Ans: to the Que: No: 2

Implementation: 1

base case:

if $n <= 0$

and $n <= 2$

$T(1) = 1$

$T(2) = 2$

$T(0) =$ invalid input

$T(n) = T(n-1) + T(n-2) + O(1)$ [two base case $= O(2)$
one addition $= O(1)$
two substraction $= O(2)$
$O(5)$
$\Rightarrow O(1)$]

$= 2T(n-1) + 1$ [$T(n-1)$
= approximate $T(n-2$]

$T(n-1)$

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

$T(n-2) \Rightarrow 2(2(2T(n-3)+1)+1)+1$

$\Rightarrow 2^n + (n \times 1)$

$\Rightarrow 2^n$

above,
we are getting 1 whenever we call the function.
thus we will get n times of 1 [From (n-1) to (n-n)]
So ~~an~~ the time complexity will be $O(n)$.

again
the recursive function is giving us a 2 as its multiplication. ~~Thus~~ Thus, the time complexity will be,

$T(5) = 2.2.2.2.2 \cancel{T(n-5)} + 1$

$= 2^5 \cancel{T(n-5)} + 1$

$T(6) = 2^6 + 1$

$T(n) = 2^n + 1$

$\therefore T(n) = 2^n + n$

$= 2^n$

$= O(2^n)$.

implementation-2

big Omega
~~[illegible]~~:

$n < 0$: $\Rightarrow O(1)$

$n <= 2$ $\Rightarrow O(1)$

big O:

$O(n) + O(1) + O(1)$

$\Rightarrow O(n) + O(1)$

$\Rightarrow O(n)$

For loop is iterating n times. $\Rightarrow O(n)$

For example: $n=5$

thus iteration:

~~[illegible]~~

$(2-1)$ $(2-2)$ — 1

$(3-1)$ $(3-2)$ — 2

$(4-1)$ $(4-2)$ — 3

~~[illegible]~~ iteration: $n=3$ $T=(n-2)$
$=1$

$n=5$ $T=(n-2)=3$

Thus ~~we~~ I can say $n \approx (n-2)$

as a result ~~don~~ on n the time complexity will

be $O(n^2)$. / Ans.

Problem 41

initializing $= O(1)$ iteration $= O(1)$

First for loop $= O(n)$
Second " " $= O(n)$
third " " $= O(n)$

Thus, all are nested,

$$O(T) = O(n) \times O(n) \times O(n) \times 1$$
$$= O(n^3) \times 1$$
$$= O(n^3)$$

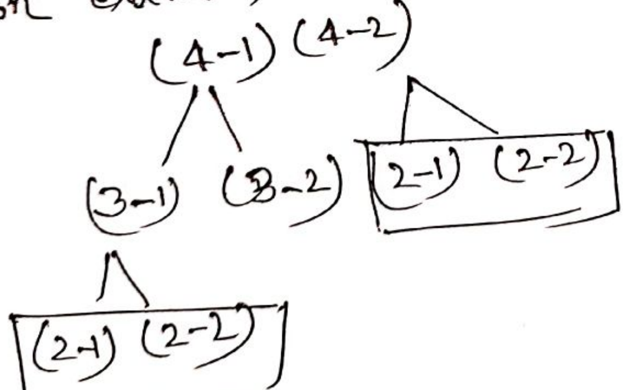
First For loop	2nd For loop	3rd For loop	total
		1	1
1	1	8	8
2	4	27	27
3	9	n^3	$\Rightarrow n^3$
n	n^2		

From above we can see every loop is in multiplication form thus final time complexity $n \times n \times n = O(n^3)$

Ans to the ques no :3

The difference between fibonacci 1 and 2 algorithm is clearly showed after generating the time complexity graph. We can see that till n=15 or 16 both algorithm worked in constant tim (O(1)). After n=16 there is a difference. The red line(fibo-1) has increasied suddenly. Thus we can say till (14-16) both algorithm work same time or gone through approx same amount of iteration. But after that the recursive algo showed some dissimilarity because recursive algo showed same number twice sometimes. recursive algo gone throw same

For example,



$\Rightarrow 2^n$

But in iterative algo the loop gone through n times only. Thus the ~~Ob~~ big O is not more than n. As input increases we can see red line is increasing where blue stay as ~~a~~ its previous position.