Recursion 2

"MOTIVATION MAY BE WHAT STARTS YOU OFF, BUT IT'S HABIT THAT KEEPS YOU GOING BACK FOR MORE."
— MIYA YAMANOUCHI
STARTUPVITAMINS

Good

Morning

Content

01. Quizzes

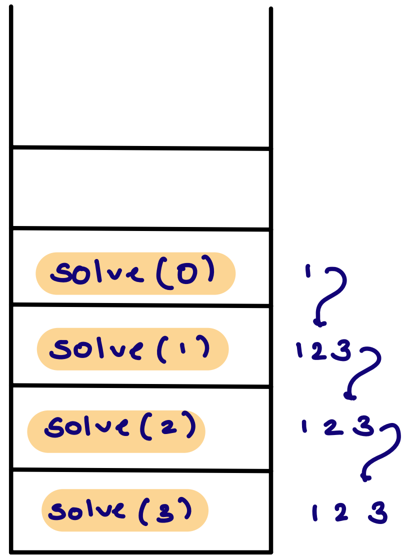
02. Tower of Hanoi Pure recursion

03. Generate paranthesis Backtracking

N = 3

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

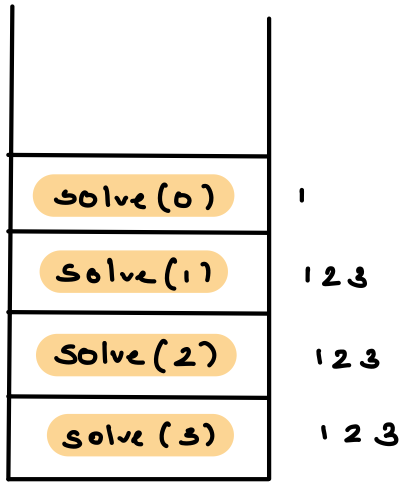
Ans = 1 2 3



3

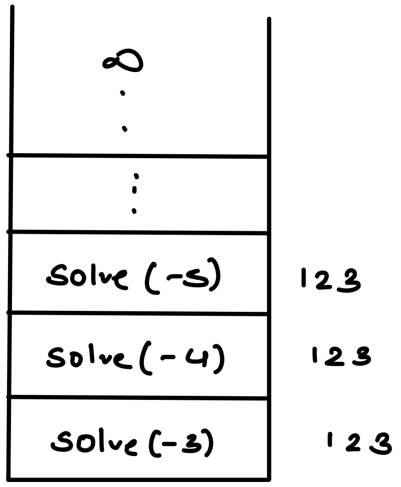
```
02. void solve (int N)
        if (N==0) return;       1
        print (N) ;             2
        solve (N-1);            3
    3
```

Ans = 3 2 1



- 3

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



Ans = -3 -4 -5 → Stack overflow error

≃ $10^5 - 10^6$ calls

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

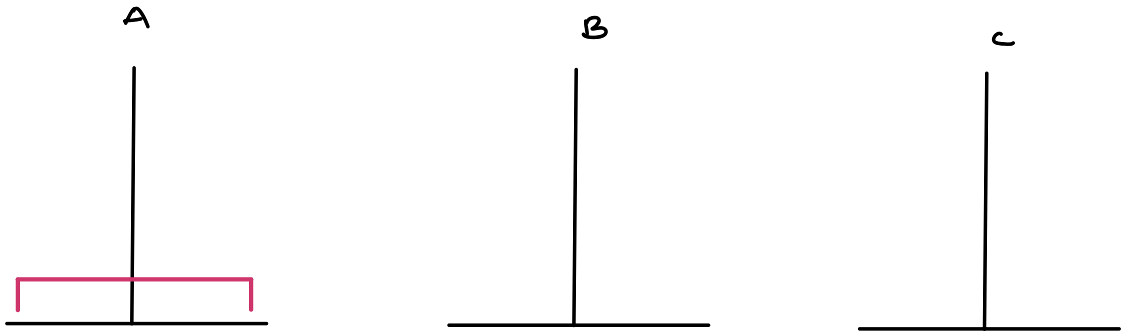
Constraint →

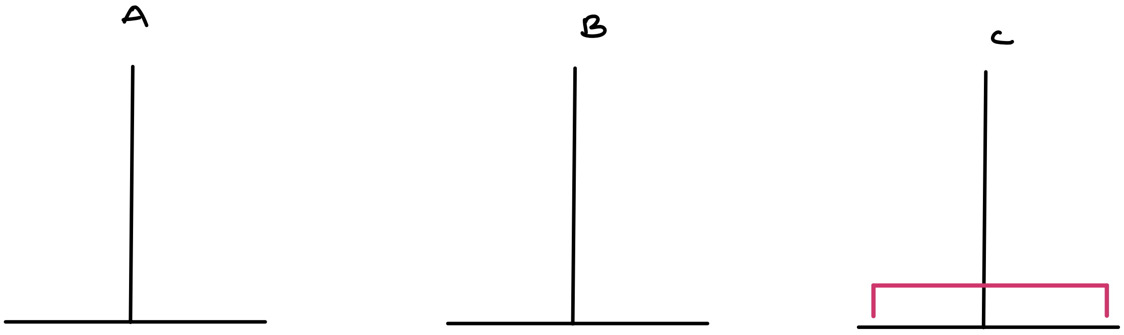
01. Only 1 disk can be moved at a time

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

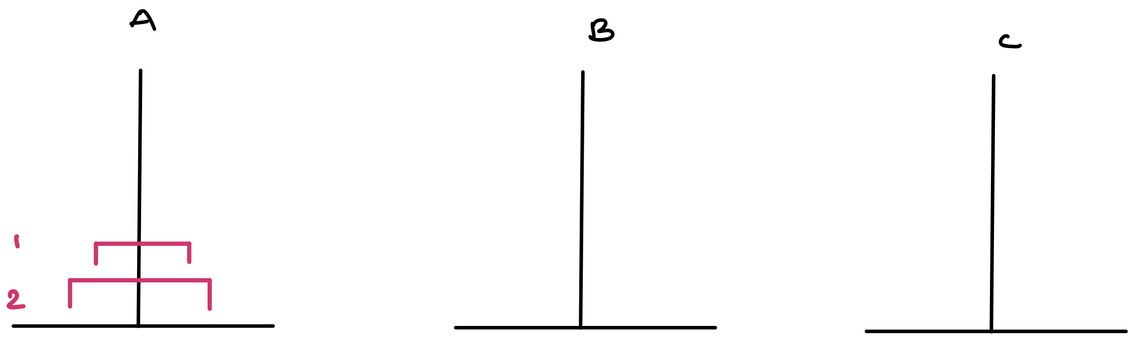
n=1



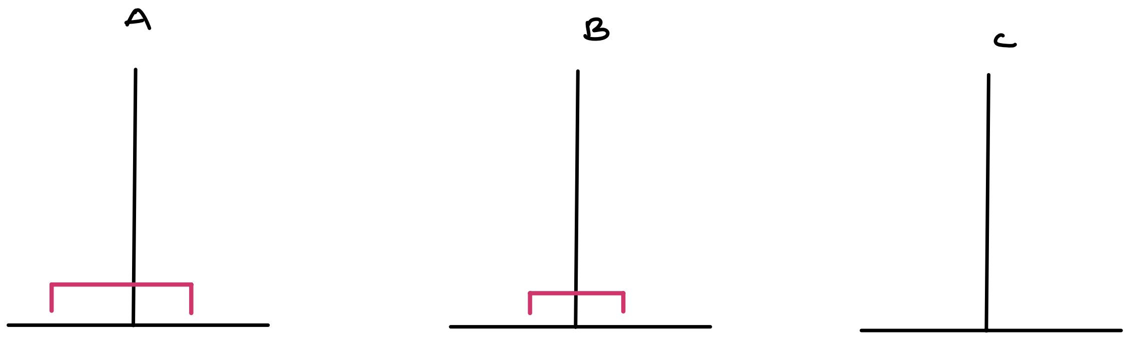


Output : 1 : A → C

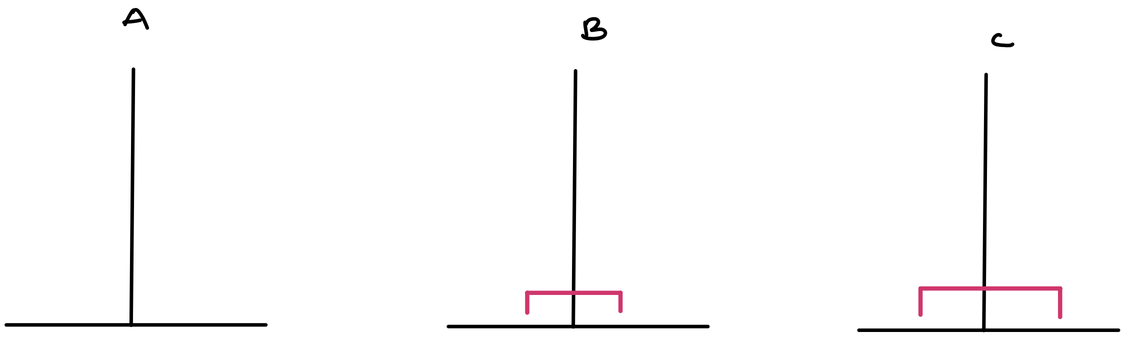
$n = 2$



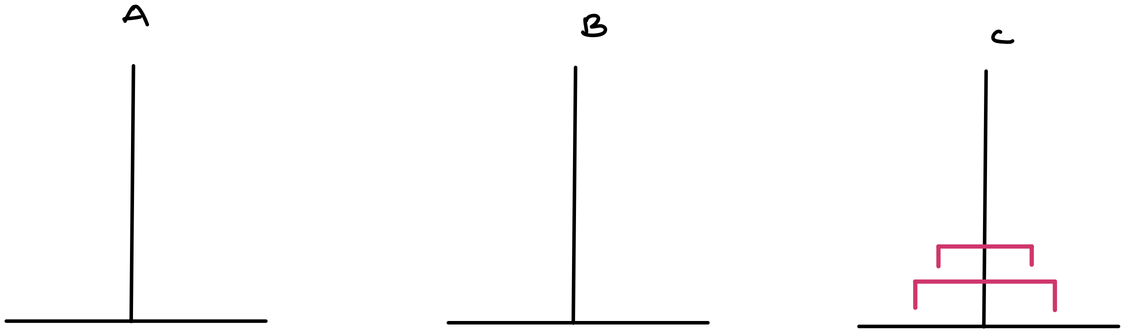
1 : A → B



2 : A → C



1 : B → C



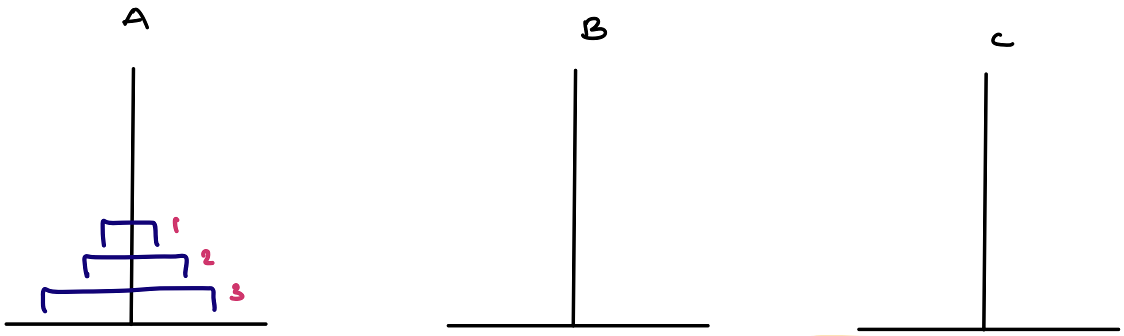
Out put :

1 : A → B

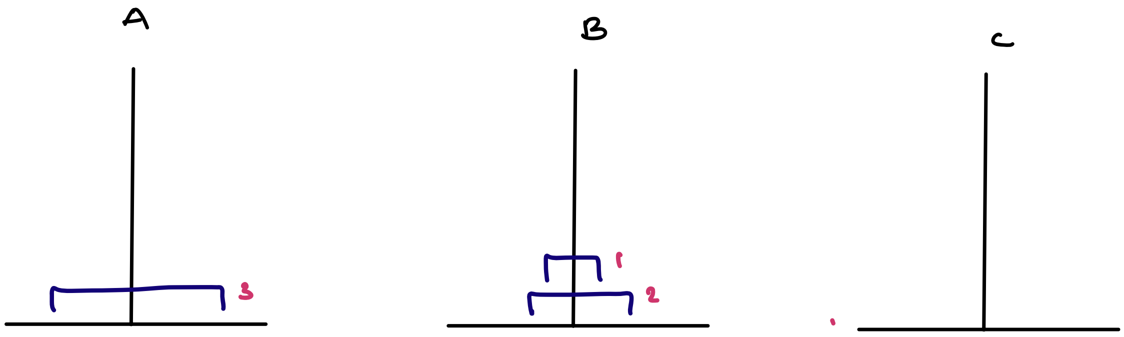
2 : A → C

1 : B → C

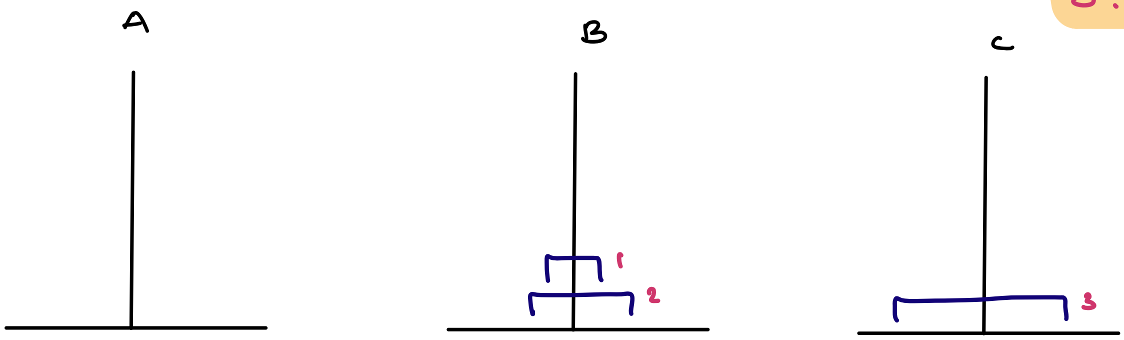
N = 3



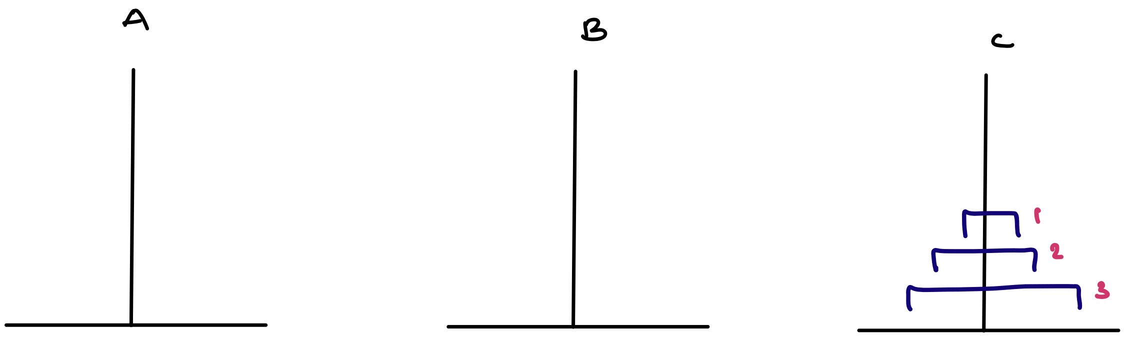
1 : A → C
2 : A → B
1 : C → B



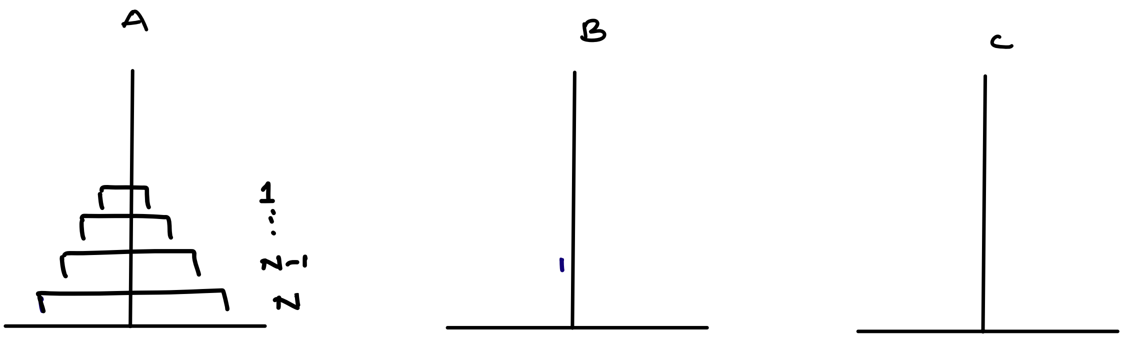
3 . A → C



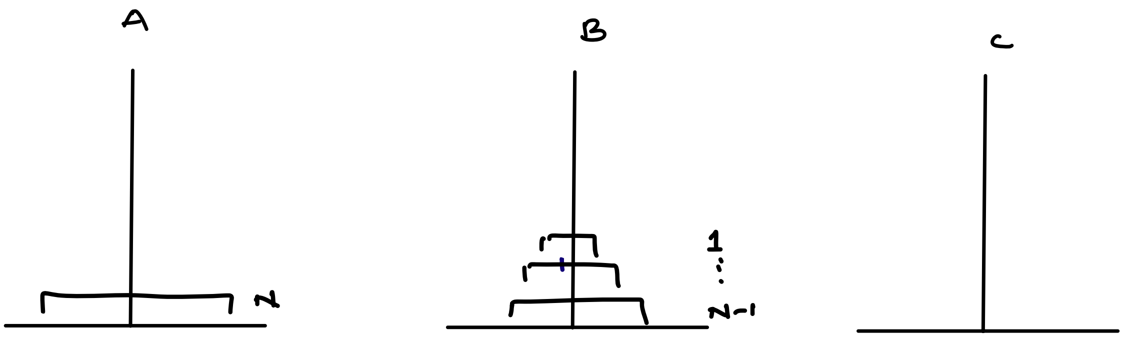
1 : B → A
2 : B → C
1 : A → C



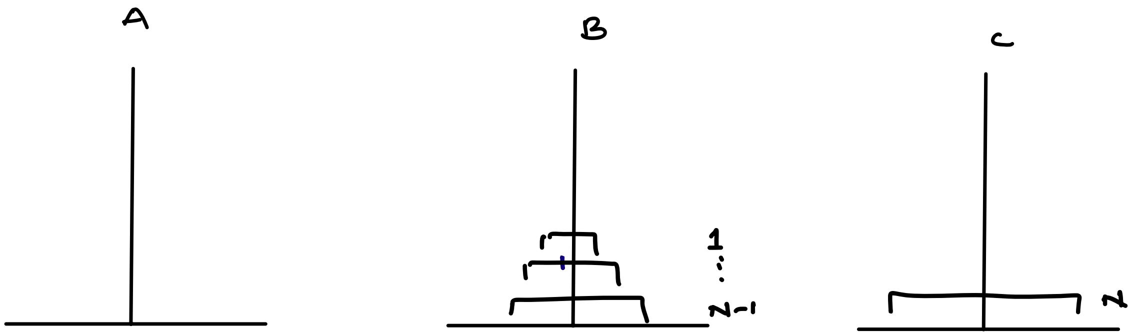
N disks



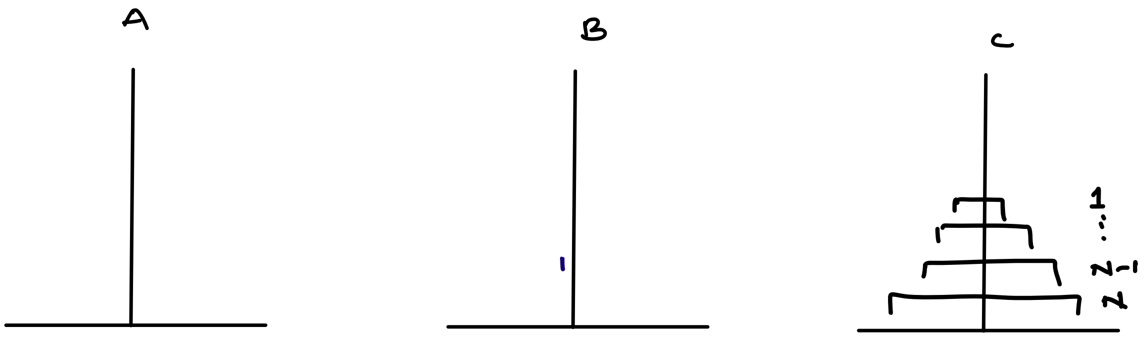
01. Move n-1 disks from A → B



02. Move N° disk from A → C



03. Move N-1 disks from B → C



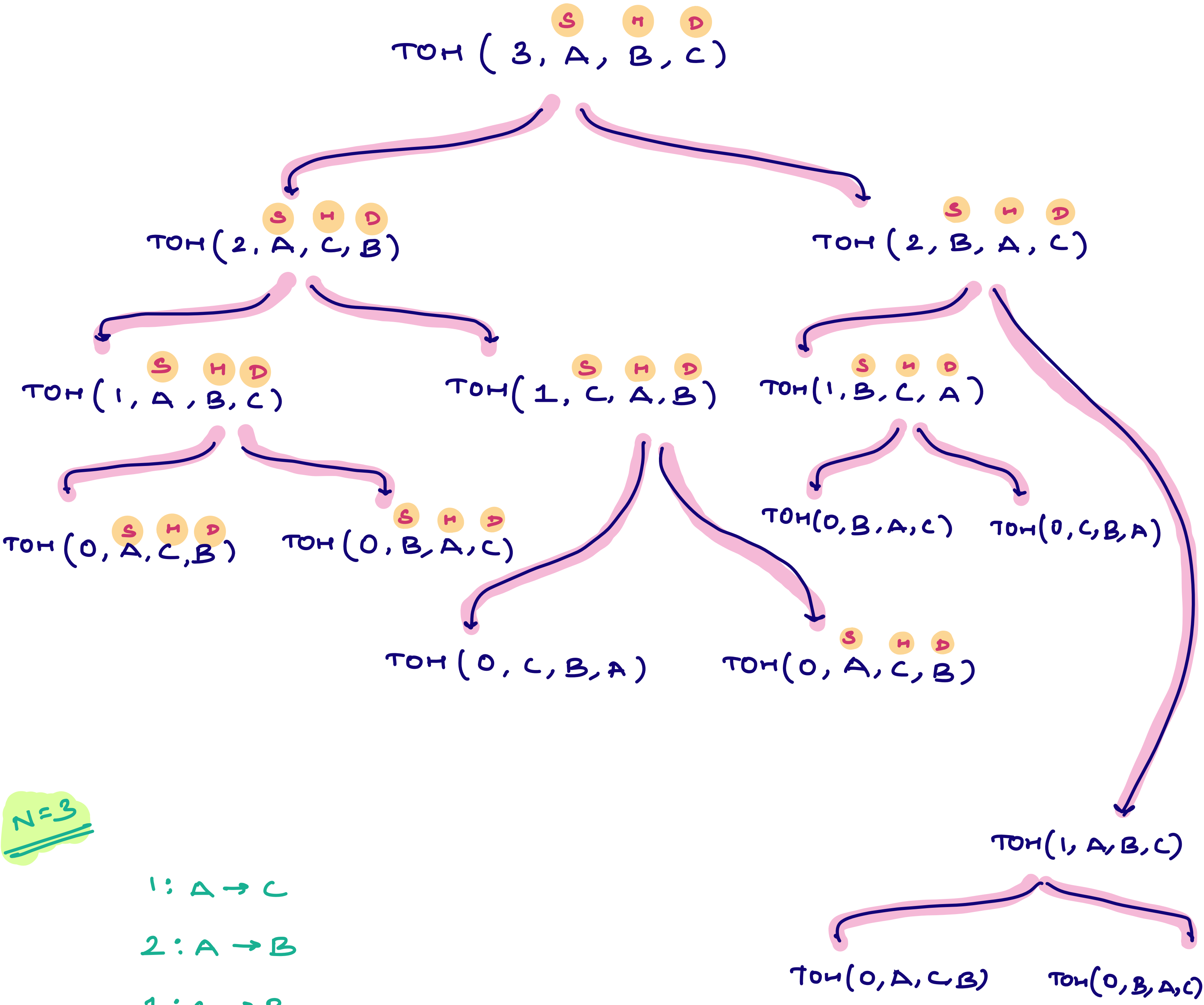
```
                         S     H     D
void TOH ( int n,        A ,   B ,   C )
    if (n== 0) return;

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

Faith ⇒ Recursion is going to move n-1 disks from src → dest with the helper tower in minimum steps, following all the constraints

Just to create our tree better

```
left call  = TOH (n-1, src, dest ⟷ helper)
right call = TOH (n-1, helper ⟷ src, dest)
```

```
print (n: src → dest)
```



N=3

1 : A → C

2 : A → B

1 : C → B

3 : A → C

1 : B → A

2 : B → C

1 : A → C

TC : $O(2^n)$

SC : $O(n)$

8:09 AM → 8:19 AM

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

→ Equal no. of opening & closing brackets

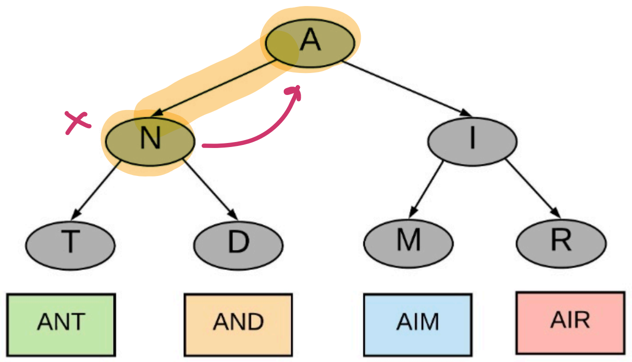
N = 1 ⟶ ( )) (

N = 2 ⟶ (( )) ( )( ) ( ))( ))(( )( )(

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

Idea → I'll build the string of length 2N, checking if it is valid or not while building it.

→ As soon as we get a state from which we can't get our valid answer, we will back track.

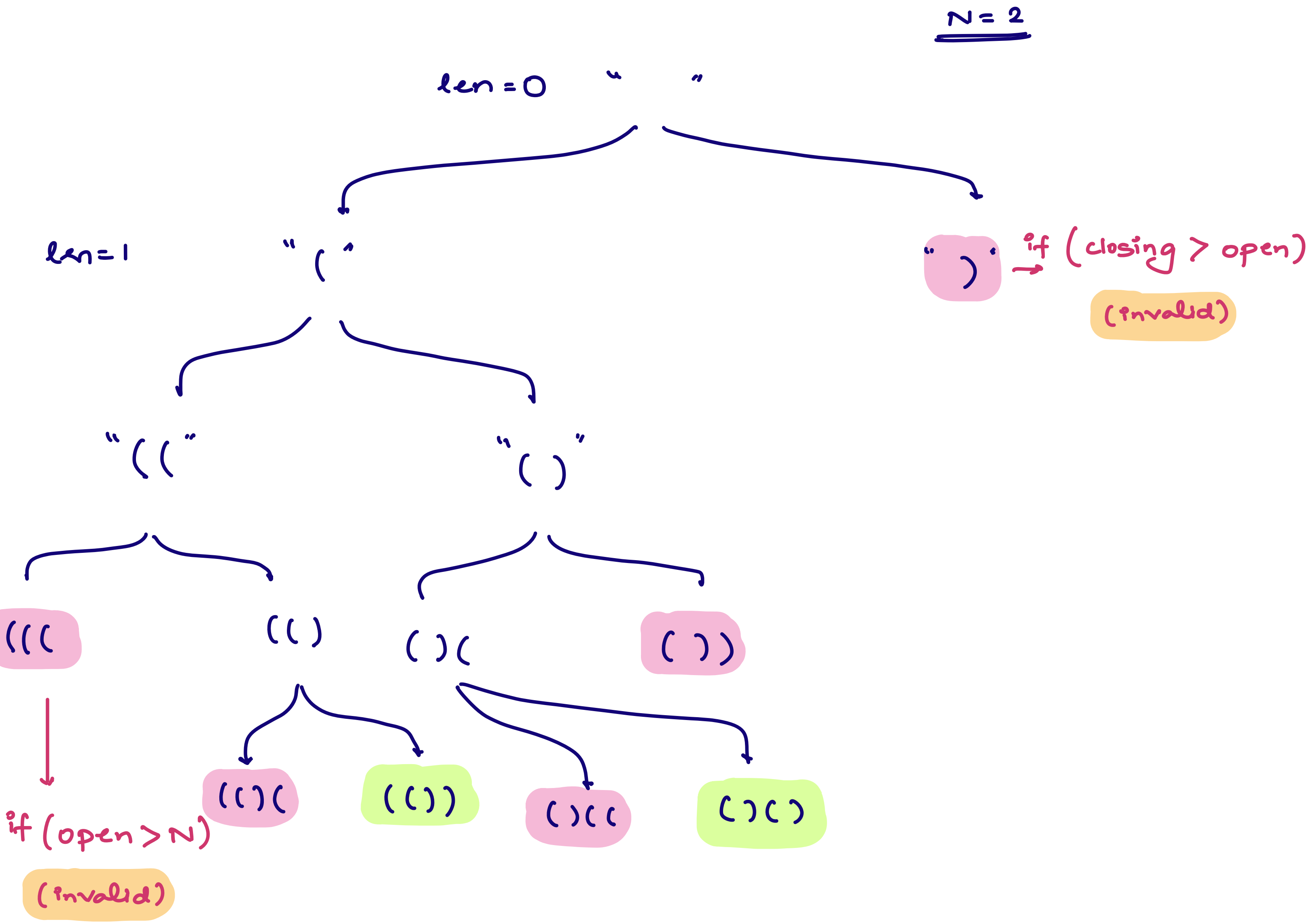


→ Words expressed as Branches

→ Find AIM

BF to find Aim ⇒ Generate the entire word on one branch & then compare it with the given word (i.e AZM)

Backtracking → As soon as you see an invalid character, you return back with exploring that branch further.



```
void main(){
    recur(" ", N, 0, 0);
}

void recur(String str, int N, int open, int close){
    if(str.length = = 2N){
        print(str);
        return;
    }
    if(open < N){
        recur(str+"(", N, open+1, close);
    }
    if(close < open){
        recur(str+")", N, open, close+1);
    }
}
```

TC : $O(2^n)$

SC : $O(n)$

```
fn (x, n)
|
|   if (n==0) return 1
|   else if (n % 2 == 0)   return   fn ____
|   els      ____
|
}
```

* Bits 1

target score = A

$\boxed{A = 3}$

Alex = Ø ~~1~~ ~~2~~ 1

Sam = S S

1

② times

A = 9

Alex = Ø ~~1~~ ~~2~~ ~~4~~ 8 9

Sam = S S

Ans = 2

Alex = ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~8~~ 9

Sam S S S S S

* <u>check palindrome</u>

$$\underbrace{1 \quad 2 \quad 1 \quad 2 \quad 5 \quad 6}$$

$$5 \wedge 6 =$$

$$\begin{array}{rl} & \quad\quad\ \downarrow \\ 5 = & 1\ 0\ 1 \\ 6 = & 1\ 1\ 0 \\ \hline & 0\ 1\ \boxed{1} \end{array}$$