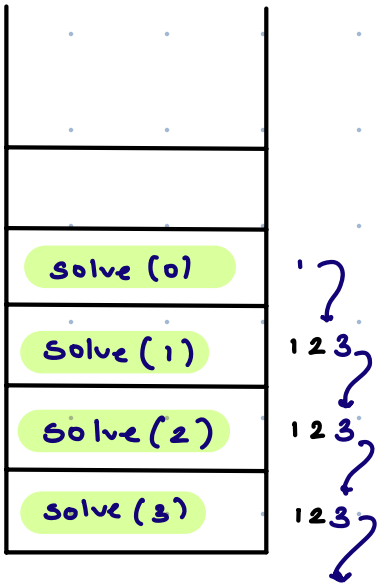
Recursion 2

Content

01. Quizzes
02. Pow function
03. Tower of Hanoi — Pure recursion
04. All indices in an Array

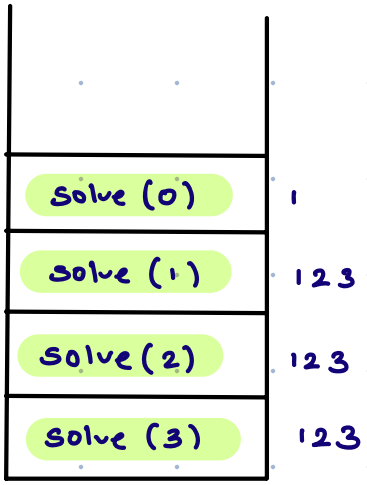
```
01. void solve (int N)        ← N=3
    |  if (N==0) return          1
    |  solve (N-1)               2
    |  print (N)                 3
    3
```

Ans = 1 2 3

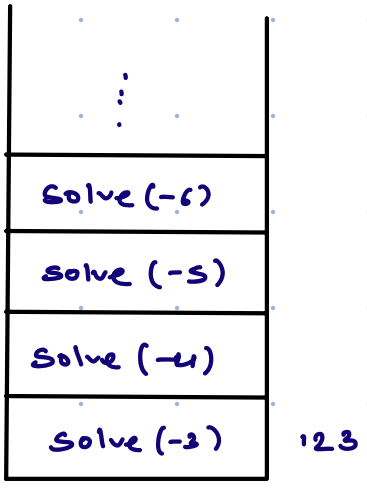



```
02. void solve (int N)        ← 3
    |  if (N==0) return;        1
    |  print (N);               2
    |  solve (N-1);             3
    3
```

Ans = 3 2 1




```
03. void solve (int N)        ← -3
    |  if (N==0) return;        1
    |  print (N)                2
    |  solve (N-1)              3
    3
```



Ans = Error (Stack overflow)

Limit for calls in stack ≈ $10^5 - 10^6$ calls

01. Power function

Given two integers a & n. Find a^n

$a = 2$

$n = 3$

Ans = $2^3 = 2 * 2 * 2 = 8$

$2^4 = \underbrace{2 * 2 * 2}_{2^3} * 2$

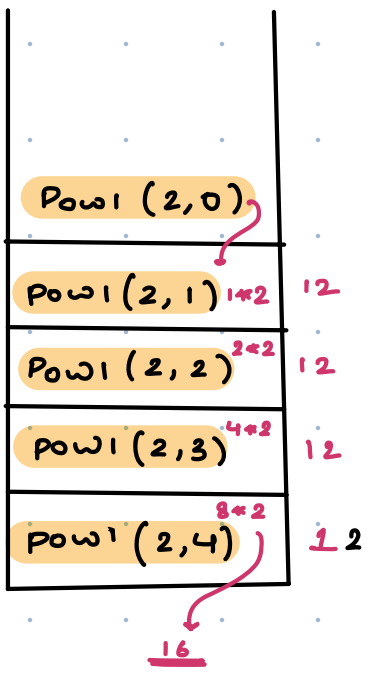
$a^n = \underbrace{a * a * a * \ldots a}_{a^{n-1}} * a$

$a^n = a^{n-1} * a$

$a^0 = 1$

```
int pow1 (a, n) {
    if (n==0) return 1          1
    return pow1(a, n-1) * a     2
}
```

a = 2
n = 4
} $2^4 = 16$



TC : No. of calls → N+1

TC : O(N)

SC : O(N)

* Approach 2

$a^{10} = a^9 * a^1$

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

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

$a_{11} = a_5 * a_5 * a$

$a_7 = a_3 * a_3 * a$

$a_{15} = a_7 * a_7 * a$

if n is even ↓

pow(a,n) = pow(a, n/2) * pow(a, n/2)

if n is odd ↓

pow(a,n) = pow(a, n/2) * pow(a, n/2) * a;

```
int pow2(a, n)
{
    if (n==0) return 1

    if (n%2==0){
        return pow2(a, n/2) * pow2(a, n/2);
    }
    else{
        return pow2(a, n/2) * pow2(a, n/2) * a ;
    }
}
```

// Assume pow2(a, n) ⟶ T(n)

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

$$T(n) = 2T(n/2) + 1 \quad \longrightarrow \text{I}^{\text{st}} \text{ eqn}$$

// $2^1 T(\frac{n}{2^1}) + 2^1 - 1$

$$T(n/2) = 2T(n/4) + 1$$

$$T(n) = 2\left(2T(n/4) + 1\right) + 1$$

$$T(n) = 4T(n/4) + 3 \quad \longrightarrow \text{II}^{\text{nd}} \text{ eqn}$$

// $2^2 T(\frac{n}{2^2}) + 2^2 - 1$

$$T(n/4) = 2T(n/8) + 1$$

$$T(n) = 4\left(2T(n/8)+1\right) + 3$$

$$T(n) = 8T(n/8) + 7 \longrightarrow \text{III}$$

// $2^3 T(\frac{n}{2^3}) + 2^3 - 1$

After K iteration

$$T(n) = 2^K T\left(\frac{n}{2^K}\right) + 2^K - 1$$

$T(0) = 1$

$T(1) = 1$

$\frac{n}{2^K} = 0 \longleftrightarrow$ Can't get any result

$\frac{n}{2^K} = 1$

$n = 2^K \longleftrightarrow K = \log_2 n$

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

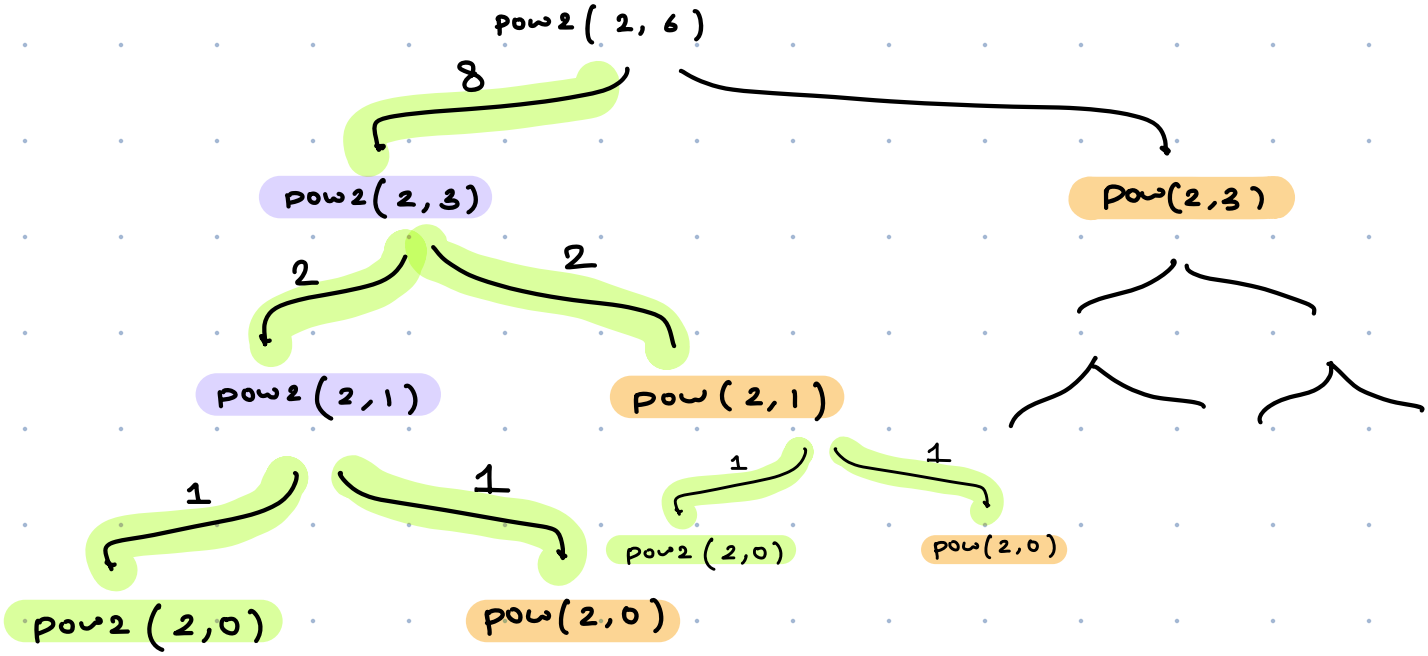
$T(n) = 2n$

TC = O(n)

SC = O(log n)

```
int pow2(a,n)
{
    if(n==0) return 1
    if(n%2==0){
        return pow2(a,n/2) * pow2(a,n/2);
    }
    else{
        return pow2(a,n/2) * pow2(a,n/2) * a ;
    }
}
```

a = 2
n = 6



* Approach 3

```
int fastpow(a,n)
{
    if(n==0) return 1;
    int x= fastpow(a,n/2)
    if(n%2==0) return x*x;
    else return x*x*a;
}
```

TC : O(log N)

SC : O(log N)

* Lets assume fastpow(n) → T(n)

$T(n) = T(n/2) + 1$ — Recurrence relation

$T(n/2) = T(n/4) + 1$

$T(n) = T(n/4) + 2$

$T(n/4) = T(n/8) + 1$

$T(n) = T(n/8) + 3$

After k^{th} step

$$T(n) = T\left(\frac{n}{2^k}\right) + k \qquad T(1) = 1$$

$$\frac{n}{2^k} = 1 \iff n = 2^k \iff k = \log_2 n$$

$$T(n) = 1 + \log_2 n$$

$$T(n) = 1 + \log_2 n$$

TC : $O(\log_2 n)$

SC : $O(\log_2 n)$

Tower Of Hanoi

There are n disks placed on tower A of different sizes

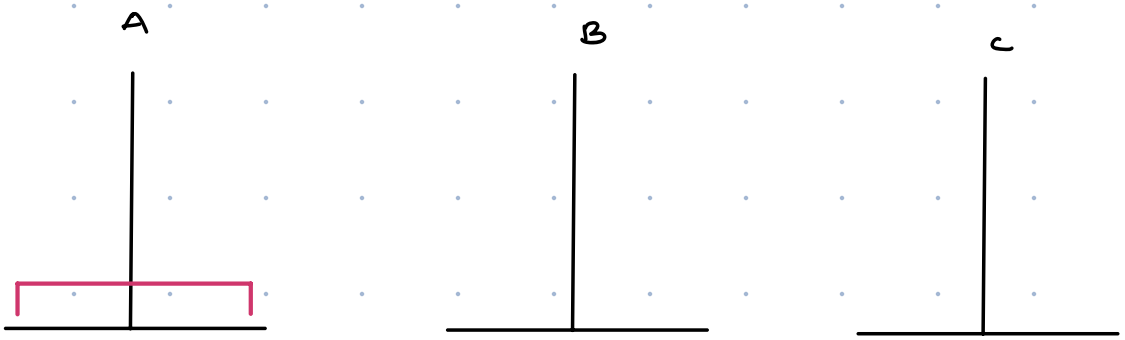
Goal → Move all disks from tower A to C using tower B if needed

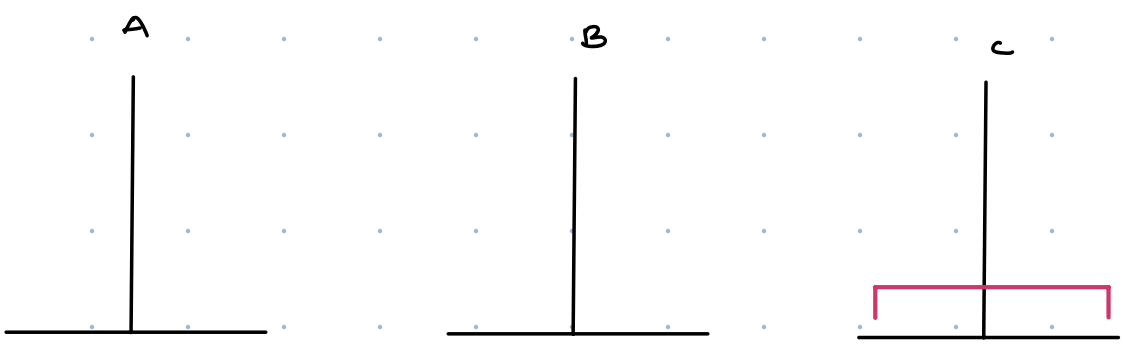
Constraint →

01. Only 1 disk can be moved at a time
02. Larger disk can't be placed at smaller disk at any step.

Print the movement of disks from A to C in minimum steps

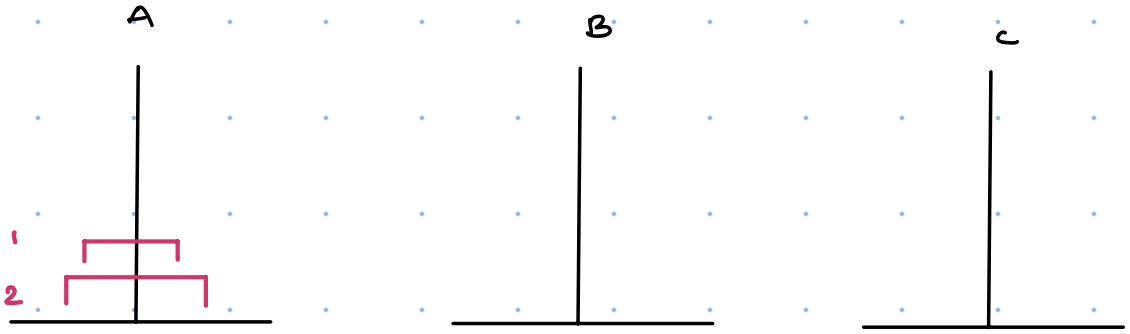
n = 1



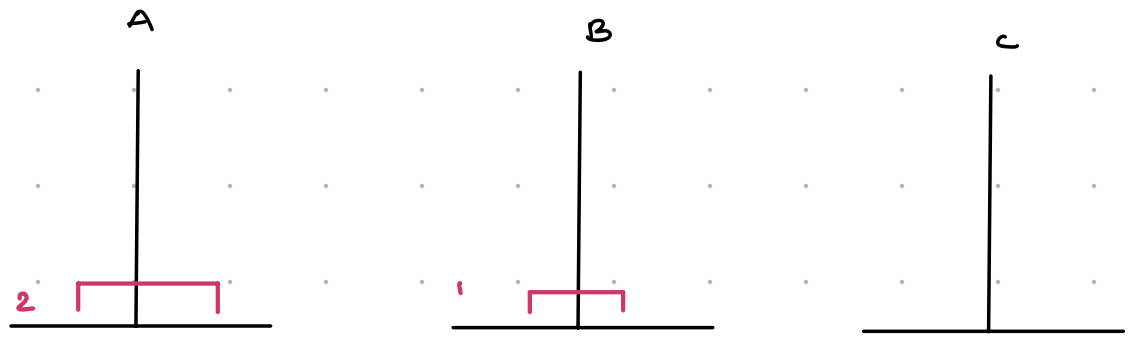


Output : 1 : A → C

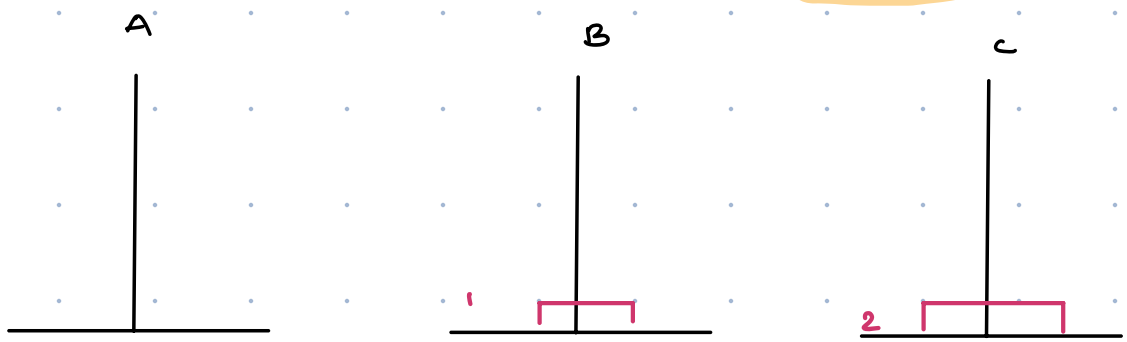
n = 2



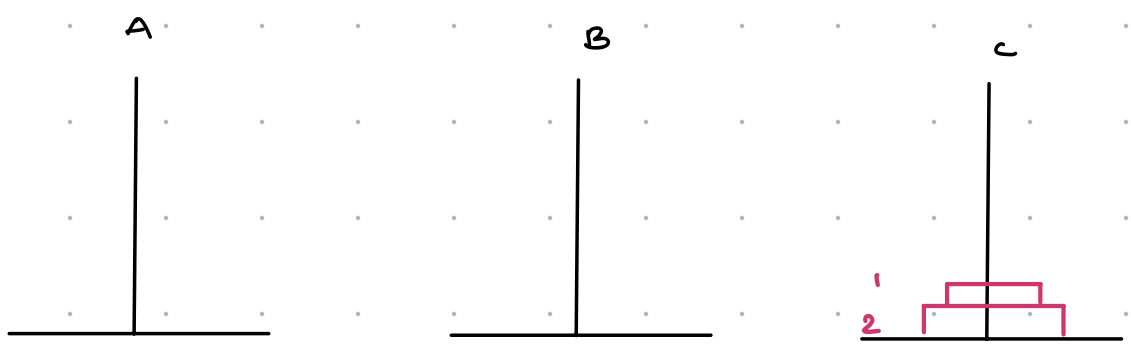
1 : A → B



2 : A → C



1 : B → C



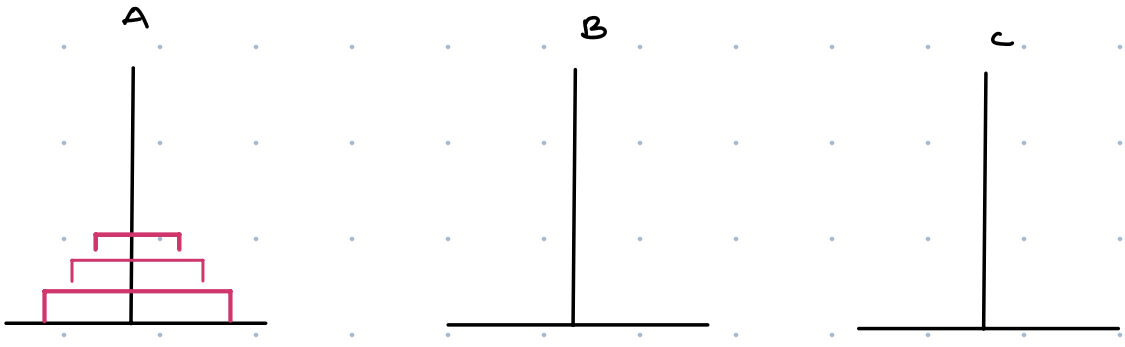
Output :

1 : A → B

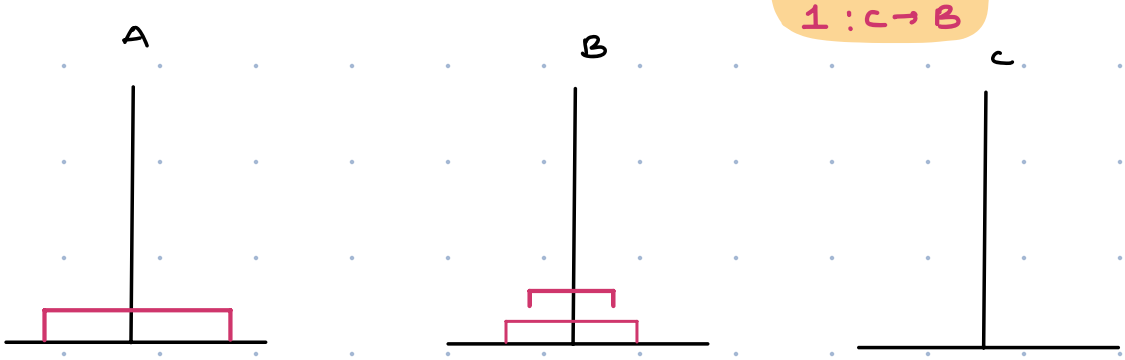
2 : A → C

1 : B → C

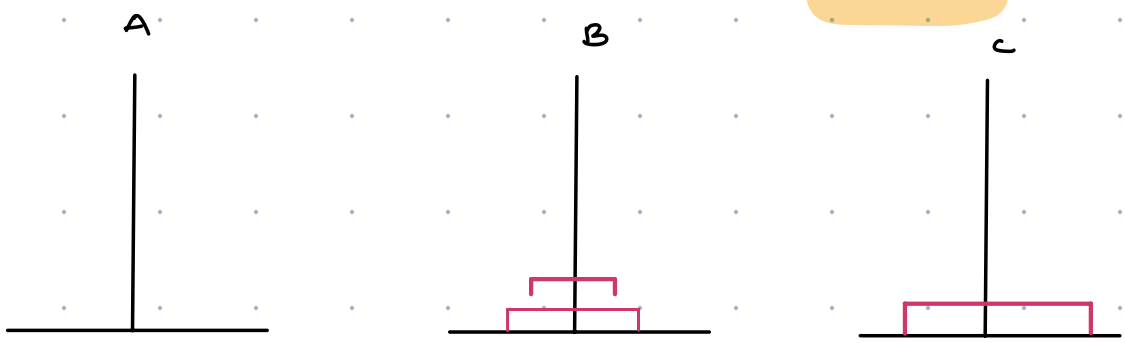
N = 3



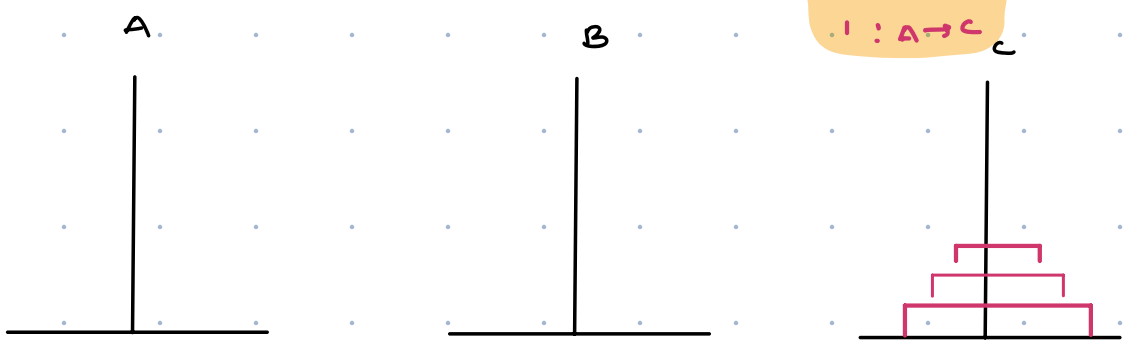
1 : A → C
2 : A → B
1 : C → B



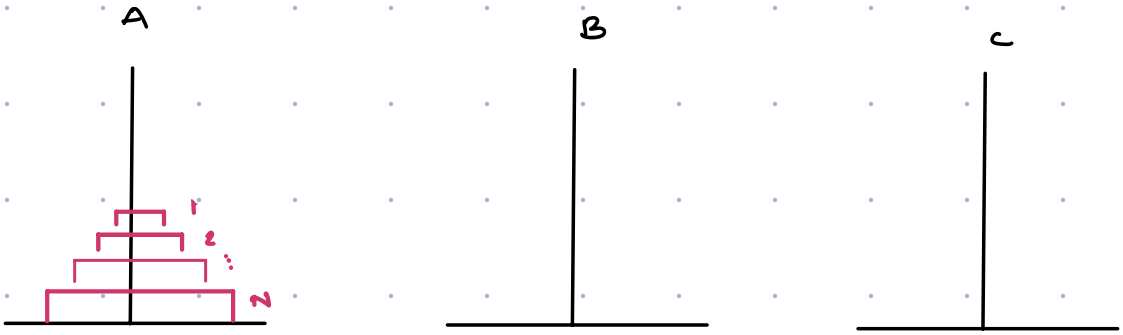
3 : A → C



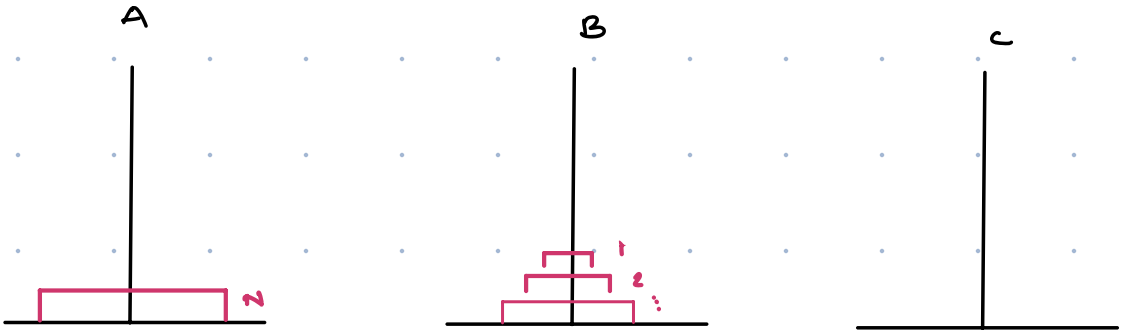
1 : B → A
2 : B → C
1 : A → C



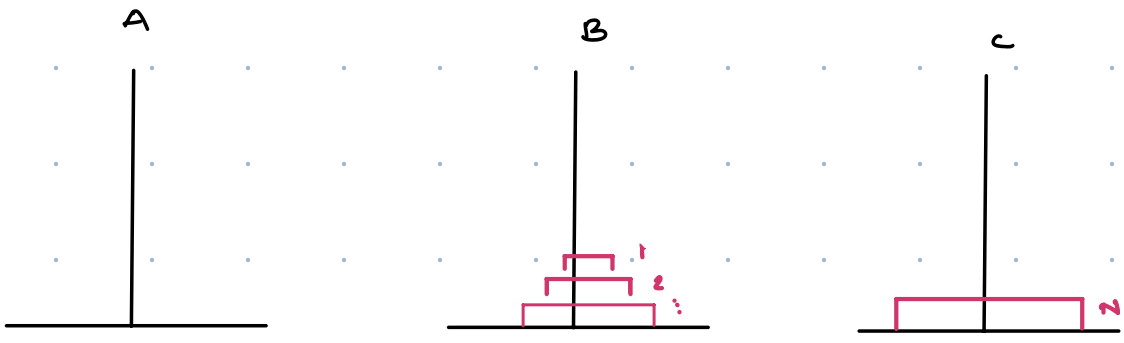
N disks



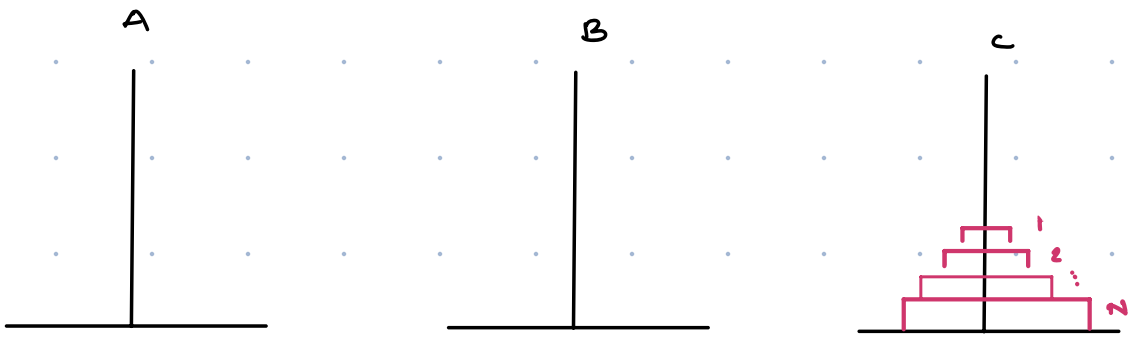
01. Move N-1 disks A → B (src → helper)



02 Move N^th disk from A → C (src → dest)



03. Move N-1 disks from B → C (helper → dest)

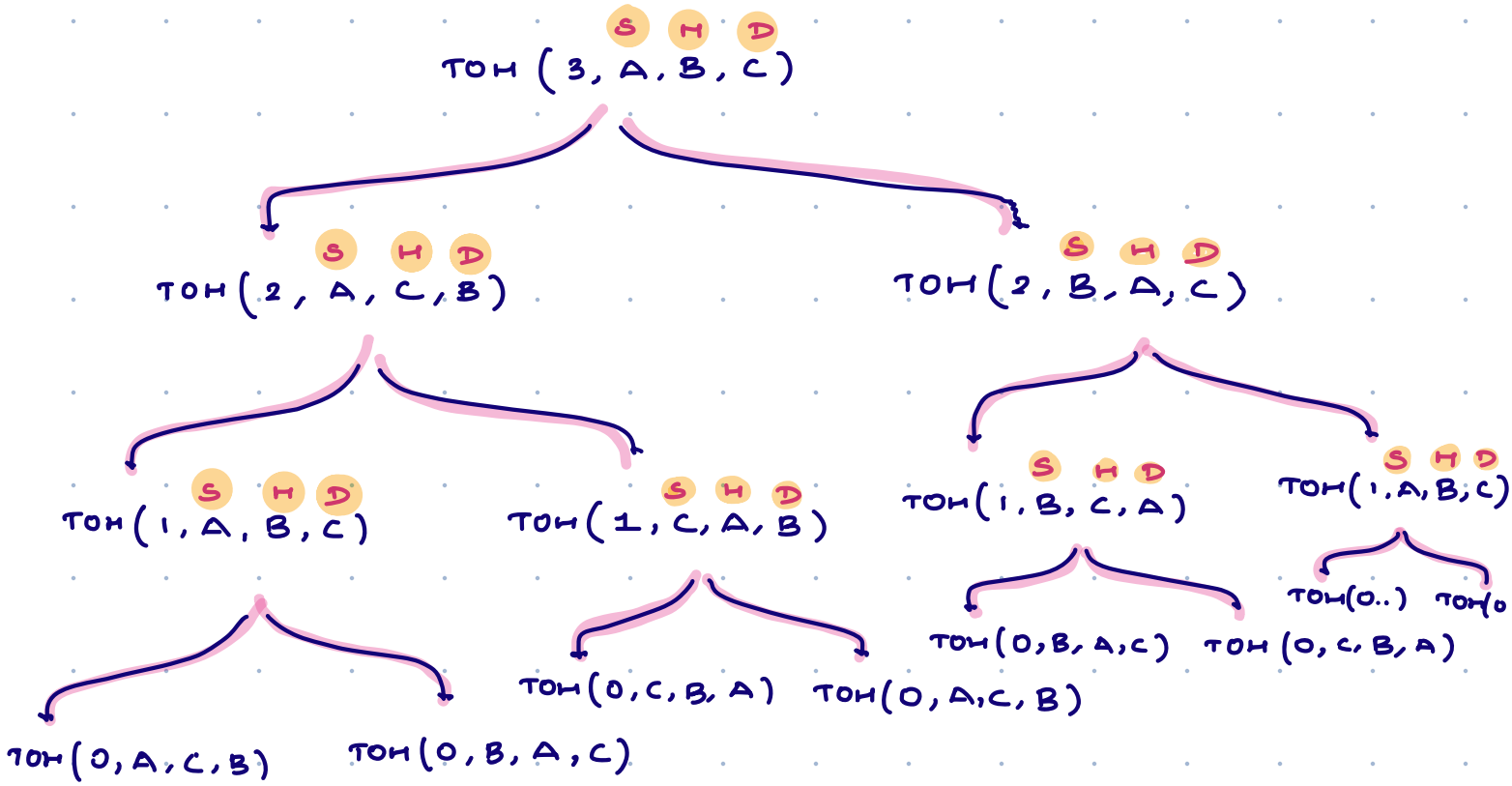


```
void TOH (int n, int A, int B, int C)
{
    if (n==0) return;
    TOH (n-1, A, C, B);
    print (n : A → C);
    TOH (n-1, B, A, C)
}
```

S H D

* To create my tree

- left call = TOH (n-1, src, dest ⟷ helper)
  - print (n : src → dest)
- right call : TOH (n-1, helper ⟷ src, dest)



10:29 → 10:39 pm

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

N	steps
1	1 = $2^1 - 1$
2	3 = $2^2 - 1$
3	7 = $2^3 - 1$
⋮	⋮
N	$2^N - 1$

TC : $O(2^n)$

SC : $O(n)$

All Indices of Array

Given an integer array A with N elements & a target value B, the task is to find all the indices at which B occurs in the array.

It is guaranteed that B exist atleast once.

A[] = { 4 5 3 1 5 4 5 } B = 5
0 1 2 3 4 5 6

Output :

1	4	6
0	1	2

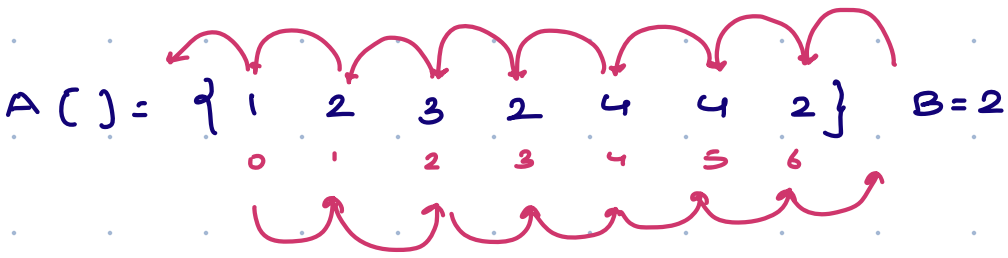
A[] = { 1, 2, 3, 1, 1 } B = 1
0 1 2 3 4

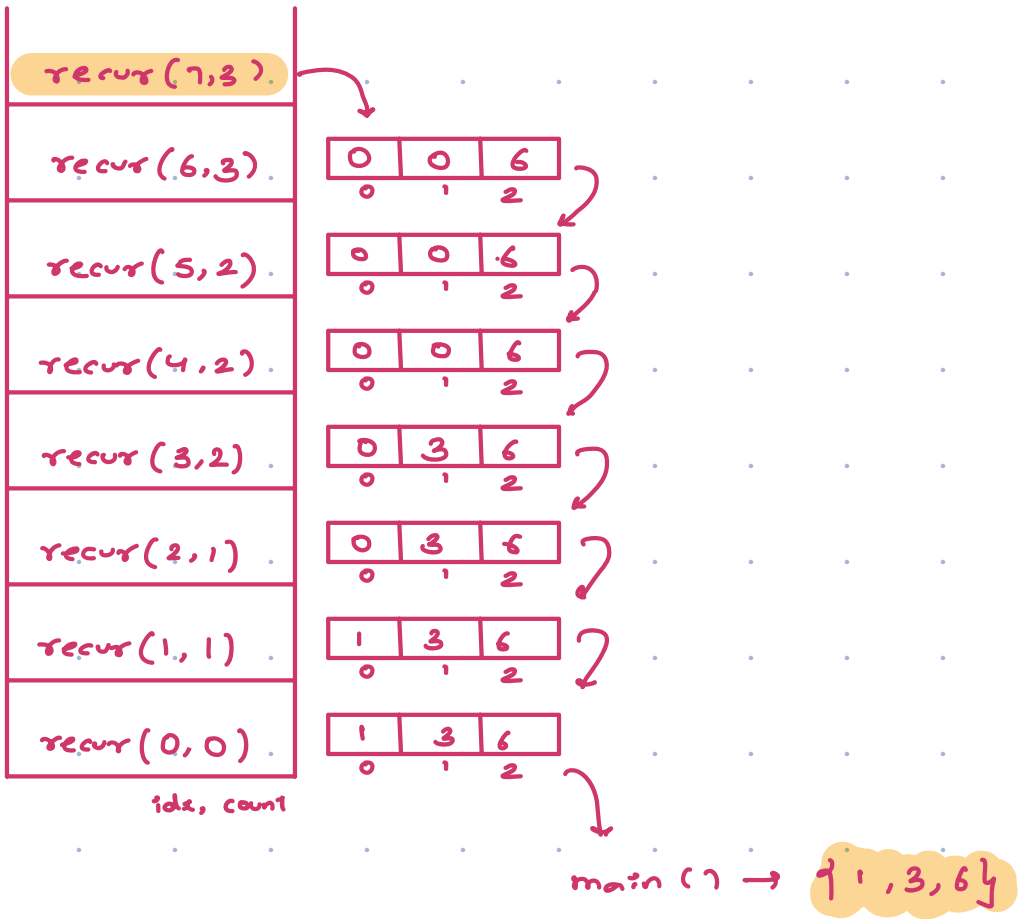
Output :

0	3	4
0	1	2

recursion (A[], B, idx, count)
0 0

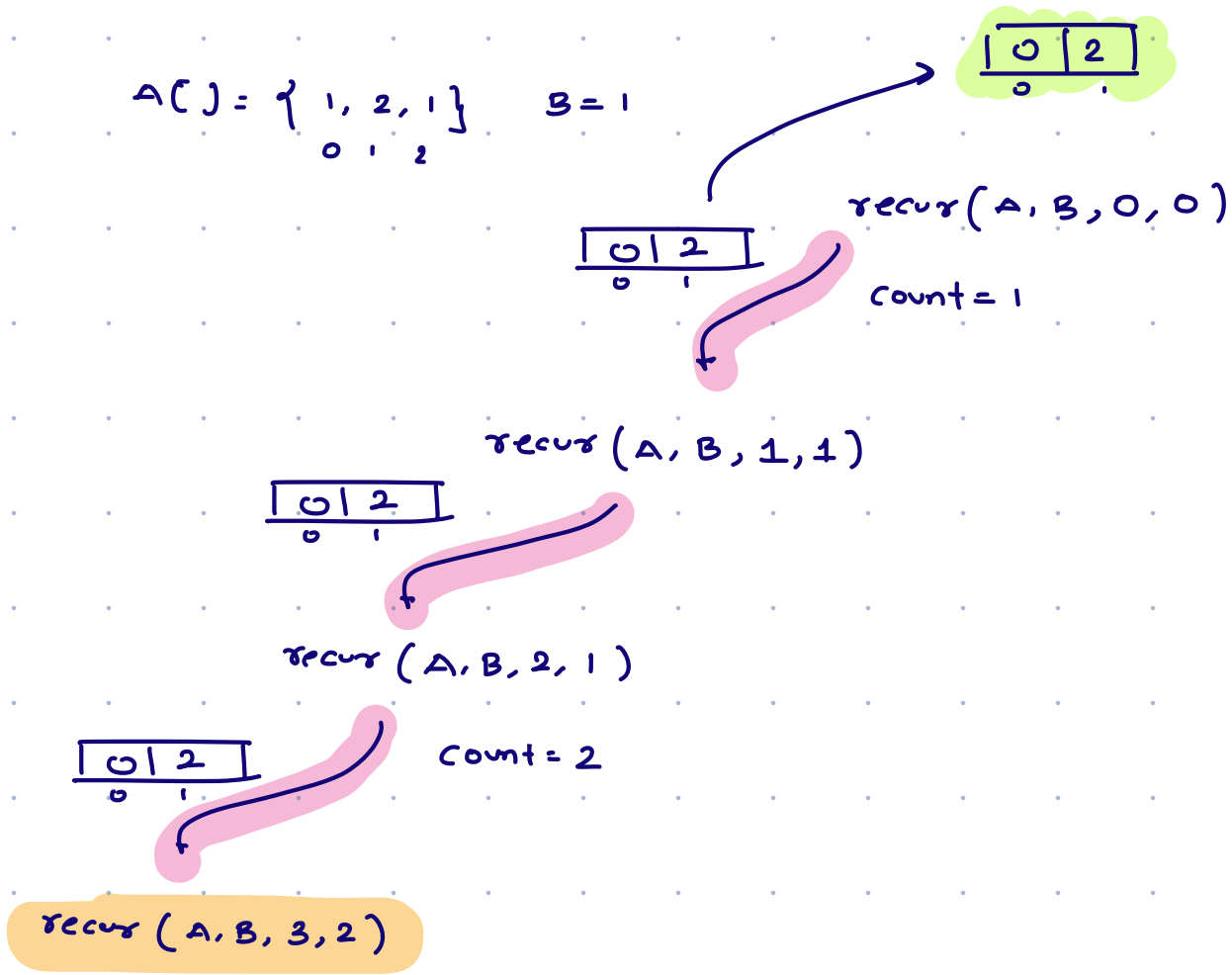
01. Count the no. of times B is occurring in array
02. if (idx == n) → return ans arr[count]
03. while coming back, fill ans array.

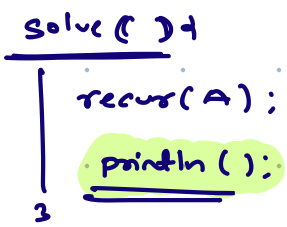


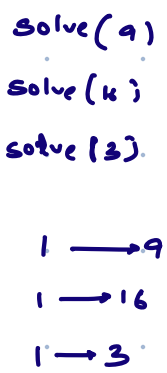


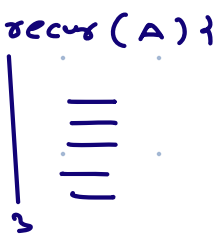
0 0

```
int [] recur(A[], B, idx, count)
    if (idx == n) {
        return new int[count];
    }
    if (A[idx] == B) count = count + 1;
    int[] ans = recur(A, B, idx+1, count);
    if (A[idx] == B) {
        ans[count-1] = idx;
    }
    return ans;
}
```


A[] = { 1, 2, 1 }
0 1 2
B = 1
0 2
0 1
recur(A, B, 0, 0)
count = 1
recur (A, B, 1, 1)
recur (A, B, 2, 1)
count = 2
recur (A, B, 3, 2)


solve( ){
recur(A);
println( );
3


solve(9)
solve(6);
solve(3)
1 → 9
1 → 16
1 → 3


recur(A){
3