Problem-2

Implementation-1 is a naive recursive 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,

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

Now, using substitution,

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

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

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

$= 2^2 [2T(n-3)]$

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

→ this leads to a pattern, at ith step the recurrence relation is $T(n) = 2^i T(n-i)$

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

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

[P.T.O]

Lets say, input = 0 is the base case.

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

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

<u>Implementation-2</u> uses memoization to avoid repititive calculation. Which means for an input "n", none of it's subproblem will be calculated more than once.

The recurrence tree ~~kind of~~ looks like this

$n \to n-1 \to n-2 \to n-3 \ldots\ldots 1$

this gives us a linear runtime, asymptotically,

the runtime is $O(n)$

Problem-4

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

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