

Problem-2

Implementation-1 is a naive recursive ~~approach~~ approach. The recurrence relation for this

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

For simplicity, let $T(n-1) = T(n-2)$ that gives us, ~~$T(n)$~~

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

Now, using substitution,

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

$$= 2[2T(n-2)]$$

$$\begin{aligned} &= 2^2 T(n-2) \\ &= 2^2 [2T(n-3)] \\ &= 2^3 T(n-3) \end{aligned}$$

↳ This leads to a pattern, at i th step the recurrence relation is $T(n) = \cancel{2^{i-1} T(n-i)} = 2^i T(n-i)$

Now, when we arrive at the base case, the recursive call $T(n-i)$ gets executed in constant time.

therefore, $T(n-i) = 1$.

[E.P.T.O.]

Lets say, input = 0 is the base case.

[again we are being a little sloppy to simplify the calculation]

$$\text{then, } n-i=0 \\ n=i$$

Plugging the value in our recurrence relation.

$$T(n) = 2^n T(n-i)$$

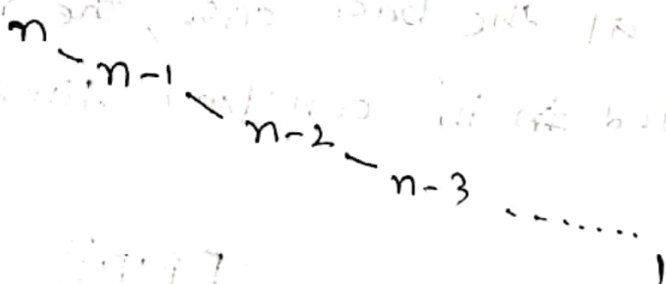
$$= 2^n \times 1$$

$$= 2^n$$

therefore, $T(n) \in O(2^n)$

Implementation-2 uses memoization to avoid repetitive calculation. Which means for an input "n", none of its subproblem will be calculated more than once.

The recurrence tree ~~looks like~~ looks like this



this gives us a linear runtime, asymptotically

the runtime is $O(n)$

if you look at the code for each loop is

$O(n)$

these three nested loops give a

runtime of $O(n^3)$

Problem-4

The code for matrix multiplication uses three nested loops. the range for each loop is $[0, n)$.

These ~~three~~ ~~the~~ three nested loops give a runtime of $O(n \times n \times n) = O(n^3)$