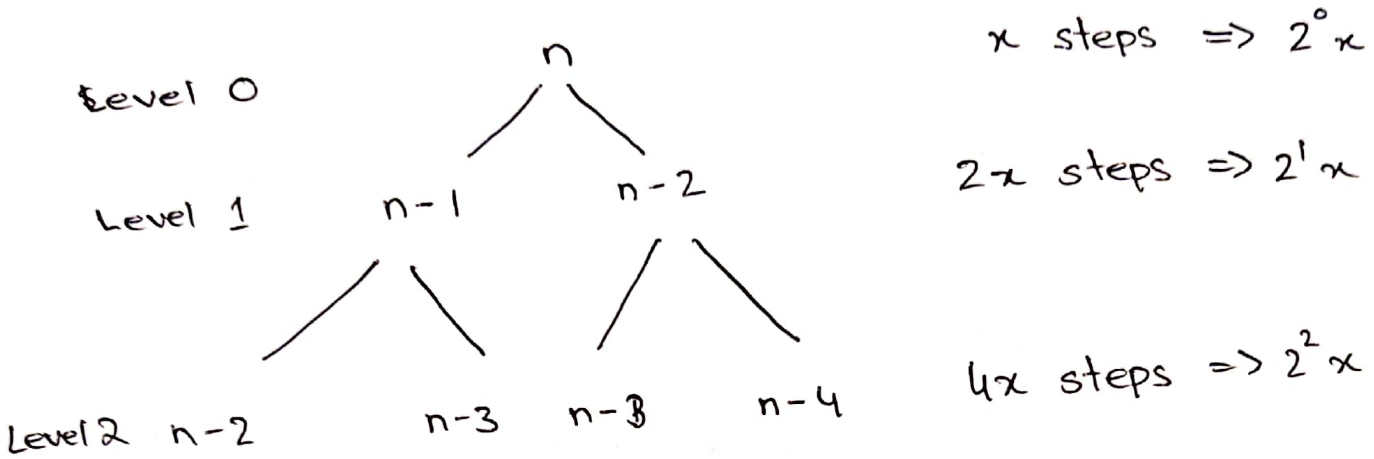
Problem 2

Implementation 1:



Considering the base condition that n-1 ~~should~~ will be returned when $n <= 2$, we can say that:

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

The for right decreases by 2 so they will reach f(2) or f(1) much more quickly than the left

time complexity is $2^{n-2} + 1$.

side. Hence the time complexity is

In terms of Big-OH, the time complexity is

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

$$\Rightarrow O(2^n . 2^{-2})$$

$$= O(2^n)$$

Implementation 2:

Time complexity for the loop in the else condition is $O(n-2)$ since the loop runs $n-2$ times. Other simple calculations such as appending the list or declaring and initializing the fibonacci list takes $O(1)$ each which can be ignored.
Thus our time complexity is:

$$O(n-2)$$
$$\Rightarrow O(n)$$

When we compare $O(n)$ and $O(2^n)$, it is immediately clear than $O(2^n) > O(n)$ and for very large values of n,

$$2^n \gg n.$$

$\therefore$ Implementation 2 is much better than implementation 1.

Scanned with CamScanner

Problem 4

Each pair of loops for matrix_A and matrix_B of my program would give a ~~ta~~ time complexity of

$$n^2 + C$$

n^2 for the two loops and C for all O(1) steps inside it.

$$\text{Total} = 2n^2 + D$$

For the triple loop to get matrix_C, we get a time complexity of $n^3 + C$

Now,

$$\text{Total} = 2n^2 + n^3 + E$$

To write the matrix_C in a text, I used double loops. Therefore time complexity for this part would be $n^2 + F$

$$\text{Total} = 3n^2 + n^3 + G$$

$\therefore$ $O(3n^2 + n^3 + G)$ would give us $O(n^3)$. ~~which is the time complexity~~