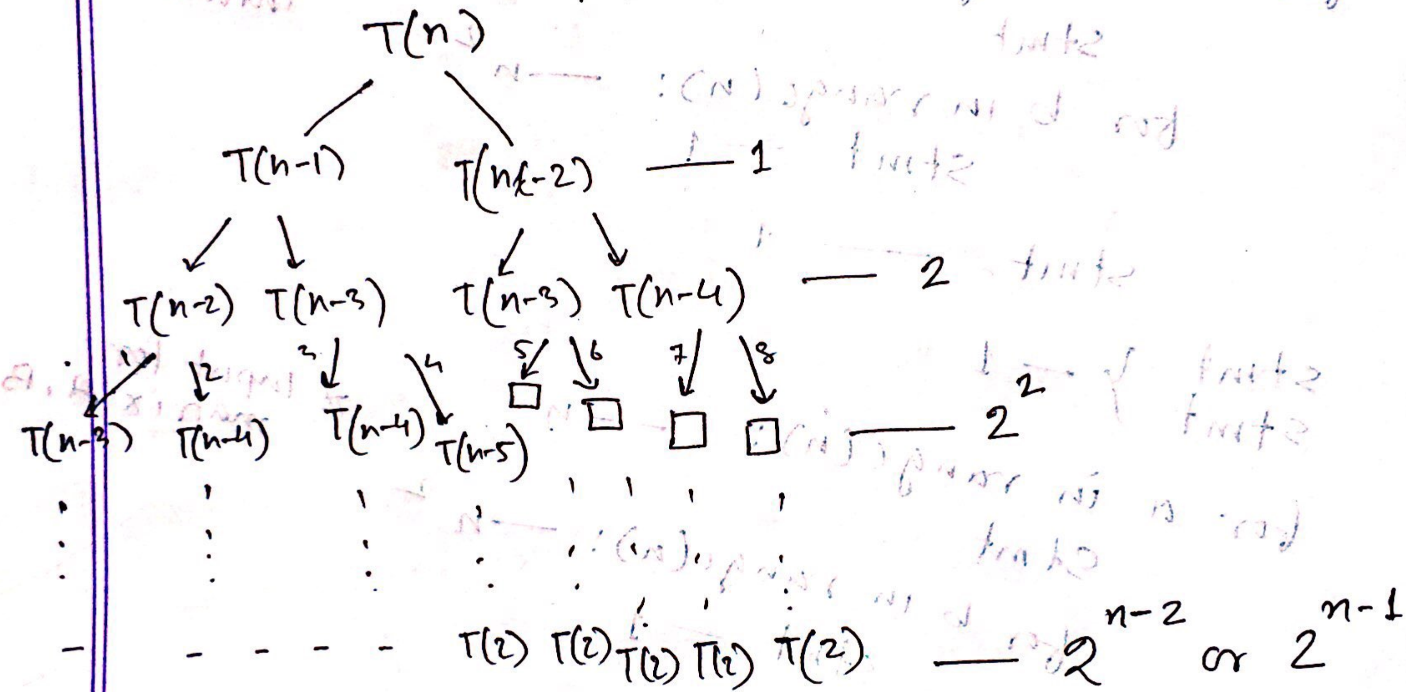
2(1).

~~2 i)~~ Recurrence function for this problem is,

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



~~So~~, Base cases are T(2) or T(1),

So, time taken is around,

$$1 + 2 + 2^2 + 2^3 \ldots\ldots 2^n$$

$$= 2^{n+1} - 1 = 2 \cdot 2^n - 1$$

$$\simeq 2^n$$

$$\simeq O(2^n).$$

Scanned with CamScanner

2(2)

```
def fibo_2(n):

    fibo_array = [0,1]
    if n<0:                      —— 1
        print("Invalid n")
    elif n<=2:                   —— 1
        return fin_arr[n-1]      ——
    else
        for i in range(2,n):     —— ≃ n
            fibo_arr.append(f_a[i-1] + f_a[i-2])
        return fibo_arr[-1].
```

$T(n) \approx n$.

So, $\underline{\theta(n)}$

i.e. for n there are almost n iterations.

Comparing implimentation 1 and 2, we see that 1st one has $\theta(2^n)$ complexity and 2nd one has $\theta(n)$ complexity, so, 2nd method is way faster than the 1st one,

$\theta(2^n) > \theta(n)$.

Scanned with CamScanner

stmt → statements.

(4) Considerings loops mainly.

```
stmt                      — 1
stmt                      — 1
for a in range (n):       — n        # matrix C
    stmt                               initialize.
    for b in range (n):   — n^2
        stmt              — 1

    stmt                  — 1

stmt } — 1
stmt }
for a in range(n):        — n        # Input for
    stmt                               matrix A, B.
    for b in range(n):    — n^2
        stmt              — 1

stmt                      — n        # matrix
for i in range(n):                     multiplication
    for j in range(n):    — n^2
        for k in range (n):  — n^3
            stmt          — 1

for i in range (n):       — n
    for j in range (n):   — n^2
        stmt
```

time T : $n^3 + 4n^2 + 4 + c$

$T(n) : O(n^3)$ Ans.