Recursion : Solving a problem from backwards

{IDEA}

Find Max ?

10	20	35	15	64	89	72
0	1	2	3	4	5	6

Iterative solution :-

loop 0 to 6 :

store next element, if greater than current element

Here we solved it from forward.

Recursive way to ~~this~~ Solve same,

→ we start from end, and then reach at beginning.

1st: we need some marker, to know when we are at beginning (or end case / base case).

so, if we start from index 6, we know that arrays beginning is from 0, so anything less than that, is not applicable.

2nd: the logic to find maximum, the calling the function, everytime for each index ...

So,

```
recurseArray(index, array)

    if (index < 0)
        return    // we stop

    else

        ~~Check~~
        Compare current
        with max, if
        greater set the max to it.

        recurseArray(index-1, array)
            -> looping backwards
```

72	89	64	15	35	20	10
6	5	4	3	2	1	0

Max

recurseArray(5)

" " (4)

" " (3)

⋮

⋮

recurseArray(0)

recurseArray(-1)

returns 0

Now in Previous Program,
we solve to find maximum, but nothing that was returning ~~from our~~ . from the recursive call.
As we solved the problem ~~of~~ of comparision before calling the function again.

But in the finding Fibonacci number the result is dependent on previous two numbers addition,

→ we know the counting idea of how it works, lets visualize it . . .

$0 + 1 = 1 + 1 = 2 + 1 =$

$3 + 2$
$= 5 + 3$
$= 8 + 5$
⋮

* follow the colors
to make more sense
of it . . .

How to solve the problem???

think of an array,

0	1	1	2	3	5	8	13

How to come up with end case, ...
we need atleast 2 numbers that come before, to count our way up to given fibonacci number.

So when we solve recursively, we know that previous 2 are must until we can't get or don't have ~~two~~ 2 numbers to return, so...

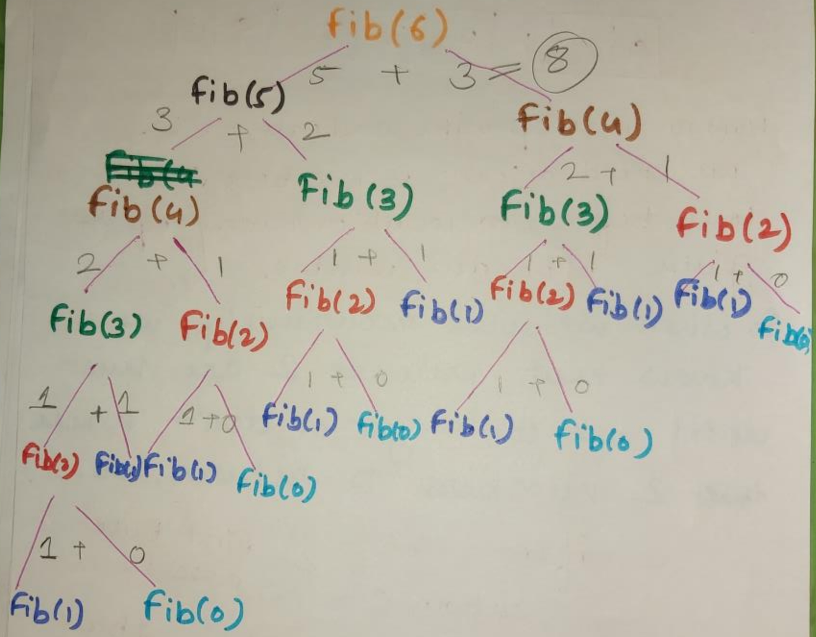
```
if (n==0)
    return 0
```
first base case

```
if (n==1)
    return 1
```
second base case

now remaining numbers are just adding them up, and forward the results ...

→ return fib(n-1) + fib(n-2).

Now lets break the recursion, to understand whats happening . . . .



```
if (a[n] != 0)
    return a[n]
```

lookup for that element, this reduces time for computation.

As it can be observed, each time the number of children double from previous level, so the tree grows exponentially as

$2^1 = 2$
$2^2 = 4$
$2^3 = 8$
$2^4 = 16$ . . . — soon, this will

2^n $O(2^n)$

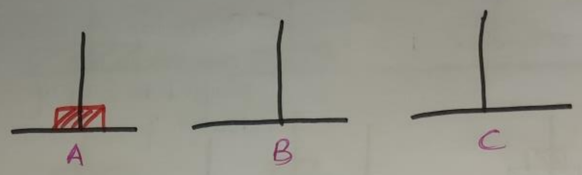
have large number of recursive calls, and computing each one is very time consuming.

To make this linear time, $\Rightarrow O(n)$ we can simply add a data structure to program.

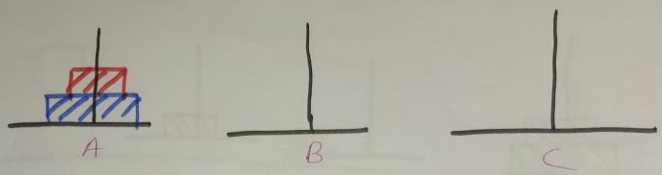
Hence, we take an array of $n+1$ size and store the answer in that place for n^{th} fibonacci number.

$a[0]=0$ $a[1]=1$ $a[2]=1$
$a[3]=2$ $a[4]=3$ $a[5]=5$
. . . . . .

Then add another condition to our base cases from previous program.



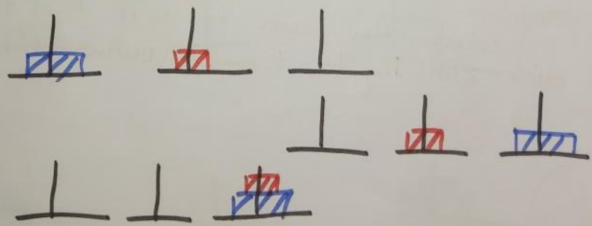
Move $\boxed{\text{Red}}$ disk to C



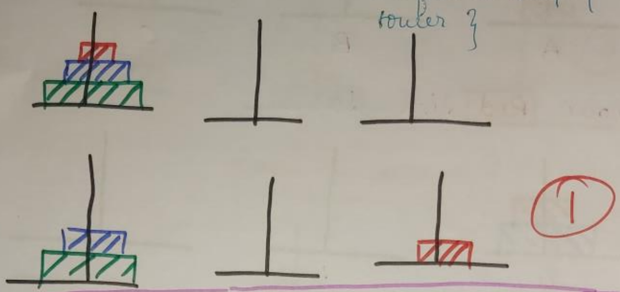
(Here firstly everything above last disk needs to be moved to empty tower, which isn't the final decision)

Move Red to B
Move Blue to C
Move Red to C

TP!! Now. It seems like just moving the top disk to the middle does the Job)



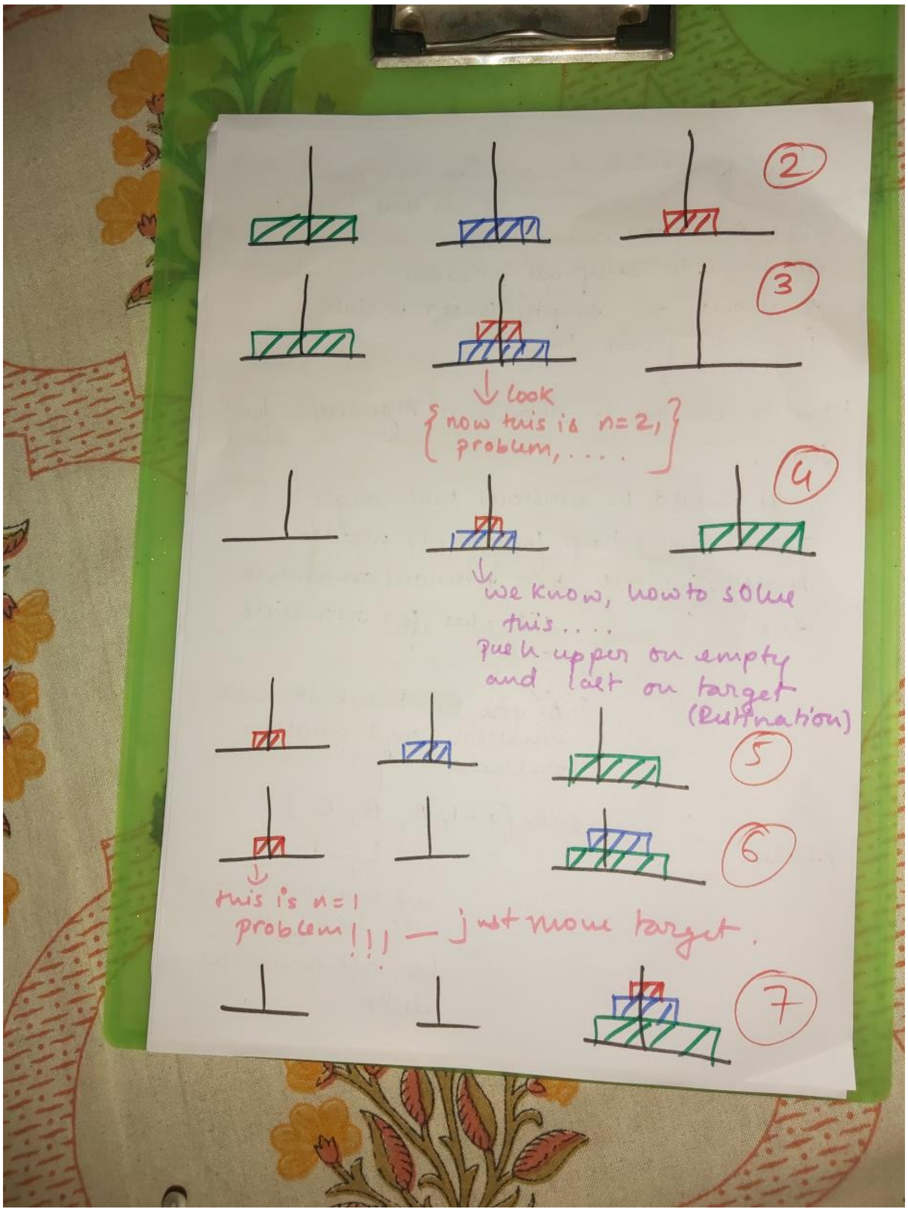
Let see for n = 3, { Remember previous ones are to be shifted to empty tower }



Q. Why move to 3^{rd} tower instead 2^{nd}

Ans:- Because we solve n = 2, problem first and then the answers would help us to move on

↳ This in itself gives rise to n = 1 problem where we have to move the 1^{st} (only) disk directly to tower C (Destination)


2
3
look
now this is n=2, problem, ...
4
we know, how to solve this....
push upper on empty and last on target (destination)
5
6
this is n=1 problem!!! — just move target.
7

Now for $n=4$? . . . I'm not gonna do that. . .

But the idea remains same, you have to setup all previous to move on empty tower which isn't the final target. . . .

How to design a recursive Algorithm for this . . . .

It should be obvious that target should be set up for empty middle tower, for all the previous or above disk. ⟵ 1st clue for recursive call

{ As you can break it into smaller... and smaller problems... }

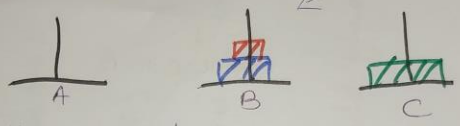
~~At this~~ ∴ placedisks(n-1, A, B, C)

we switched target to be "B" for all previous disks . . . .

Once we reach $n=1$, we know that a move has to be performed . . .

Finally ~~once~~ when we are in a state where 3^{rd} disk ($n=3$) is on destination and 1^{st} & 2^{nd} ~~are~~ are on middle ones. . .

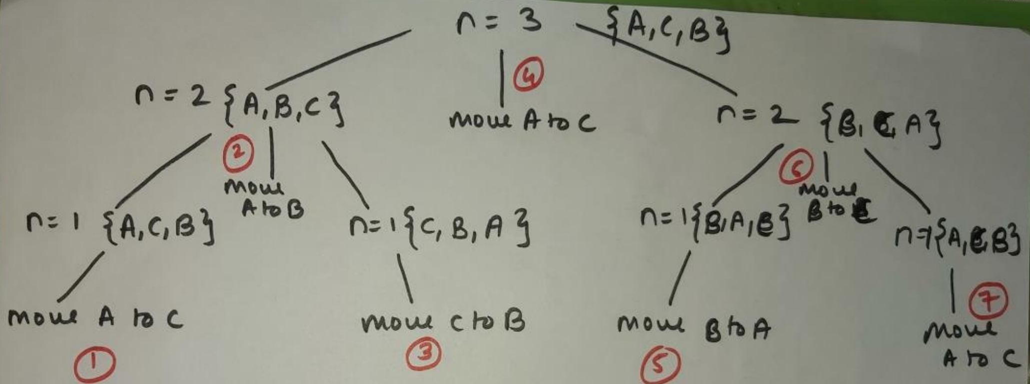
Here the source (Initial position) changes from A to B, Because all the disks are on B . . .



So here we need another recursive call to change the source, and destination

placedisks ($n-1$, B, __ , __)

for $n=2$
{C, A}



→ Try to match with the diagrams, you should find correct answer.

{previously explained}