SCALER

Recursion - 2

Problems on Recursion

Class starts at 7:05 IST

Question - Given two integers a and n, find a^n using recursion.

$a = 2$

$N = 3$

$2^3 = 8$

$2^3 = \underbrace{2 \cdot 2}_{2^2} \cdot 2$

$2^3 = 2^2 \cdot 2$

$a^n = \underbrace{a \cdot a \cdot a \ldots \_ \cdot a}_{a^{n-1}} \cdot a$

$a^n = a^{n-1} \cdot a$

Assumption : Pow(a, n) → calculate & return a^n.

Main logic : Pow(a, n) = Pow(a, n-1) * a

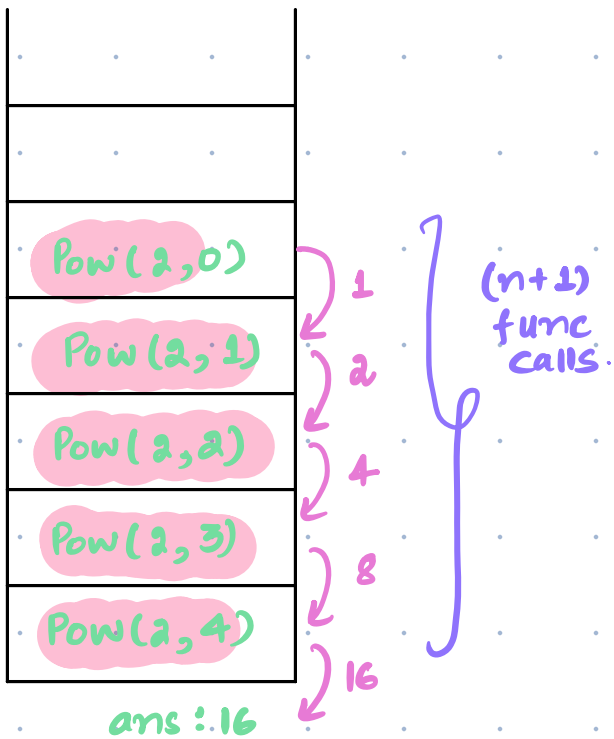
Base Case : If n == 0 return 1 $a^0 = 1$

```
int Pow(a, n) {

    // Base Case
    if (n == 0) return 1;

    return ( Pow(a, n-1) * a);

}
```



2^4 → Pow(2,4)

T.C = No of func call * Time per call

= (N+1) * C

= NC + C

T.C. = O(N)

S.C = Max stack space

= O(N+1)

S.C = O(N)

Approach - 2:

$a^{10} = a^9 . a$

$a^{10} = a^5 . a^5$

$a^{12} = a^6 . a^6$

$a^{11} = a^5 . a^5 . a$

$a^9 = a^4 . a^4 . a$

n (even)

$a^n = a^{n/2} . a^{n/2}$

n (odd)

$a^n = a^{n/2} . a^{n/2} . a$

Assumption : Pow(a, n) → Calculate & return a^n.

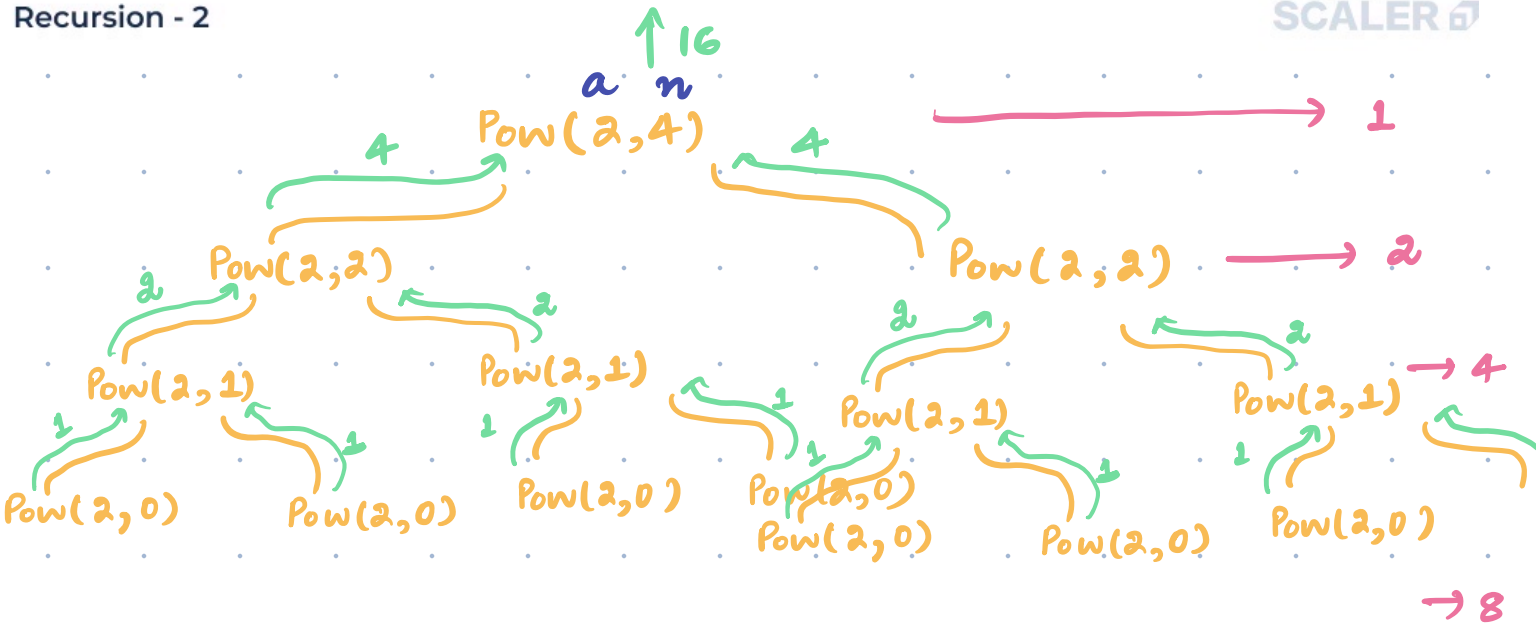
Main logic :

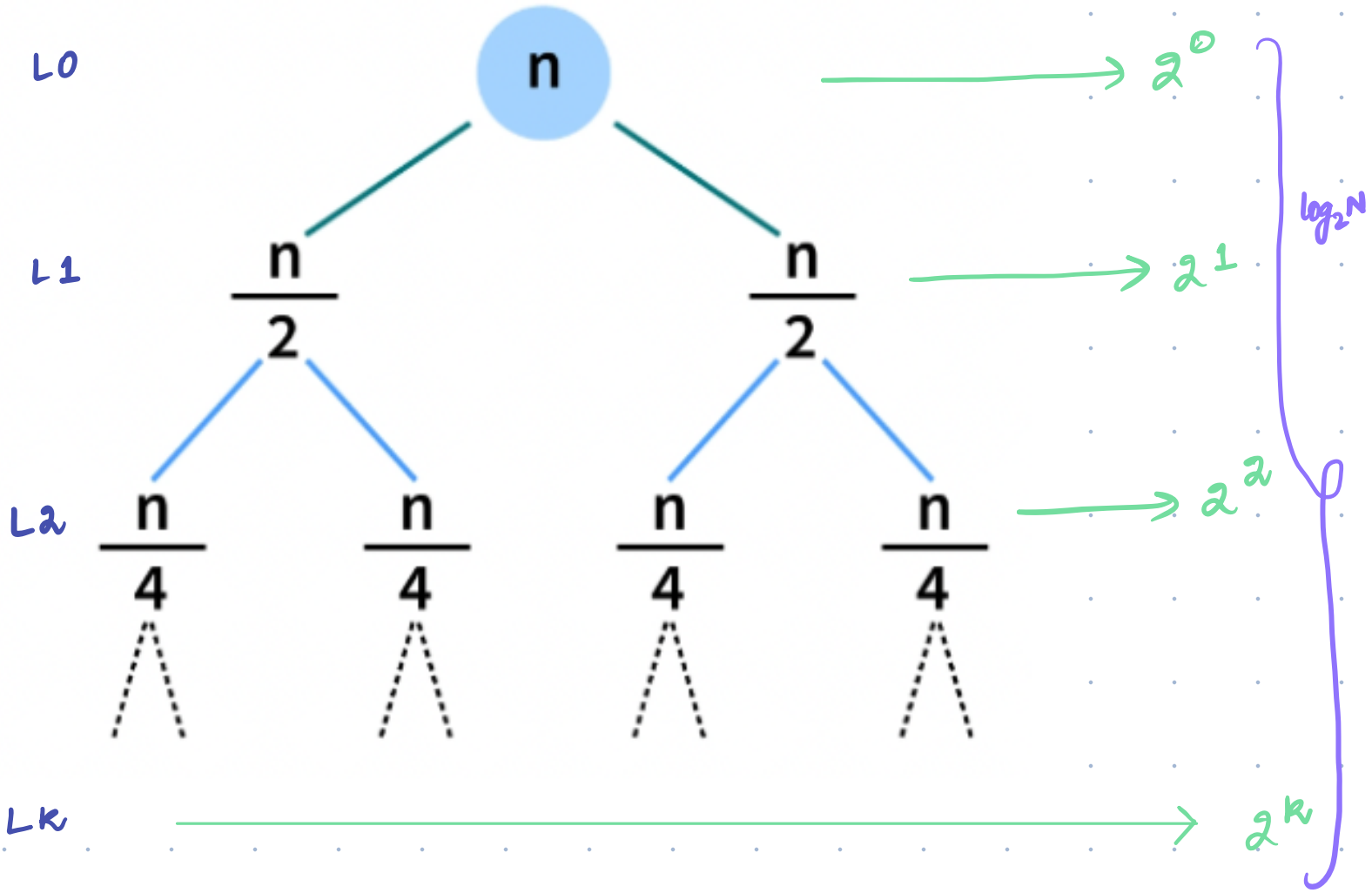
n (even) Pow (a, n) = Pow (a, n/2) * Pow (a, n/2)

n (odd) Pow (a, n) = Pow (a, n/2) * Pow (a, n/2) * a

Base Case : If n == 0 return 1 $a^0 = 1$

```
int Pow(a, n) {
    // Base case
    if (n == 0) return 1;
    if (n%2 == 0)    // n is Even
        return (Pow(a, n/2) * Pow(a, n/2));
    else
        return (Pow(a, n/2) * Pow(a, n/2) * a);
}
```


a
n
16
Pow(2,4)
4
4
1
Pow(2,2)
Pow(2,2)
2
2
2
2
Pow(2,1)
Pow(2,1)
Pow(2,1)
4
1
1
1
1
1
1
Pow(2,0)
Pow(2,0)
Pow(2,0)
Pow(2,0)
Pow(2,0)
Pow(2,0)
8



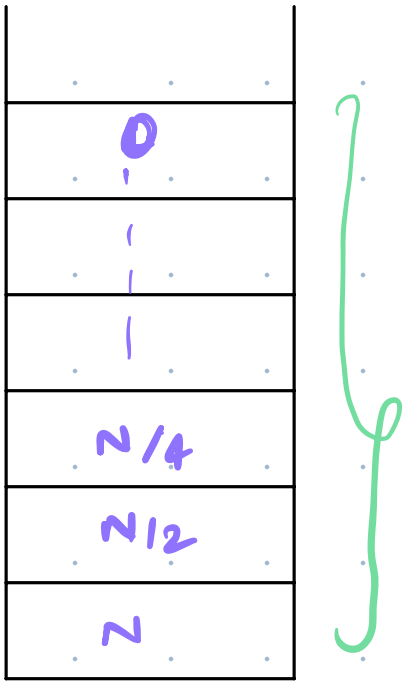
$N \rightarrow N/2 \rightarrow N/4 \; - - - - - - \; 1$

$N/2^0 \rightarrow N/2^1 \rightarrow N/2^2 \; - - - - - \; N/2^k$

$N/2^k = 1$

$N = 2^k$

$\log_2 N = k$



$K = \log_2 N$ calls.

S.C = Max Stack Space
= Max func calls
× Space per func.

S.C : $O(\log_2 N)$

Total func calls $= 2^0 + 2^1 + 2^2 - \cdots - 2^k \rightarrow GP$

$= a\left[\frac{r^n - 1}{r-1}\right] = 1\left[\frac{2^{k+1} - 1}{2-1}\right]$

$= 2^{k+1} - 1$

$K = \log_2 N$ $= 2^{\log_2 N + 1} - 1$

$= 2^{\log_2 N} \cdot 2^1 - 1$

$= N \cdot 2 - 1$

$= 2N - 1$

T.C $= O(N)$

Optimised Approach :

Observation:

Pow(a, n/2) is called twice in above approach.

We can call Pow(a, n/2) once & store its result & use that.

Assumption : Pow(a, n) → Calculate & return a^n.

Main logic : Calculate Pow(a, n/2) and store result in p.

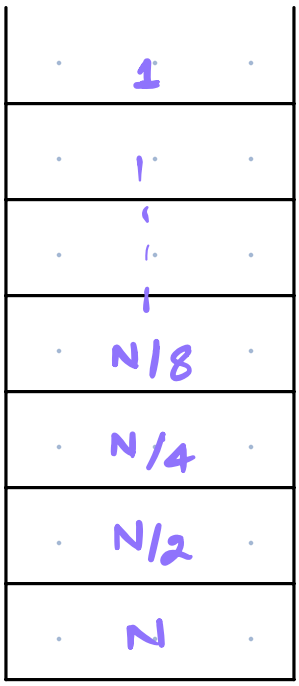
n : even → return p * p

n : odd → return p * p * a

Base Case : If n == 0 return 1 $\quad a^0 = 1$

```
int Pow(a, n) {
    // Base case
    if (n == 0) return 1;
    int p = Pow(a, n/2);
    if (n%2 == 0)    // n is Even
        return p*p;
    else
        return p*p*a;
}
```

Fast Power
or
Fast Exponentian



$(\log_2 N)$ calls

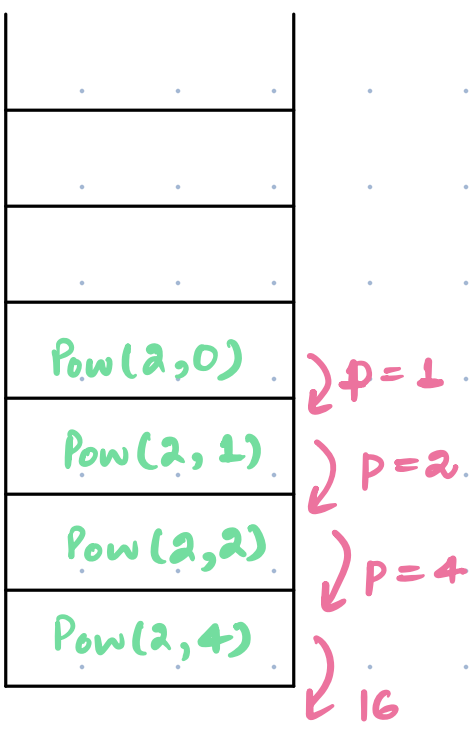
$T.C = \log_2 N \cdot C$

$= O(\log_2 N)$

$S.C = O(\log_2 N)$

$2^4 \rightarrow Pow(2,4)$

```
int Pow(a, n) {
    // Base case
    if (n==0) return 1;
    int p = Pow(a, n/2);
    if (n%2==0)    // n is Even
        return p*p;
    else
        return p*p*a;
}
```



O/p : 16.

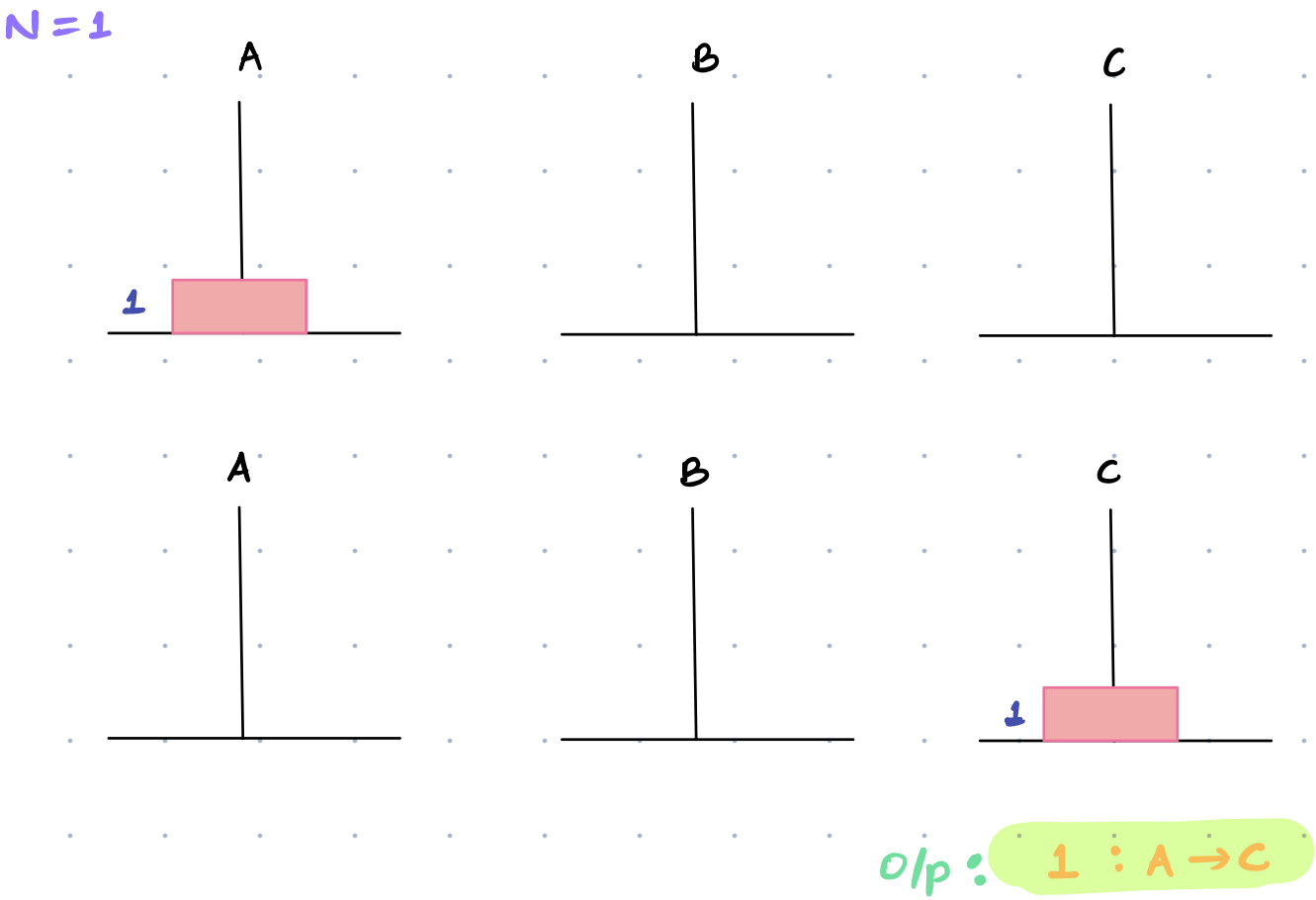
Tower of Hanoi

- Given 3 towers A, B and C,
- There are n-disks placed on tower A of different sizes,
- Move all the disks from A to C {using tower B if needed}

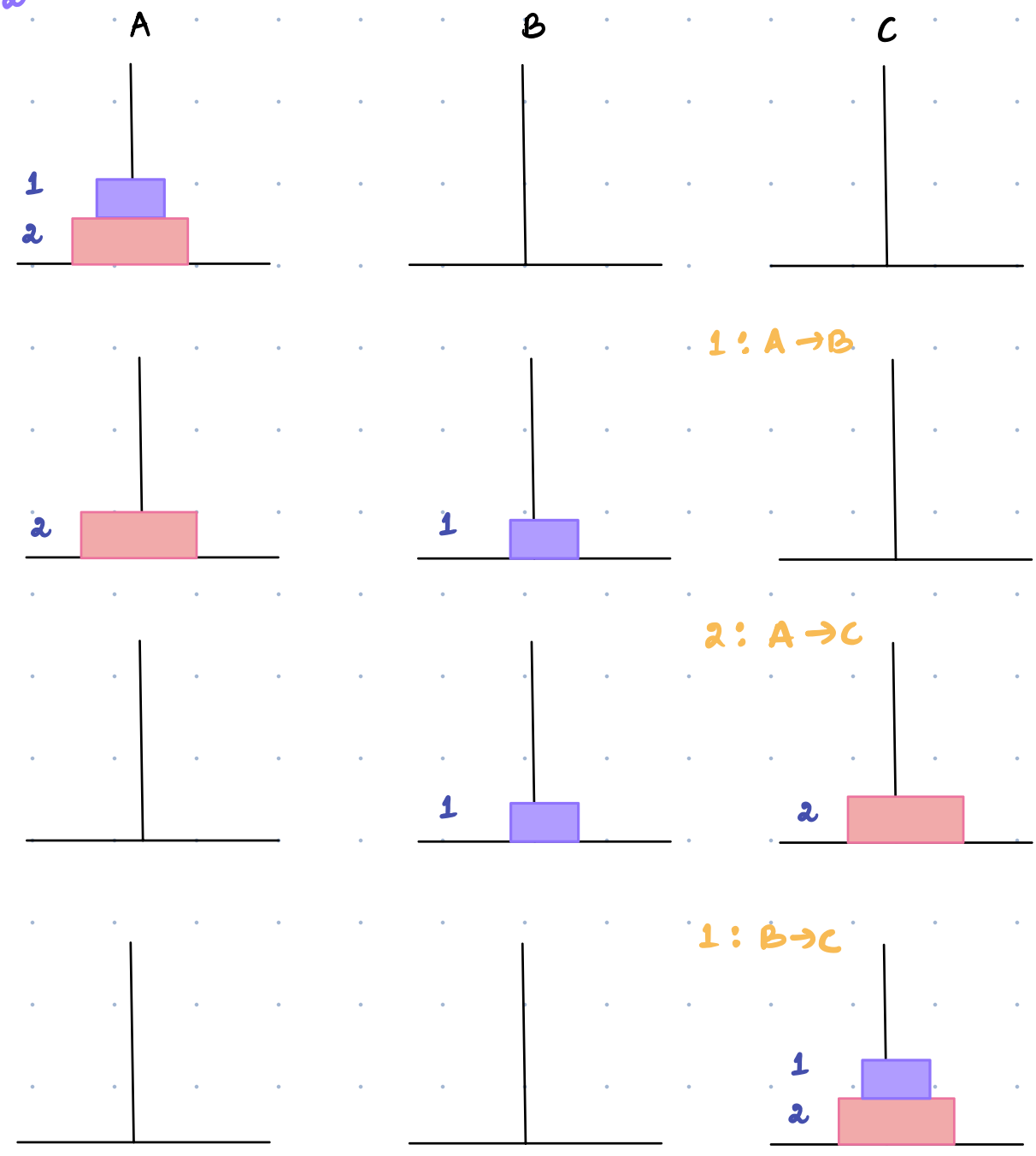
Note :

1. Only one disk can be moved at a time (Topmost)
2. Larger disk can't be placed on a smaller disk.

< Question > **:** Print movement of the disks from A to C in minimum steps.



N = 2



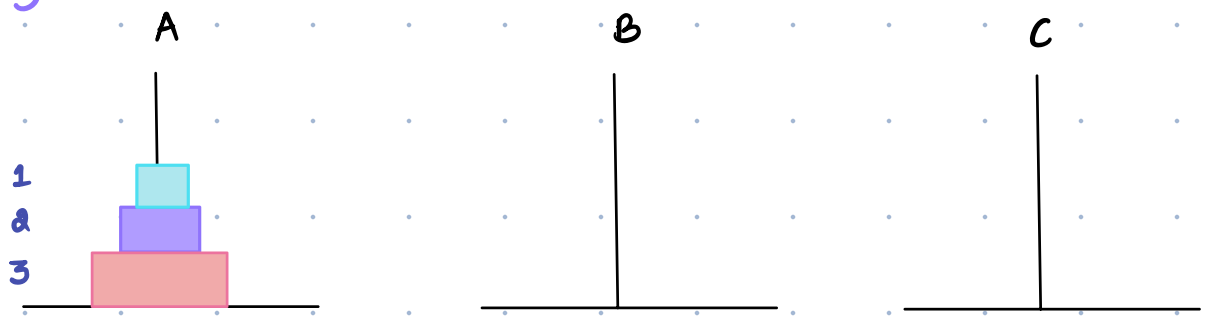
o/p :

1 : A → B

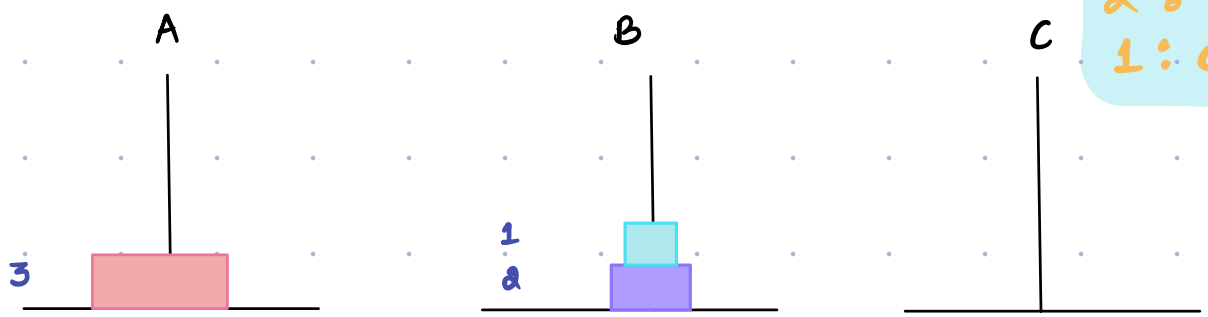
2 : A → C

1 : B → C

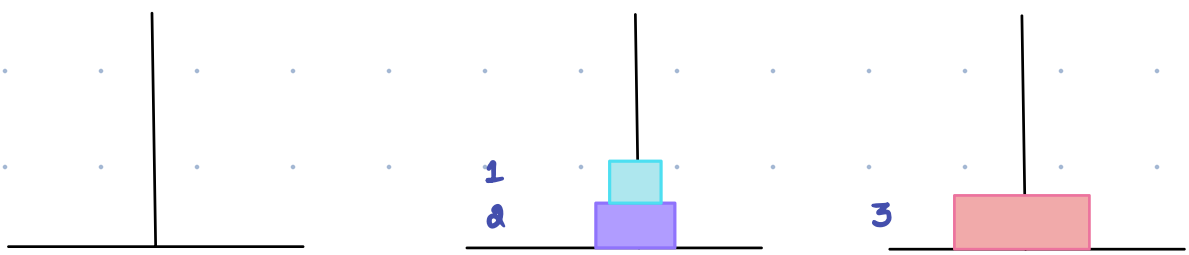
N = 3



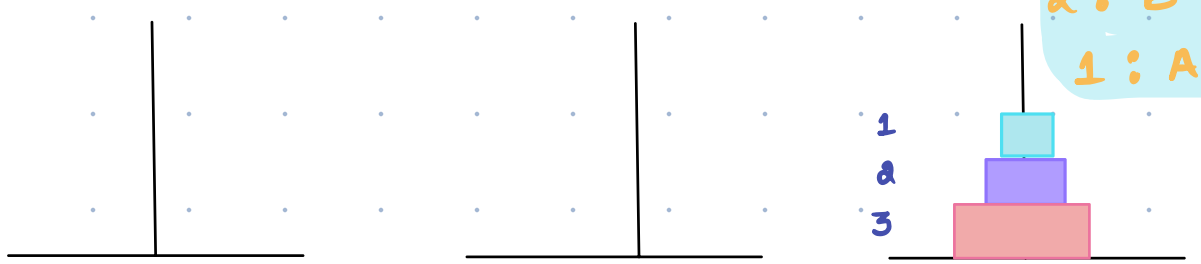
1 : A → C
2 : A → B
1 : C → B



3 : A → C

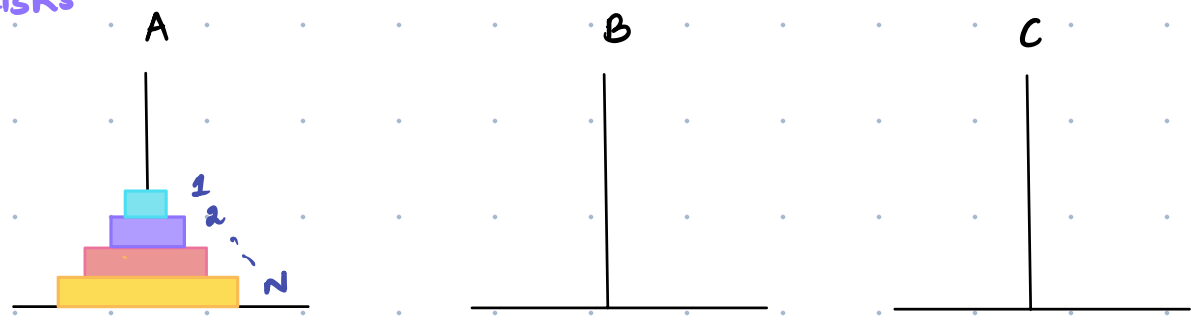


1 : B → A
2 : B → C
1 : A → C

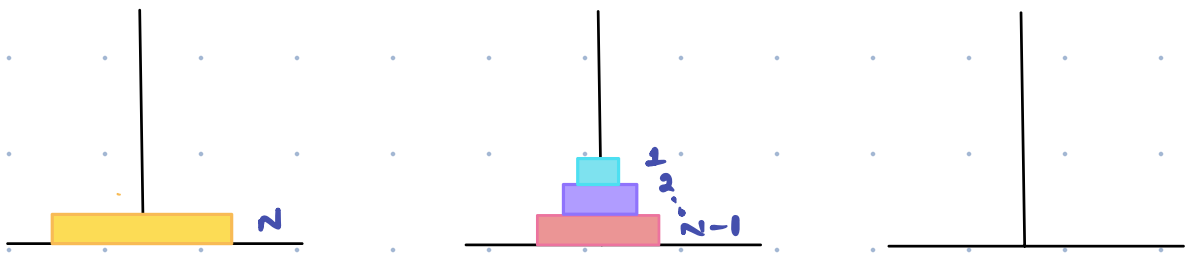


O/p :

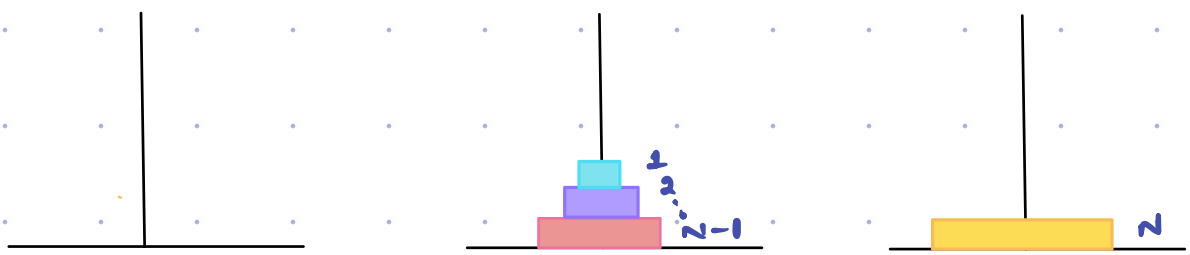
N disks



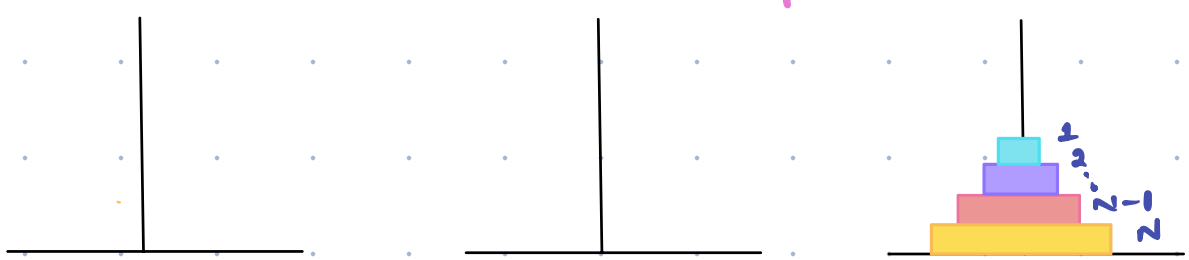
1. Move (N-1) disks from A → B
(src) → (helper)



2. Move N^{th} disk from A → C
(src) → (dest^n)



3. Move (N-1) disk from B → C
(helper) → (dest^n)



Assumption : TOH(n, A, B, C) (src, helper, dest^n) → minimum steps for movement of disks from A → C.

Main logic :

1. Move (N-1) disks from A → B (src → helper)

2. Move N^{th} disk from A → C (src → dest^n)

3. Move (N-1) disks from B → C (helper → dest^n)

Base case : N = 0 return

```
void TOH(int n, int A(s), int B(h), int C(d)){
    if(n==0) return;

    TOH(n-1, A, C, B);
    print(n : A → C);
    TOH(n-1, B, A, C);
}
```

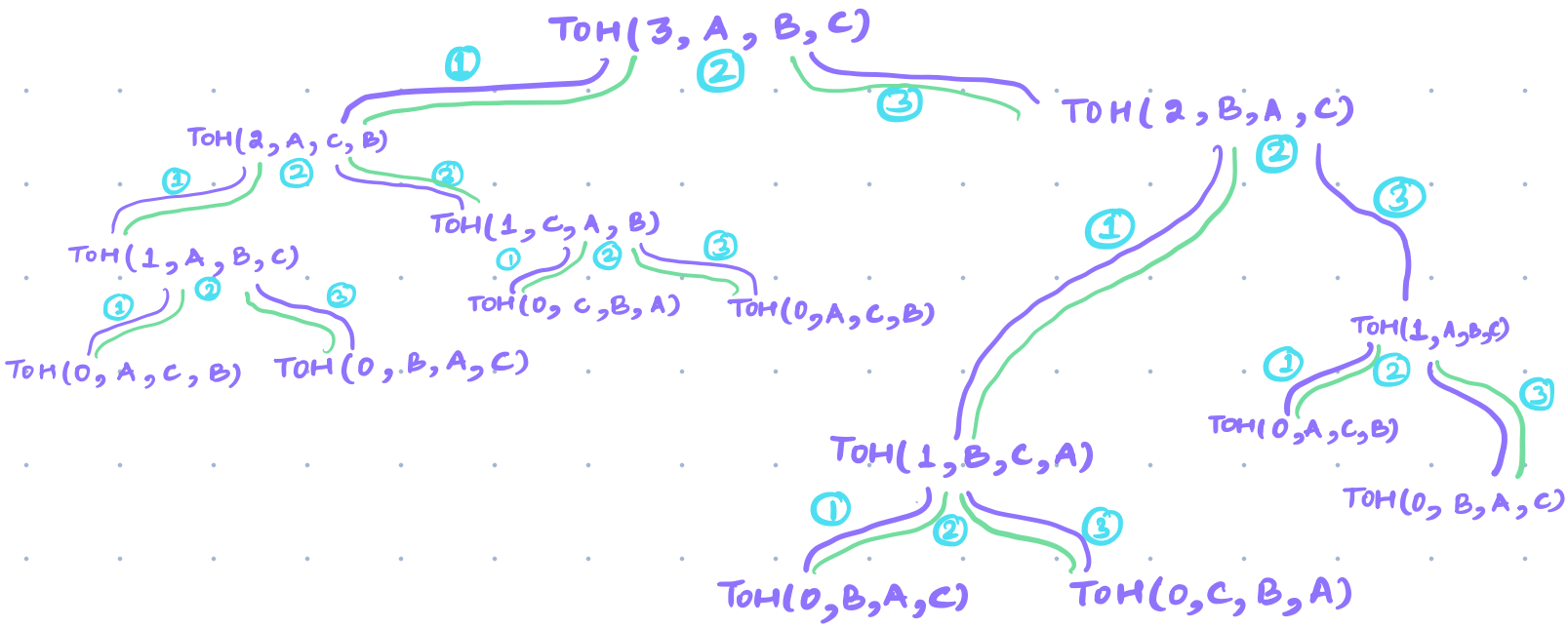
A : src B : helper C : dest"

*Dry-Run

```
TOH(n-1, A, C, B);   left call : TOH(n-1, src, helper ↔ dest")
print(n : A → C);
TOH(n-1, B, A, C);   right call : TOH(n-1, src ↔ helper, dest")
```

```
void TOH(int n, int A, int B, int C) {
    if (n == 0) return;

    TOH(n-1, A, C, B);   1
    print(n : A → C);    2
    TOH(n-1, B, A, C);   3
}
```



1 : A → C
2 : A → B
1 : C → B
3 : A → C
1 : B → A
2 : B → C
1 : A → C

L0 $\rightarrow 2^0$

L1 $\rightarrow 2^1$

L2 $\rightarrow 2^2$

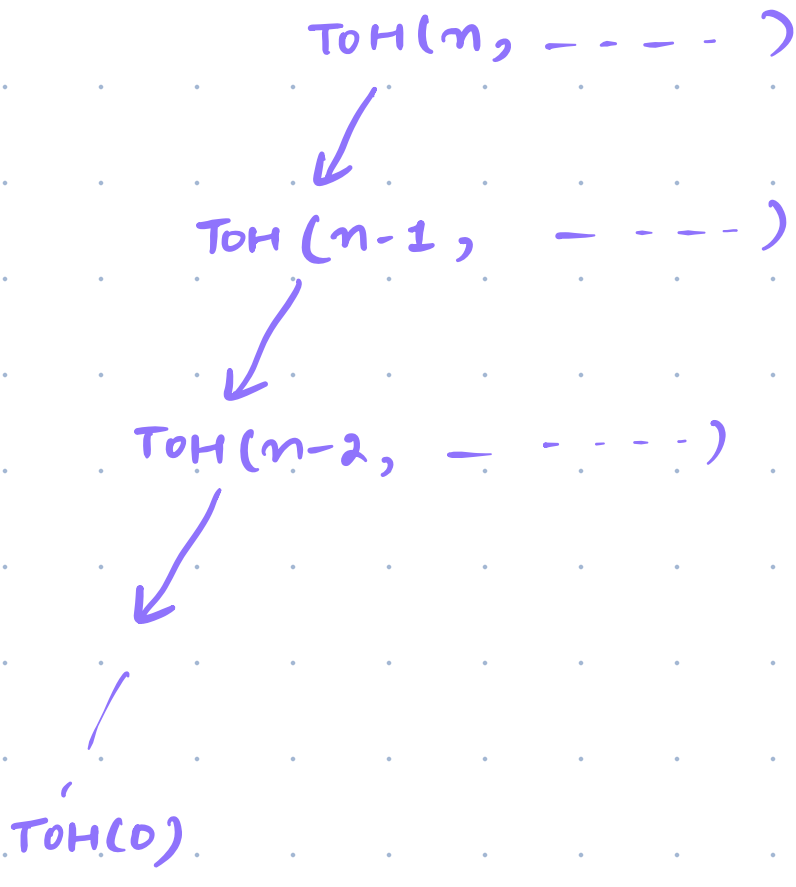
LN $\rightarrow 2^N$

Total calls: $2^0 + 2^1 \cdots 2^N$

$= 2^{N+1} - 1$

T.C = $O(2^{N+1} - 1)$

T.C = $O(2^N)$



(n+1) calls.

S.C : O(N+1)

S.C. = O(N)

Question - Given an array of integers, write a recursive function to find the maximum element in the array.

will cover in Thursday's class.