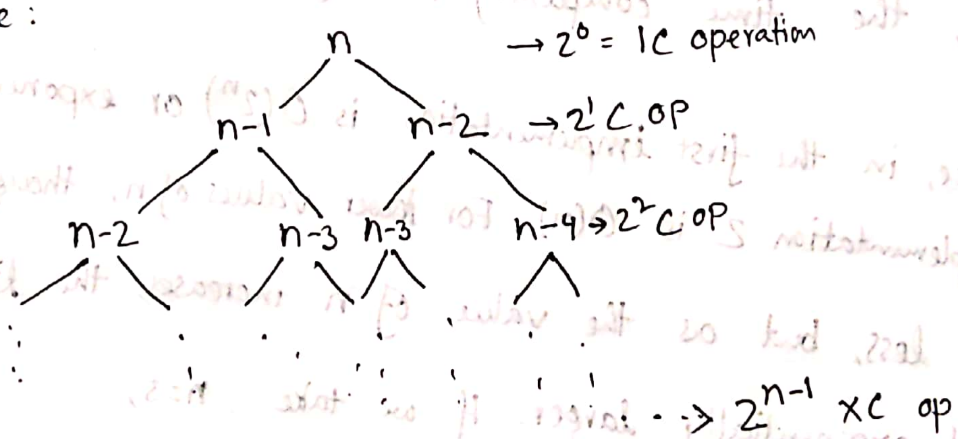
```
def fibonacci1(n):
    if n <= 0:
        print ("Invalid input")          } O(1)

    elif n <=2
        return n-1                       } O(1)

    else:
        return fibonacci1(n-1) + fibonacci1(n-2)
```

$T(n) = T(n-1) + T(n-2) + c$ → from constants

Recursion tree :



$T(n) = c + 2c + 2^2c + \ldots\ldots + 2^{n-1}c$

$\Rightarrow T(n) = c(1 + 2 + 4 + \ldots\ldots 2^{n-1})c$

$T(n) = 2^n$ (ignoring constants.

So, the time complexity is 2^n

Implementation 2 :

```
def fibonacci_2(n):
    fibonacci_array = [0,1]

    if n<0:
        print ('invalid input')      } O(1)
    elif n<=2:
        return fibonacciarray[n-1]   } O(1)

    else:
        for i in range (2,n):        } O(n)
            fibonacci_array.append(fibarr[i-1] + fibarr[i-2])

        return fibonacci_array [-1]
```

So, the time complexity is $O(n)$

Here, in the first implementation is $O(2^n)$ or exponential where the implementation 2 is $O(n)$. For lower values of n, though the difference will be less, but as the value of n increases, the time difference will get exponentially larger. If we take $n=5$,

implementation 1 : $O(2^5) = 32$

implementation 2 : $O(5) = 5$

So, implementation 2 is faster than implementation 1.

Task-4

Procedure Multiply-matrix (A, B)

Input A, B $n \times n$ matrix

Output C $n \times n$ matrix

```
begin
  Initialize C as a n×n zero matrix
  for i = 0 to n-1                          → O(n)
    for j = 0 to n-1                        → O(n)
      for k = 0 to n-1
        C[i, j] += A[i, k] * B[k, j]        } O(n)
      end for
    end for
  end for
```

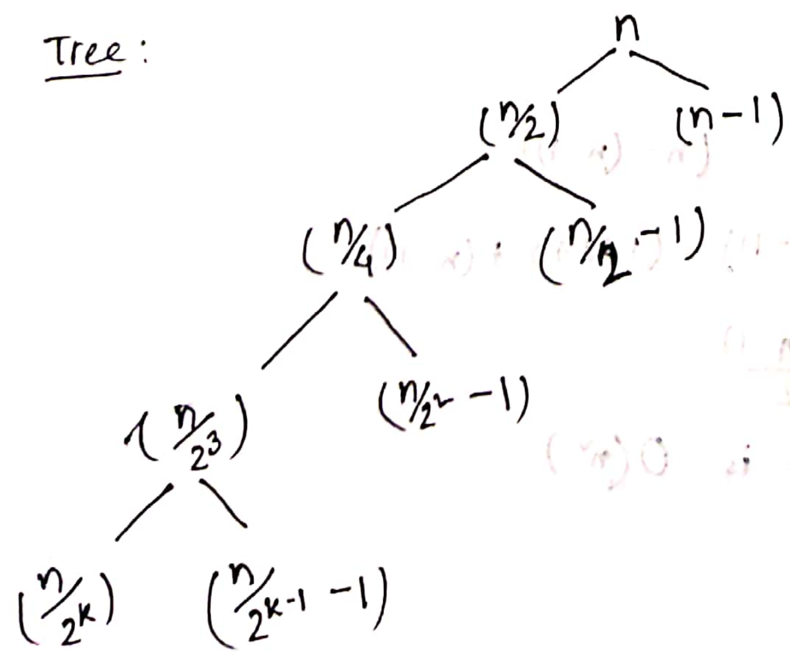
$O(n^2)$ (for j and k loops), $O(n^3)$ (all three loops)

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

Task-5

1. $T(n) = T(n/2) + (n-1)$, $T(1) = 0$

Tree :



So, $T(n) = (n-1) + (n/2 - 1) + (\frac{n}{2^2} - 1) + \cdots\cdots (\frac{n}{2^k} - 1)$

$= (\frac{n}{2^0} + \frac{n}{2^1} + \frac{n}{2^2} + \cdots\cdots \frac{n}{2^k})$

Or, $\frac{n}{2^k} = 1$

$\Rightarrow 2^k = n$

$\Rightarrow k = \log n$

So, $n(1 + \frac{1}{2} + \frac{1}{2^2} + + \cdots \frac{n}{2^{\log n}})$.

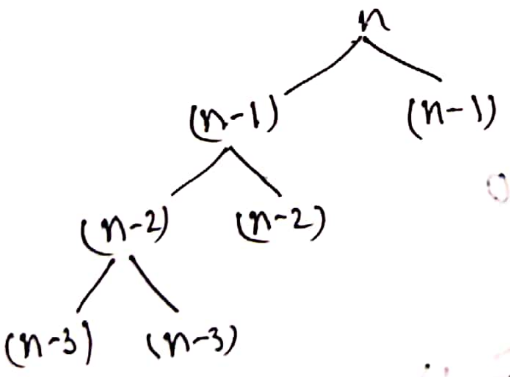
Also, $T(n) = n(\frac{1}{1 - 1/2}) - \log n$; $2n - \log n$

So, $T(n) = O(n)$

The time complexity is $O(n)$

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

Tree :



So, $T(n) = (n-1) + (n-2) + \cdots\cdots (n - (n-1))$.

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

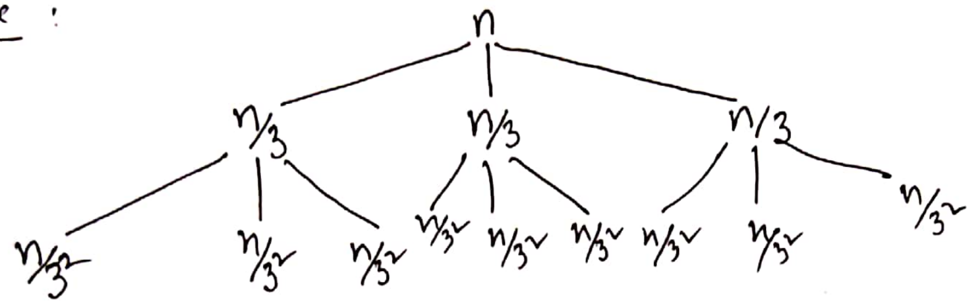
$= n(n-1) - \frac{n(n-1)}{2}$

So, time complexity is $O(n^2)$

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

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

Tree :



Let, $n/3^k = 1$

So, $k = \log_3 n$

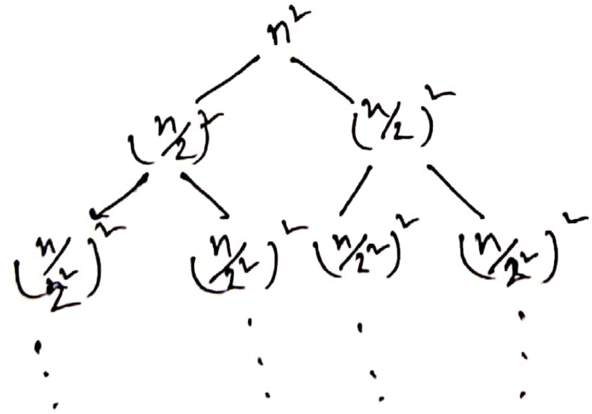
Now $T(n) = (n + n + \cdots \log n)$

$= n \log n$

So, the time complexity is $O(n \log n)$

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

Tree:



By Master Theorem, $a=2$, $b=2$, $k=2$, so, complexity is $O(n^2)$

So, the worst case complexity will be n^2.