Nth element in 3 Fibonacci Series - QOTD 30 Jan

Leetcode Link : Click

$[TLE] Approach - 1$ (simple recursion)

Time : $O(3^n)$

> because we have to calculate 3 things with 3 different calls for each of the n

Space : O(n)

> recursive call stack

```
/* ✔★ Approach - 2 (simple recursive approach)
        explanation :-

        -> // Fun.2 : solve()

                step 1 : base case
                step 2 : we will check if the Tn we want already exist in map, if yes
then return that value

                step 3 : if the desired value of Tn is not in the map then we need to
calculate it every time we calculate a value of Tn we will store it in the map
                step 4 : we want recursion to give us values of Tn-1 Tn-2 Tn-3, and
then we add up and return them as Tn

   ->// main fucntion

         step 1 : we want to map every Tn with its resulting value,

    ☑Time : O(3^n) bit it is not 3^n is bigger test cases as we have used map to
memorize the valuess
    ☑Space : o(n) - for map and recursive stack

*/
```

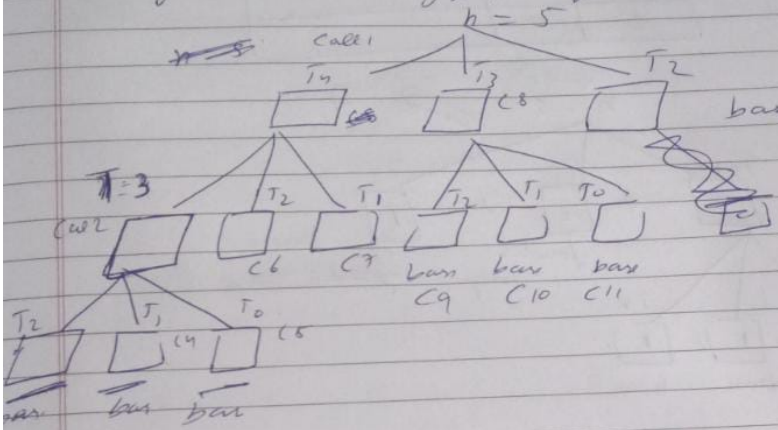
code :-

```
public:
    int tribonacci(int n) {
        // base case
        if(n == 0) return 0;
        if(n == 1 || n == 2) return 1;

        // we want recursion to give us values of Tn-1 Tn-2 Tn-3, and then we add up
and return them as Tn
        return tribonacci(n-1) + tribonacci(n-2) + tribonacci(n-3);

    }
};
```

Whats the problem with this approach and how it can be stored ?

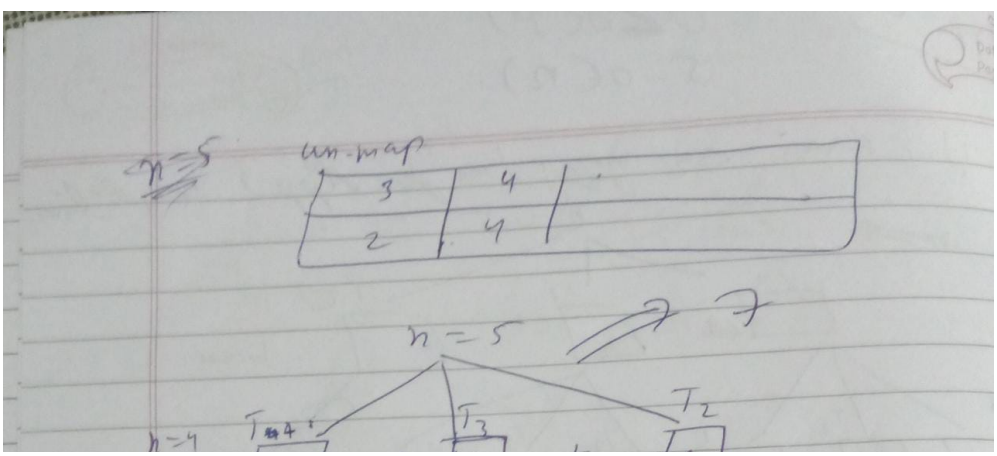
App-2 ~~O(3^n)~~ O(n).

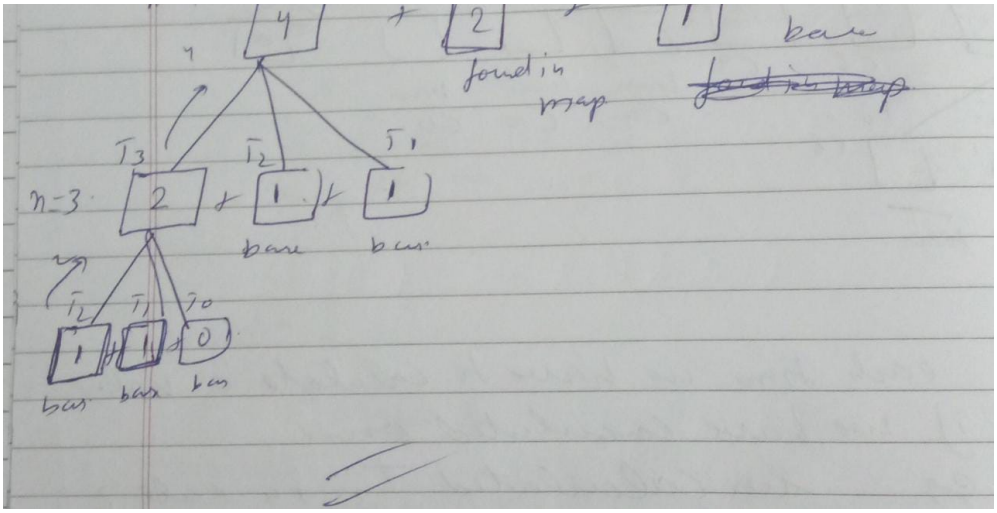
if i draw rec tree for App 1 it goes like



here each time we have to calculate a value even if we have calculated once for eg we calculated T_3 in call 2 once but then in call 8 we again calculated T_3, so we calculate T_3 2 time when n = 5 ~~t/y~~ if n is more larger no. then there will be more recursive calls.

So what if we store anything we calculated once in a map like if we calculated a value we store it in map & use it.





🌟 $Approach - 2$ (using recursion + map to store the prv outputs) - DP

Time : O(3^n)

> because we have to calculate 3 things with 3 different calls for each of the n

Space : O(n)

> recursive call stack

Approach/steps

```
/* ✔️🌟 Approach - 2 (simple recursive approach)
        explanation :-

        -> // Fun.2 : solve()

                step 1 : base case
                step 2 : we will check if the Tn we want already exist in map, if yes
then return that value

                step 3 : if the desired value of Tn is not in the map then we need to
calculate it every time we calculate a value of Tn we will store it in the map
                step 4 : we want recursion to give us values of Tn-1 Tn-2 Tn-3, and
then we add up and return them as Tn

   ->// main fucntion

         step 1 : we want to map every Tn with its resulting value,

    ☑Time : O(3^n) bit it is not 3^n is bigger test cases as we have used map to
memorize the valuess
    ☑Space : o(n) - for map and recursive stack

*/
```

code :-

```
private:
    int solve(int n, unordered_map<int,int> &map){
        // base case
        if(n == 0) return 0;
        if(n == 1 || n == 2) return 1;

        // we will check if the Tn we want already exist in map, if yes then return
that value
        if(map.count(n) != 0){
            return map[n];
        }

        // if the desired value of Tn is not in the map then we need to calculate it
every time we calculate a value of Tn we will store it in the map
        // we want recursion to give us values of Tn-1 Tn-2 Tn-3, and then we add up
and return them as Tn
        int Tn = solve(n-1, map) + solve(n-2, map) + solve(n-3, map);
        map[n] = Tn;

        return Tn;

    }
public:
    int tribonacci(int n) {

        // we want to map every Tn with its resulting value,
        unordered_map<int,int> map;

        return solve(n, map);

    }
};
```