Recursion

Good Evening

Today's content
- → Recursion Introduction
- → Fibonacci series
- → Output based question
- → Tower of Hanoi
- → Generate paranthesis

Recursion → Function calling itself

→ Solving curr problem with the help of subproblem / smaller instance of problem

Q → Sum of first n natural number

$$sum(5) = \underbrace{1+2+3+4}_{sum(4)} + 5$$

sum(4) is a subproblem

$$sum(n) = \underbrace{1+2+3+4+.....+(n-1)}_{sum(n-1)} + n$$

$$sum(n) = sum(n-1) + n$$

Steps for code

01. **Expectation** = Figure out what your code is expect to do.

02. **Logic** → solving the function expecation by using the subproblem

03. **Base case** → Smallest subproblem for which you know the answer

* Sum of n natural no.

Expectation → Find & return the sum of n natural no

```
int sum(n)
{
    if (n==0) return 0;
    return sum(n-1)+n;
}
```

// Assume time taken by sum(n) = f(n)

$f(n) = f(n-1) + 1 \rightarrow 1$ $\quad f(0) = 1$

$f(n-1) = f(n-1-1) + 1$

$= f(n-2) + 1$

$f(n) = f(n-2) + 1 + 1$

$f(n) = f(n-2) + 2$ — 2^{nd} sub

$f(n-2) = f(n-2-1) + 1$

$= f(n-3) + 1$

$$f(n) = f(n-3) + 1 + 2$$

$$= f(n-3) + 3 \quad \text{— } 3^{rd}$$

After k steps

$$f(n) = f(n-k) + k \qquad f(0) = 1$$

$$n - k = 0$$

$$n = k$$

$$f(n) = 1 + n$$

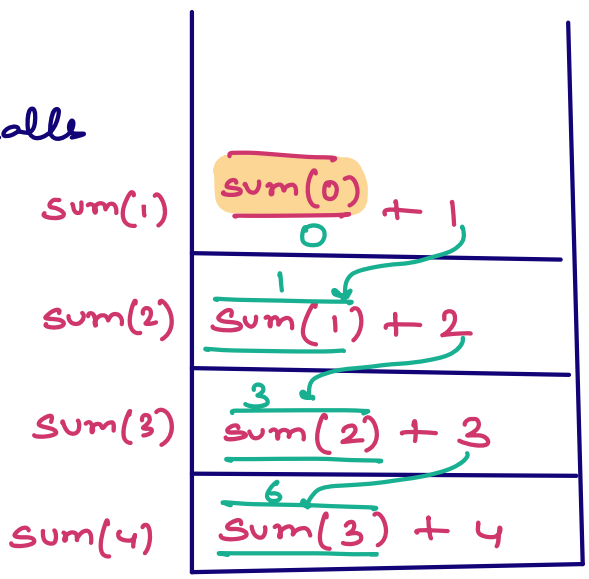
TC ≅ O(n)

* Space complexity.

sum(4) = 4 recursive calls

sum(n) = n recursive calls

SC: O(n)



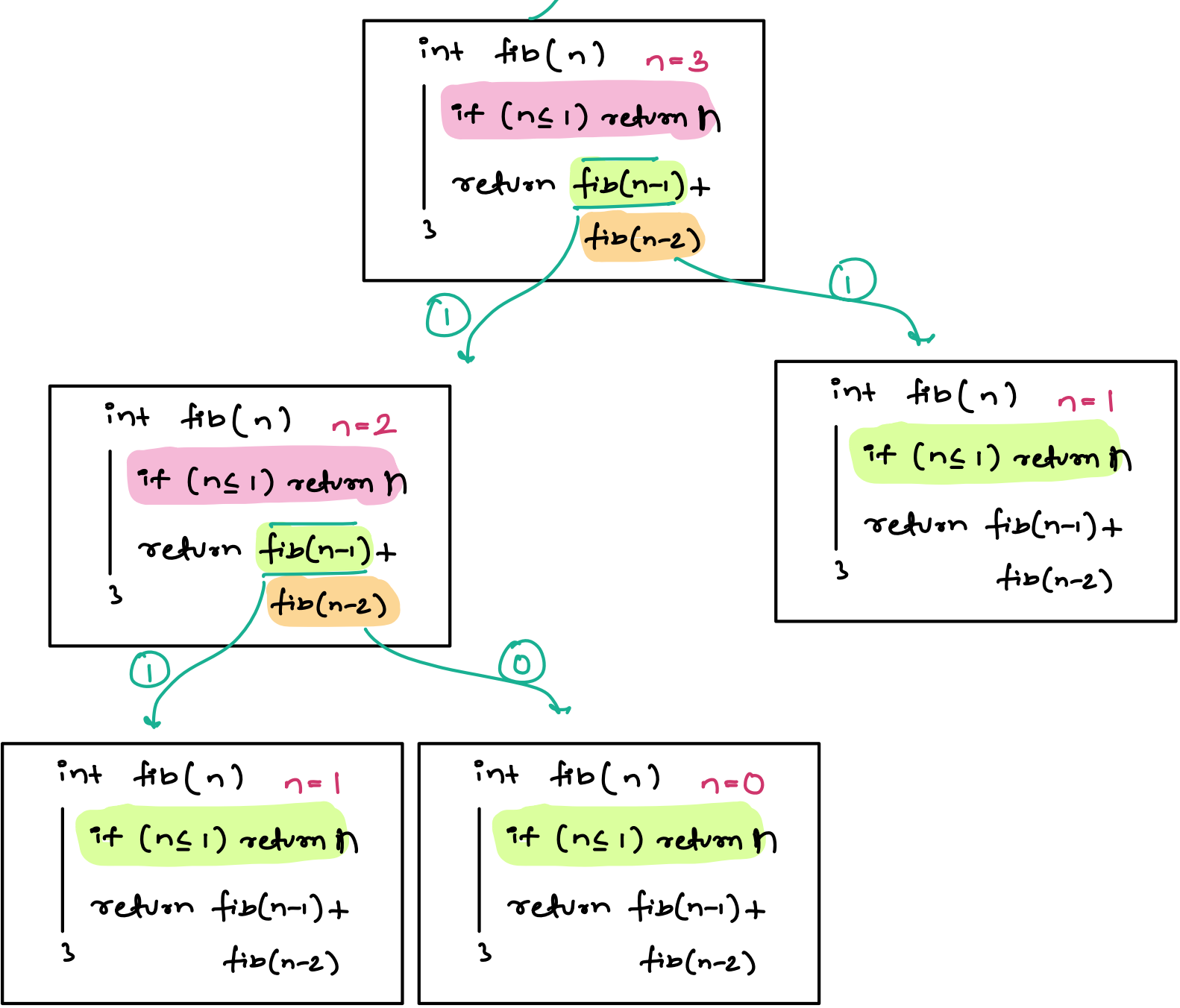
* Fibonacii series $n \geq 0$

fib(n) = sum of previous two fib no.

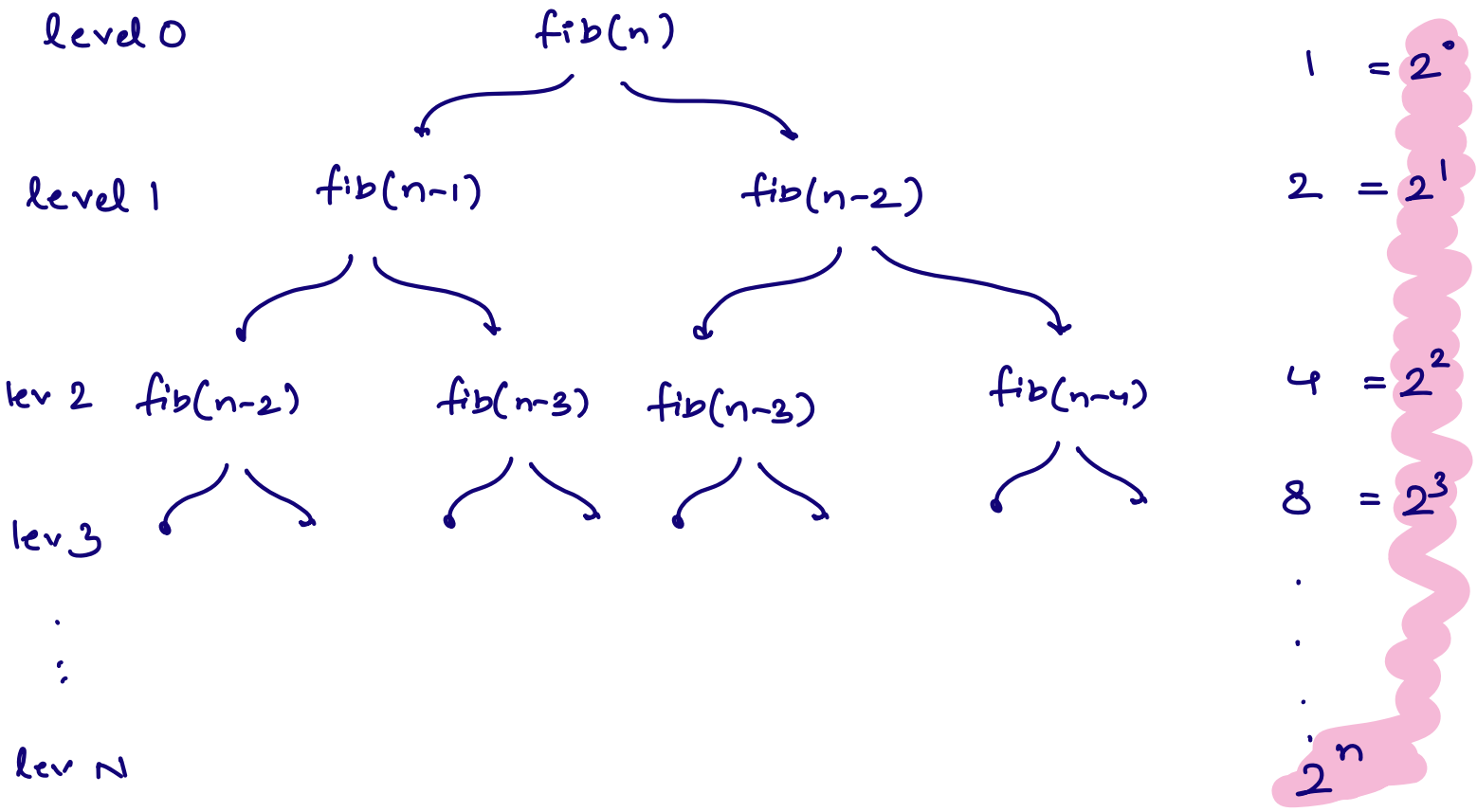
input	0	1	2	3	4	5
Output	0	1	1	2	3	5

$$fib(n) = fib(n-1) + fib(n-2)$$

```
int fib(n){
  if (n<=1) return n;
  return fib(n-1) + fib(n-2);
}
```


int fib(n) n=3
if (n≤1) return n
return fib(n-1)+
fib(n-2)
3
1
1
int fib(n) n=2
if (n≤1) return n
return fib(n-1)+
fib(n-2)
3
1
0
int fib(n) n=1
if (n≤1) return n
return fib(n-1)+
fib(n-2)
3
int fib(n) n=1
if (n≤1) return n
return fib(n-1)+
fib(n-2)
3
int fib(n) n=0
if (n≤1) return n
return fib(n-1)+
fib(n-2)
3

Time complexity



Total fn calls = $2^0 + 2^1 + 2^2 + \cdots + 2^n$

$$\text{Sum} = \frac{a(r^n - 1)}{r - 1} = \frac{1 * (2^{n+1} - 1)}{2 - 1}$$

$$\text{sum} = 2^{n+1} - 1$$

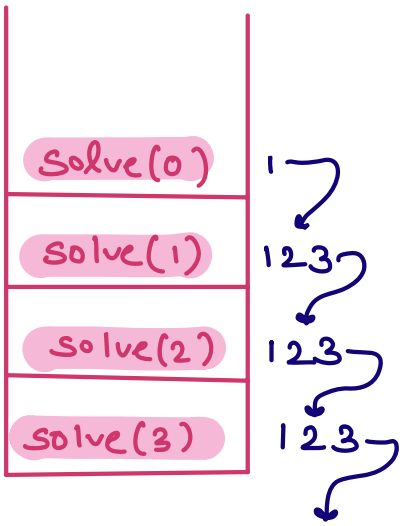
TC $\simeq O(2^n)$

SC $= O(n)$

* Output based quizzes

```
void solve (n)        <- 3
1 | if (n==0) return
2 | solve (n-1)
3 | print (n)
  }
```

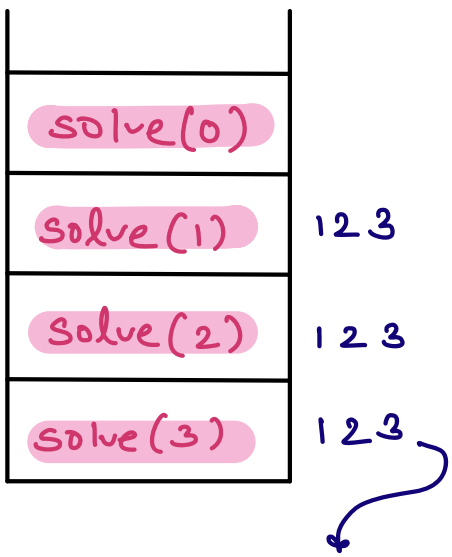
Ans = 1 2 3



02.

```
void solve (n)        <- 3
1 | if (n==0) return
2 | print (n)
3 | solve (n-1)
  }
```

Ans 3 2 1



03.
```
void solve (n)        // -3
{
    if (n==0) return
    print (n)
    solve (n-1)
}
```

⇒ Error, stack overflow

we will never reach zero

04.
```
int solve (n)         // TODO
{
    if (n==0) return 0

    return (solve(n/10) + n%10);
}
```

10:03 → 10:13 pm

Q Print all valid paranthesis of length 2N for a given value N

→ Equal no. of opening & closing brackets

N = 1 ⟶)(( )

N = 2 ⟶ (( )) ((())(() ( )( )

N = 3 ⟶ ((( ))) (( ))( ) ( )(( )) ( )( )( )

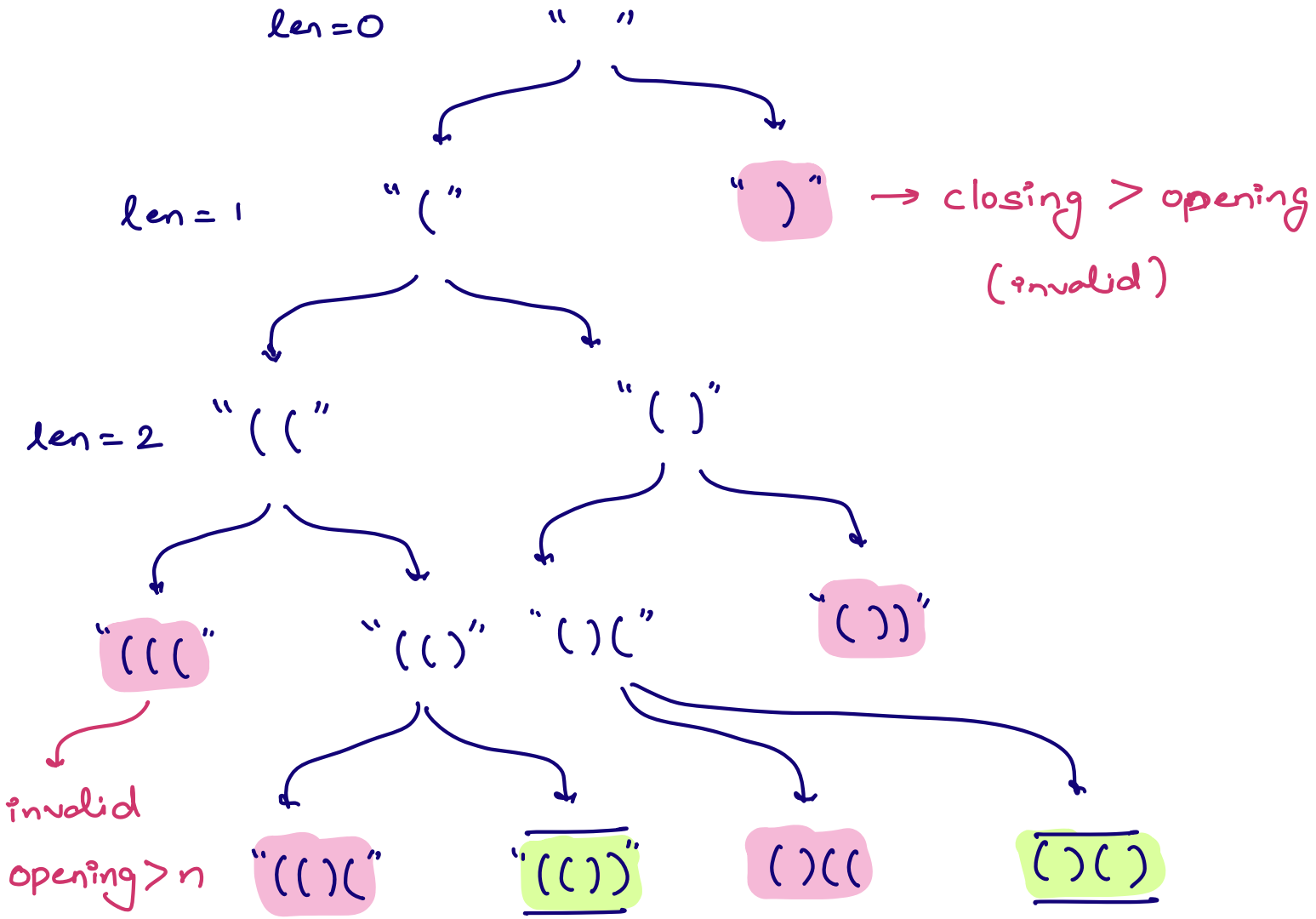
(( ) ( ))

* Idea → start with an empty string & explore each & every option

* Observation

01. The opening bracket must be greater than closing bracked

N = 2 → len = 4




```
void solve( N , str, opening, closing )
|   if (str.length() == 2N){
|   |   print(str); return;
|   3
|   if (opening < N)
|   |   solve(N, str+'(', opening+1, closing)
|   3
|   if (closing < opening){
|   |   solve(N, str+')', opening, closing+1)
|   3
3
```

TC : $O(2^n)$

SC : $O(2n) \approx O(n)$

solve (2, " ", 0, 0)

Make the rest of tree by yourself

solve (2, (, 1, 0)

solve (2, ((, 2, 0)

solve (2, ( ), 1, 1)

solve (2, (( ), 2, 1)

solve (2, (( )), 2, 2)

Tower Of Hanoi

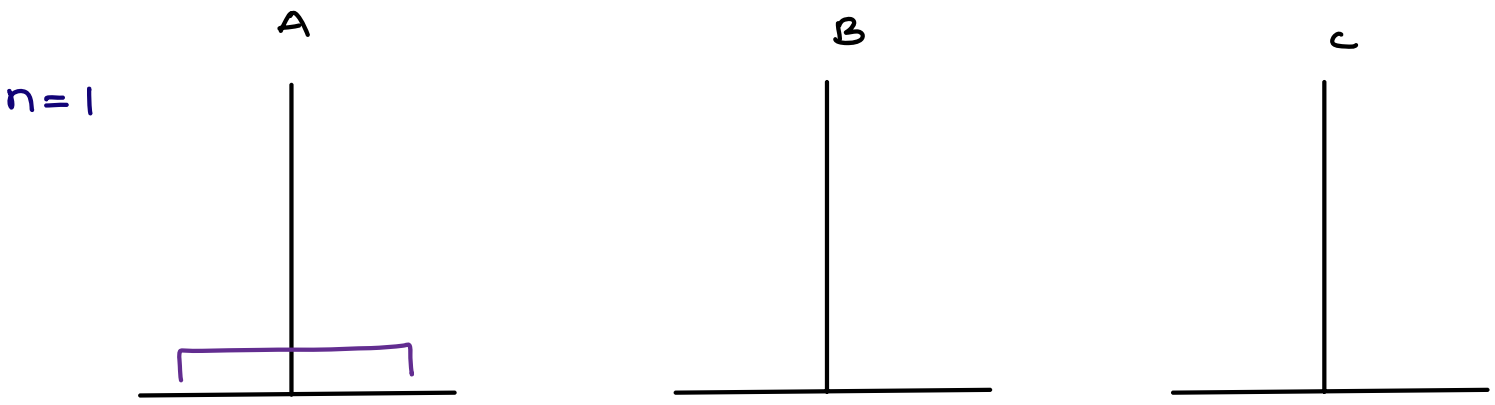
There are n disks placed on tower A of different sizes

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

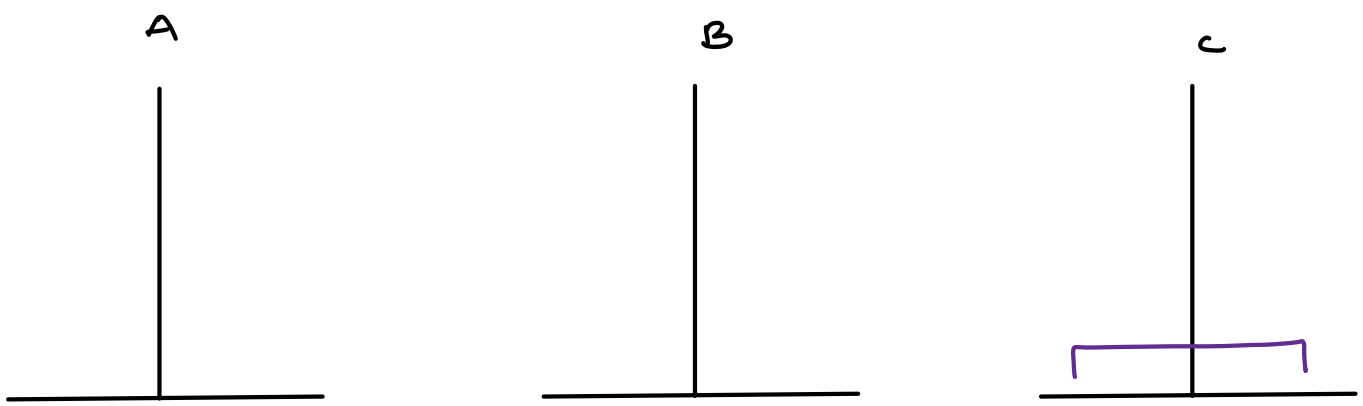
Constraint →

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

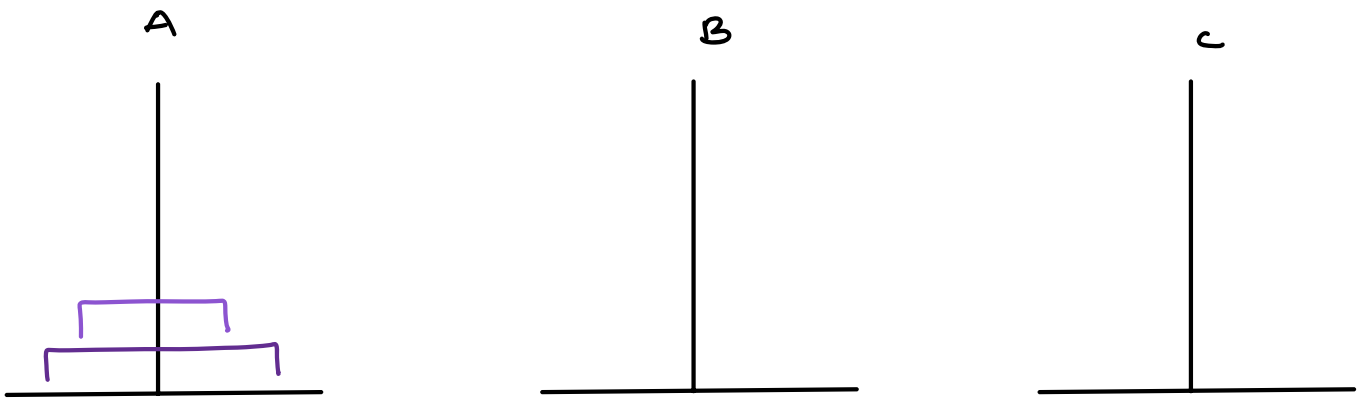
Print the movement of discs from A to C in **minimum steps**



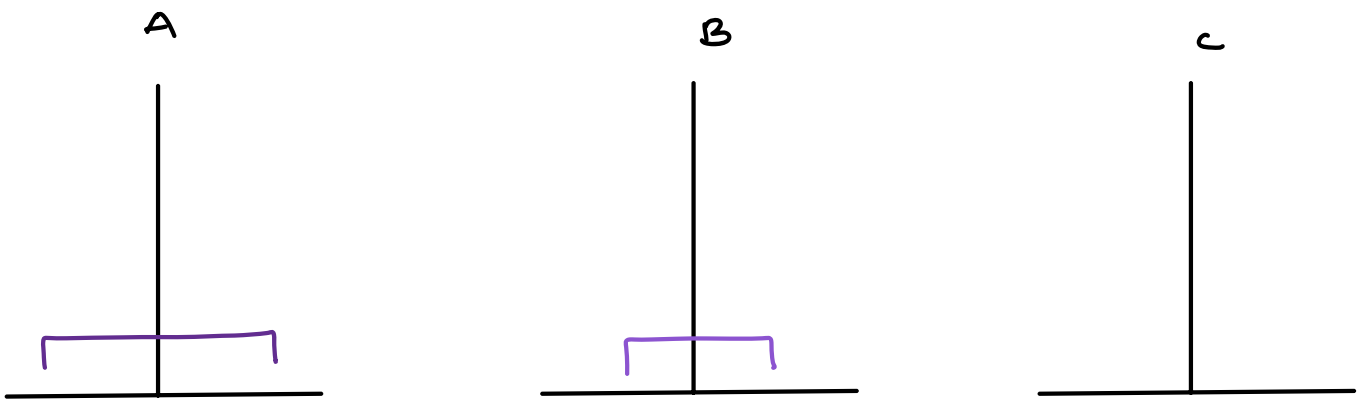
Output 1 : A → C



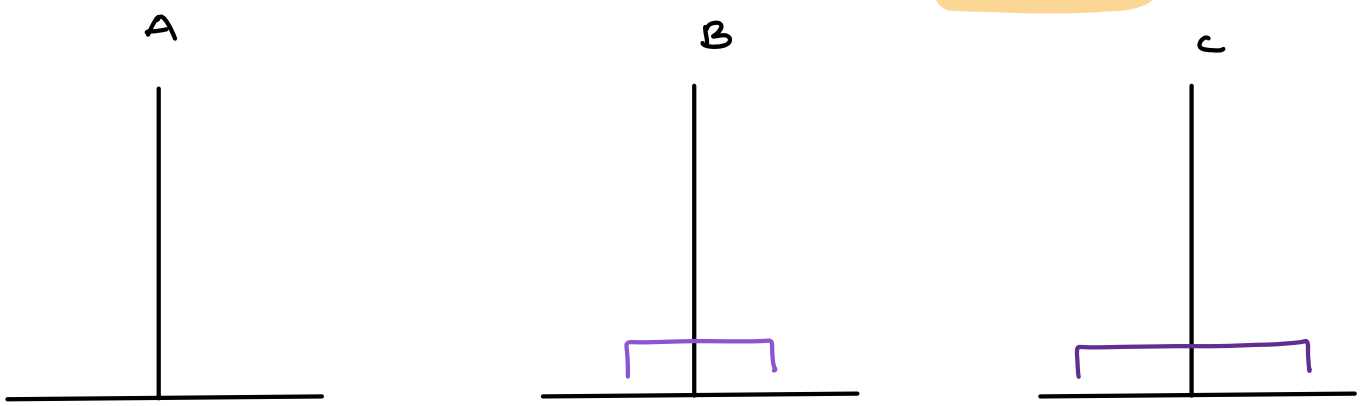
n=2



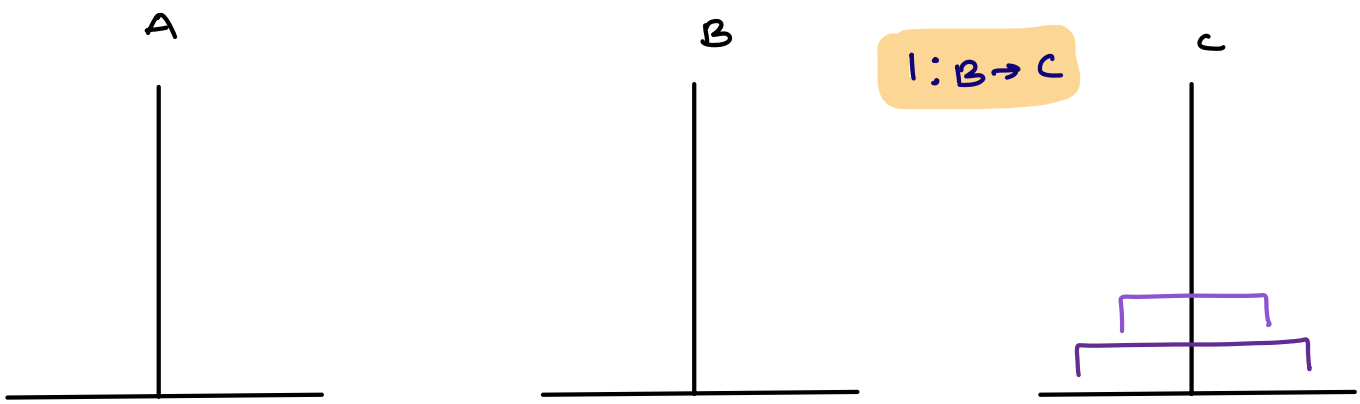
1 : A → B



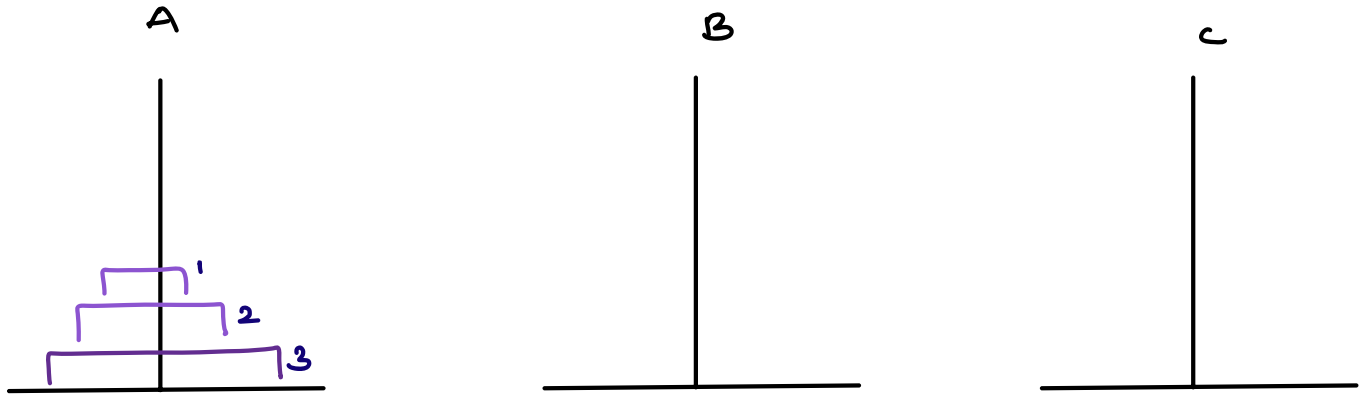
2 : A → C



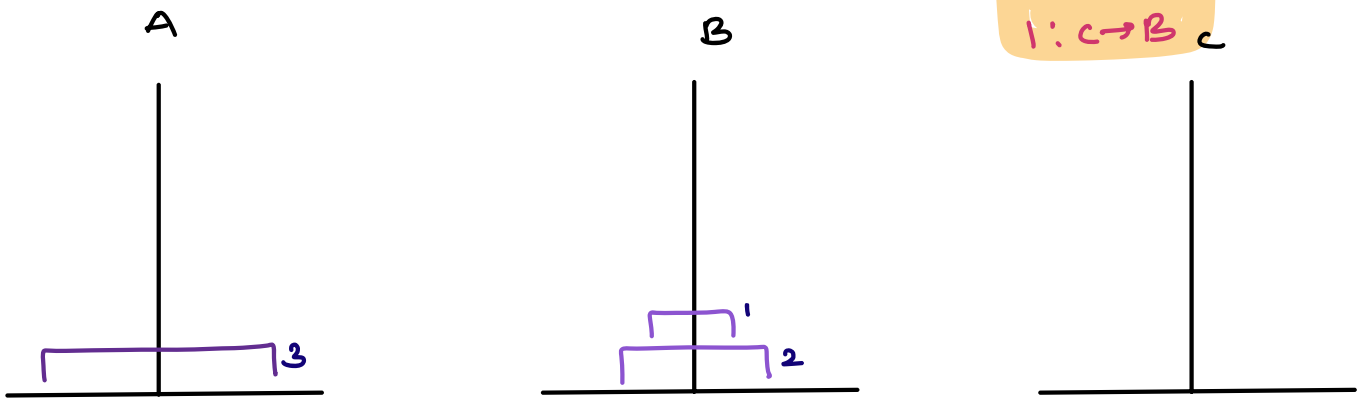
1 : B → C



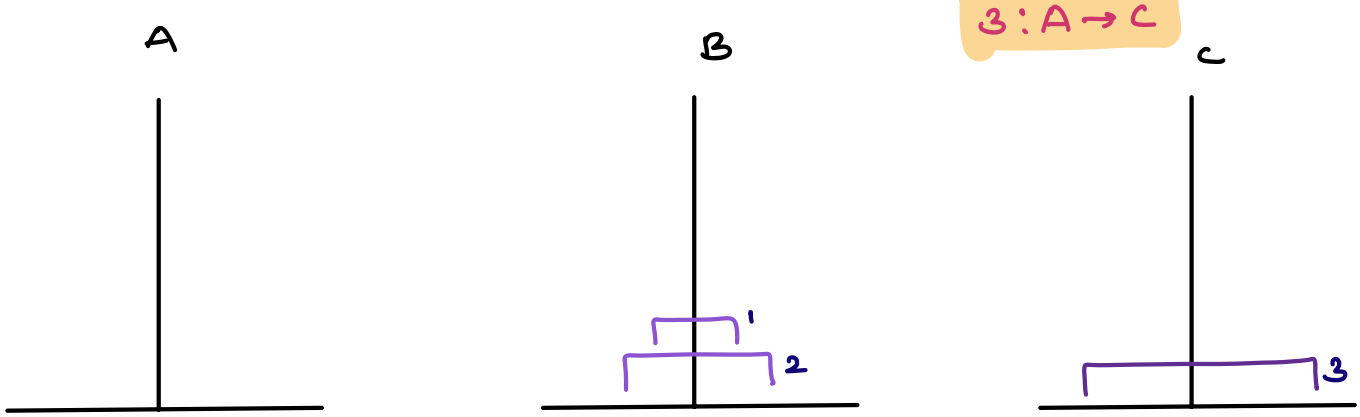
n = 3



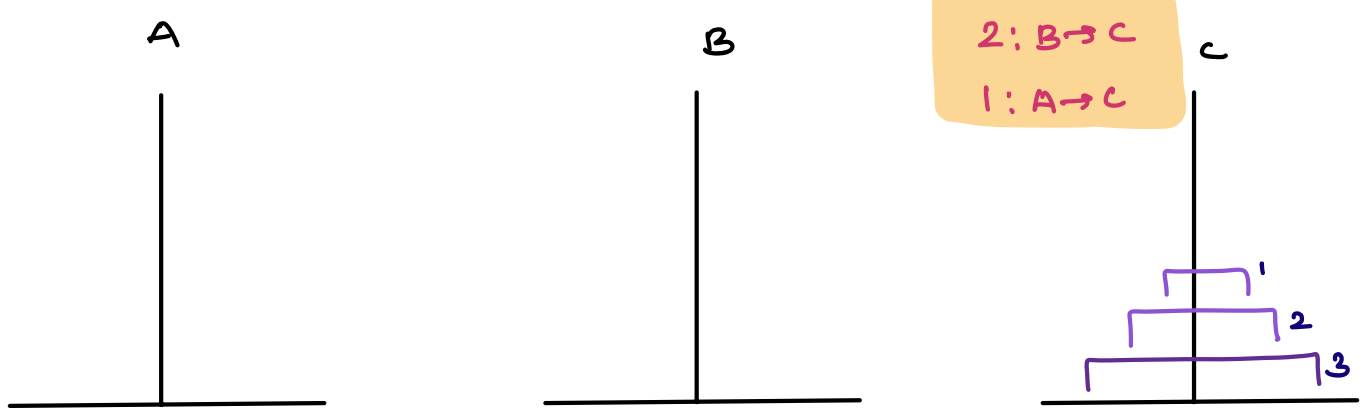
1 : A → C
2 : A → B
1 : C → B



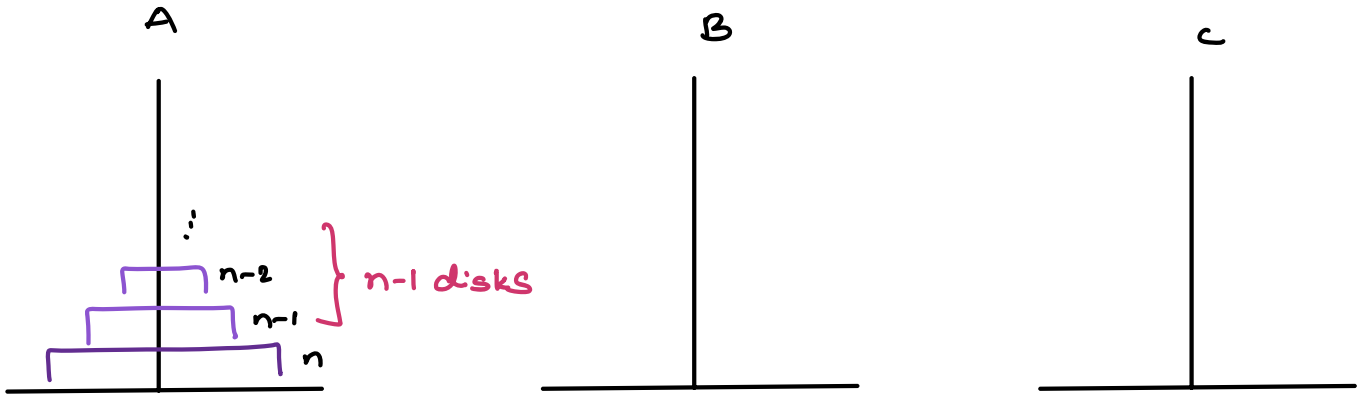
3 : A → C



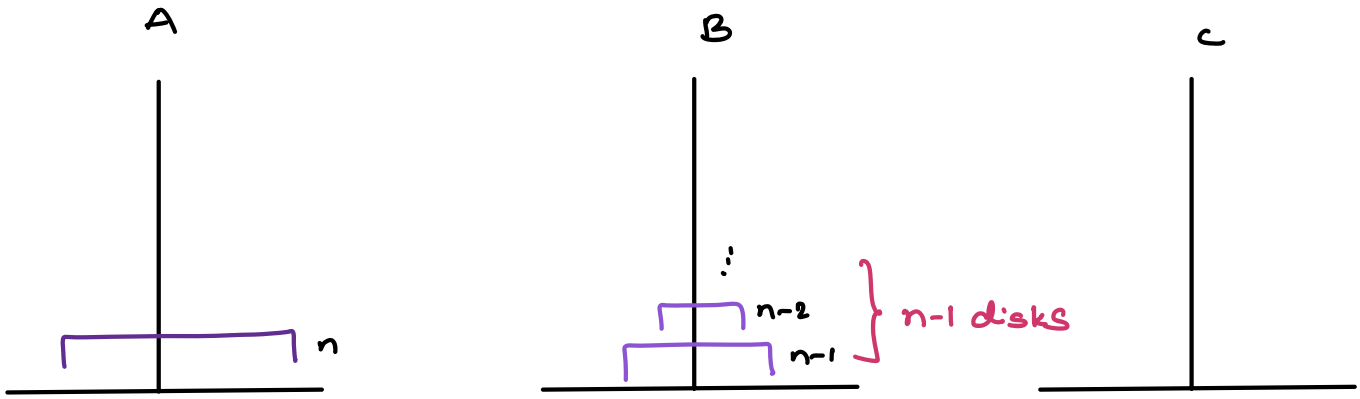
1 : B → A
2 : B → C
1 : A → C



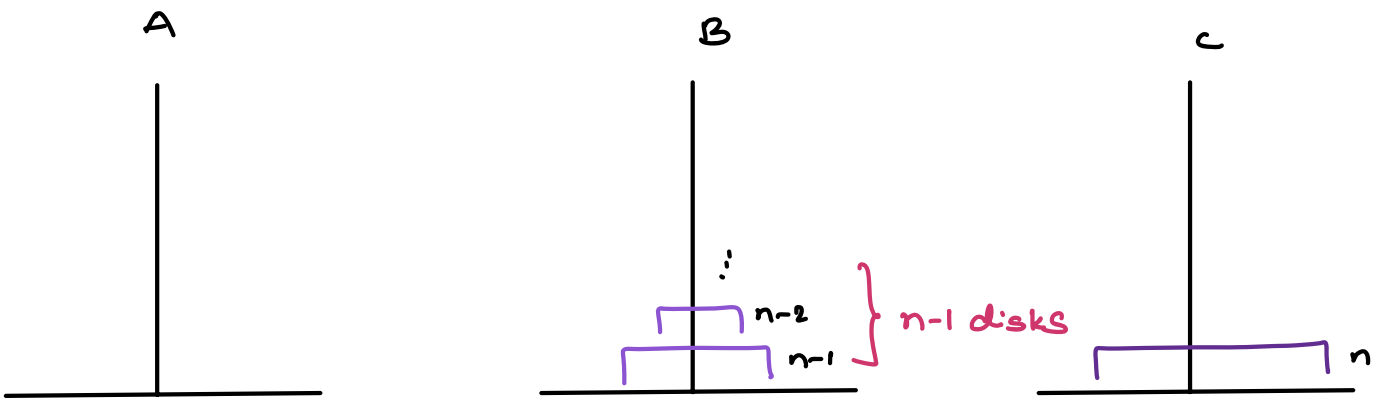
n discs



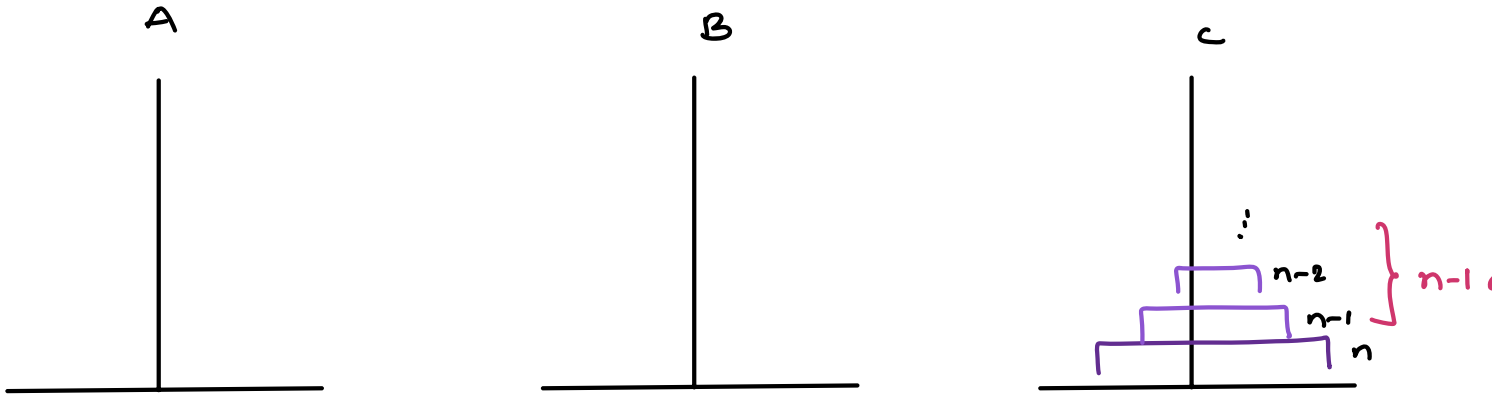
move (n-1) disks from A to B



move n^{th} disk to C

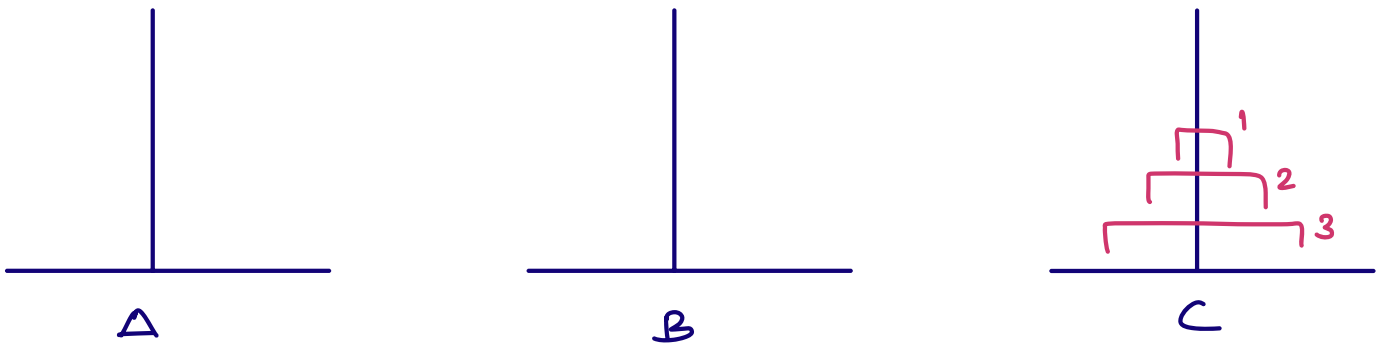


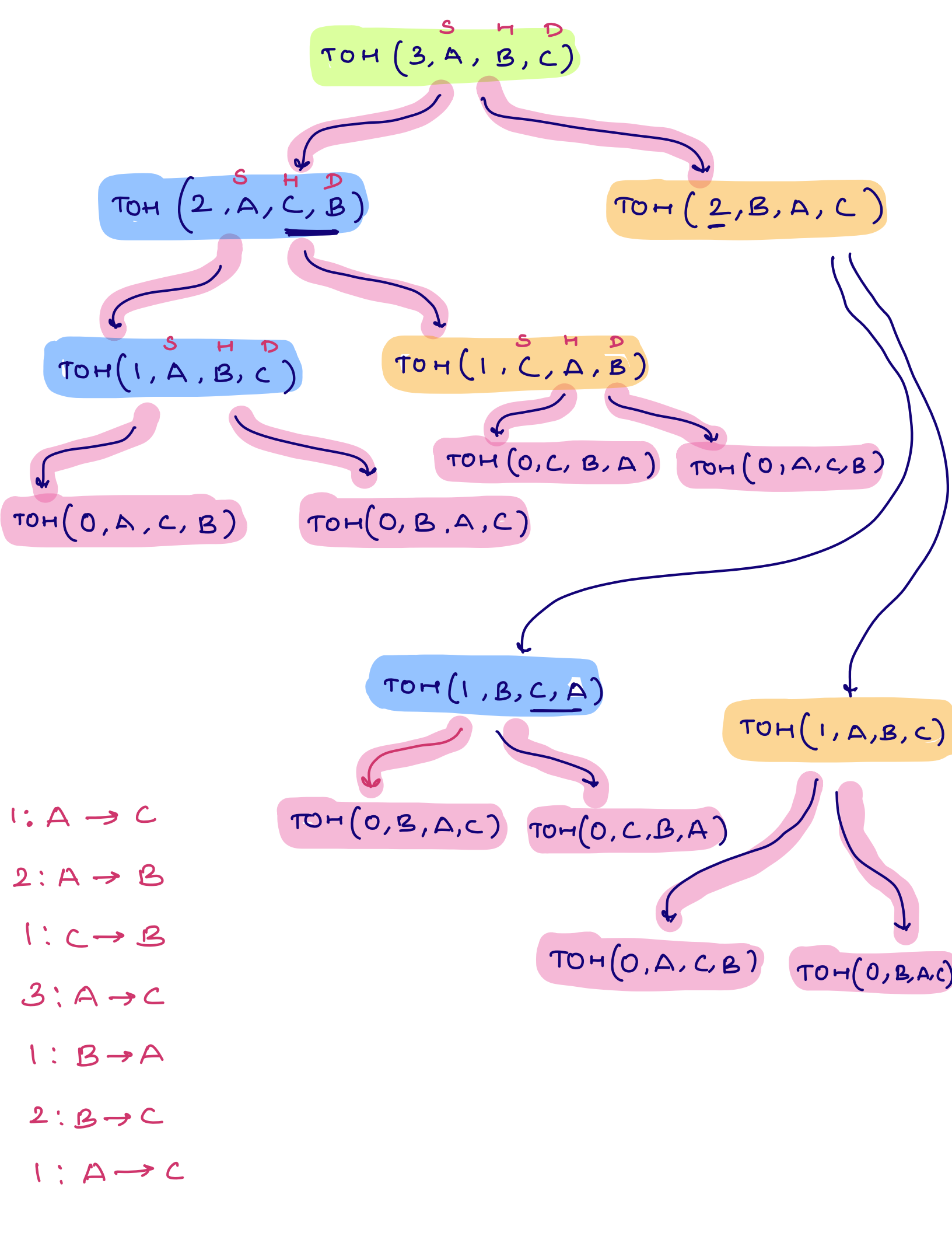
move n disks from B to C



src help dest

```
void TOH(int n, A, B, C)
{
    if (n==0) return;
    TOH(n-1, A, C, B)   // move n-1 disks A→B
    print(n : A ⟶ C)    // moving nth disk A→C
    TOH(n-1, B, A, C)   // move n-1 disks B→C
}
```




S H D
TOH (3, A, B, C)
S H D
TOH (2, A, C, B)
TOH (2, B, A, C)
S H D
TOH(1, A, B, C)
S H D
TOH(1, C, A, B)
TOH(0, A, C, B)
TOH(0, B, A, C)
TOH(0, C, B, A)
TOH(0, A, C, B)
TOH(1, B, C, A)
TOH(1, A, B, C)
TOH(0, B, A, C)
TOH(0, C, B, A)
TOH(0, A, C, B)
TOH(0, B, A, C)
1: A → C
2: A → B
1: C → B
3: A → C
1: B → A
2: B → C
1: A → C

Time complexity $= 2^n$

$$f(n) = f(n-1) + 1 + f(n-1)$$

$$f(n) = 2f(n-1) + 1 \qquad f(0) = 1$$