CSE-221

Mohammad Rakibul Hasan Mahin

20201220

Section: 08

Problem - 2

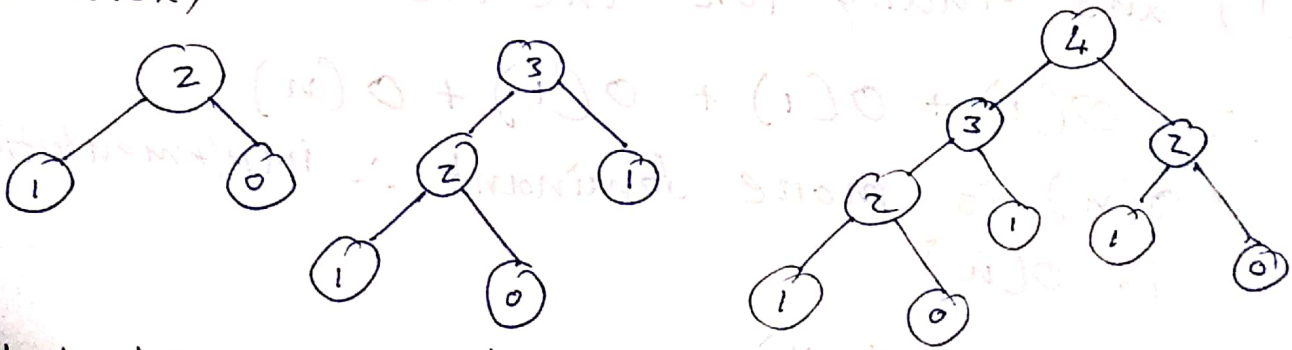
Implementation - 1

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

For first block of if condition $O(1)$, for elif $O(1)$
∴ for both time = $O(1)$

thus, $T(n) = O(1) + T(n-1) + T(n-2)$

For recursion,



If we look at the generated calls, leftmost nodes go down in descending order. Which means that height of the tree will be n.

Total number of call, in a complete binary tree $2^n - 1$

~~So the tree is not complete.~~

$O(2^n)$ is more dominant compared to $O(1)$

∴ Time complexety of implementation-1 is $O(2^n)$

Implementation-2

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

for if and elif, $O(1)$ for the return part $O(1)$ and finally for the for loop $O(n)$

∴ $O(1) + O(1) + O(1) + O(n)$

$O(n)$ is more dominant ∴ implementation-2 is $O(n)$

$O(n) < O(2^n)$

Thus, Implementation 2 is more efficient

Problem - 4

```
def Multiply-matrix(A, B):
O(1) ← n = Len(A)
for i in rang(0, ~~len(A)~~ n):
    for j in range(0, ~~len(A)~~ n):
        for k in range(0, ~~len(A)~~ n):
            k { C[i][j] += A[i][k] * B[k][j]
```

nxnxnxk
~~nxnxnxk~~
nxnxk
~~nxnxk~~
n

$\therefore \quad n \times n \times n \times k = n^3 k$

k is constant time so we can drop it

$\therefore$ we have on n^3

thus, $O(n^3)$