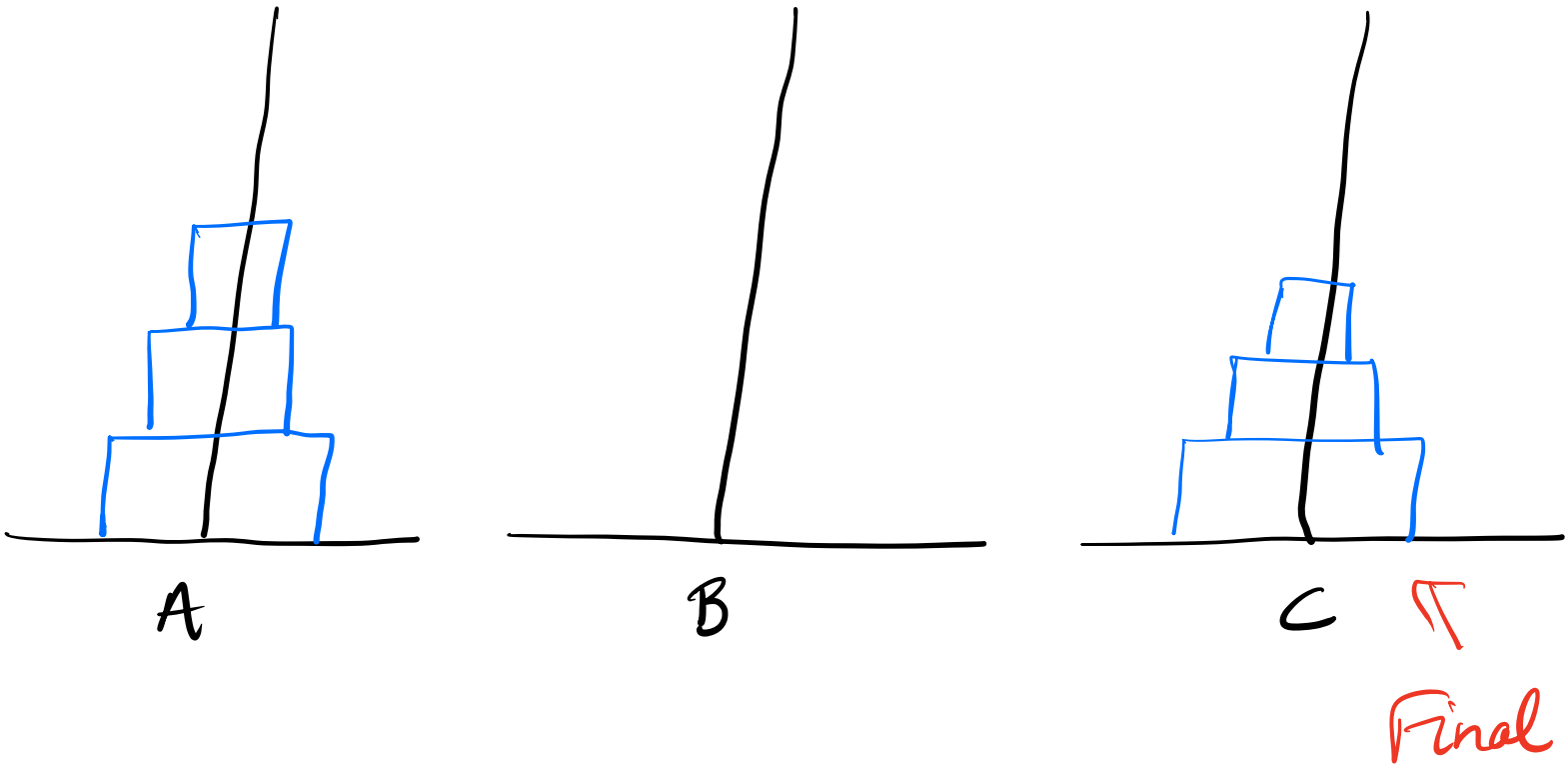
- Tower of Hanoi Given 3 towers A, B, C

  N discs are on Tower A

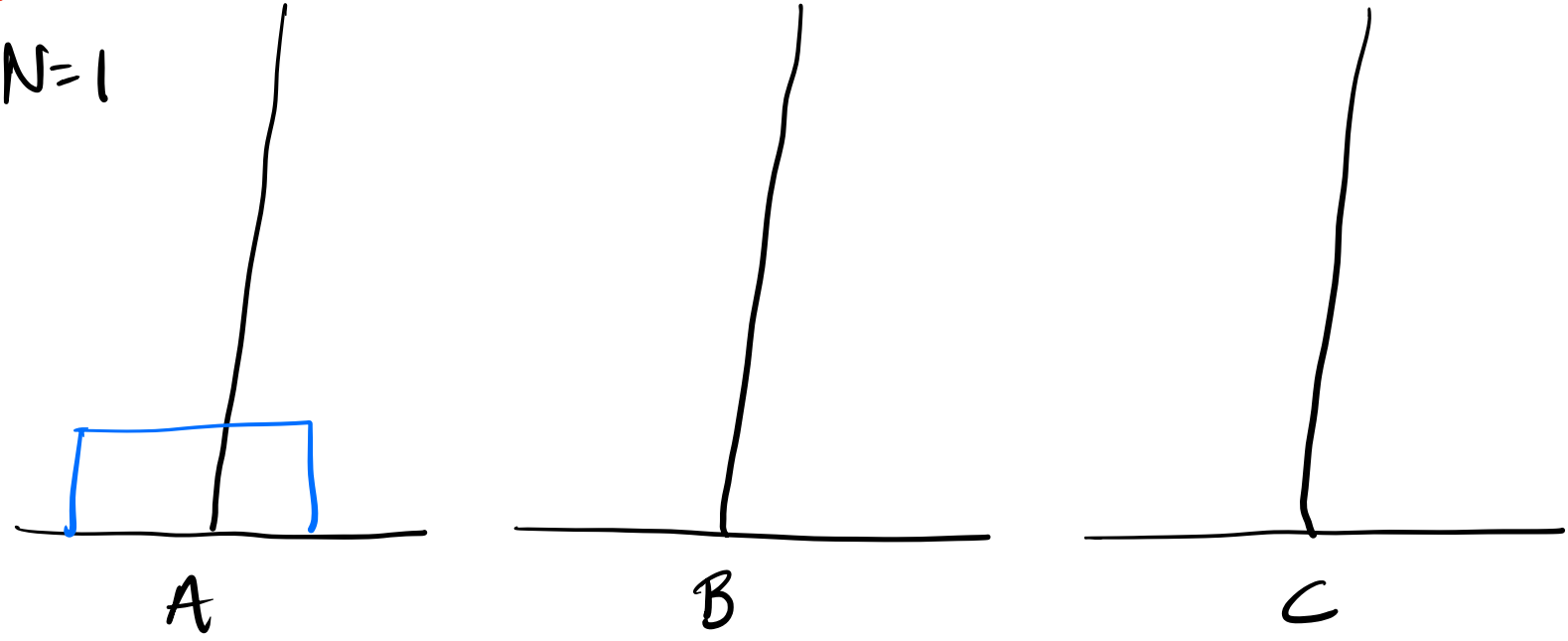
  move all discs from A → C

  - Only 1 disc moves at a time.
  - Larger disc cant be placed on smaller disc.

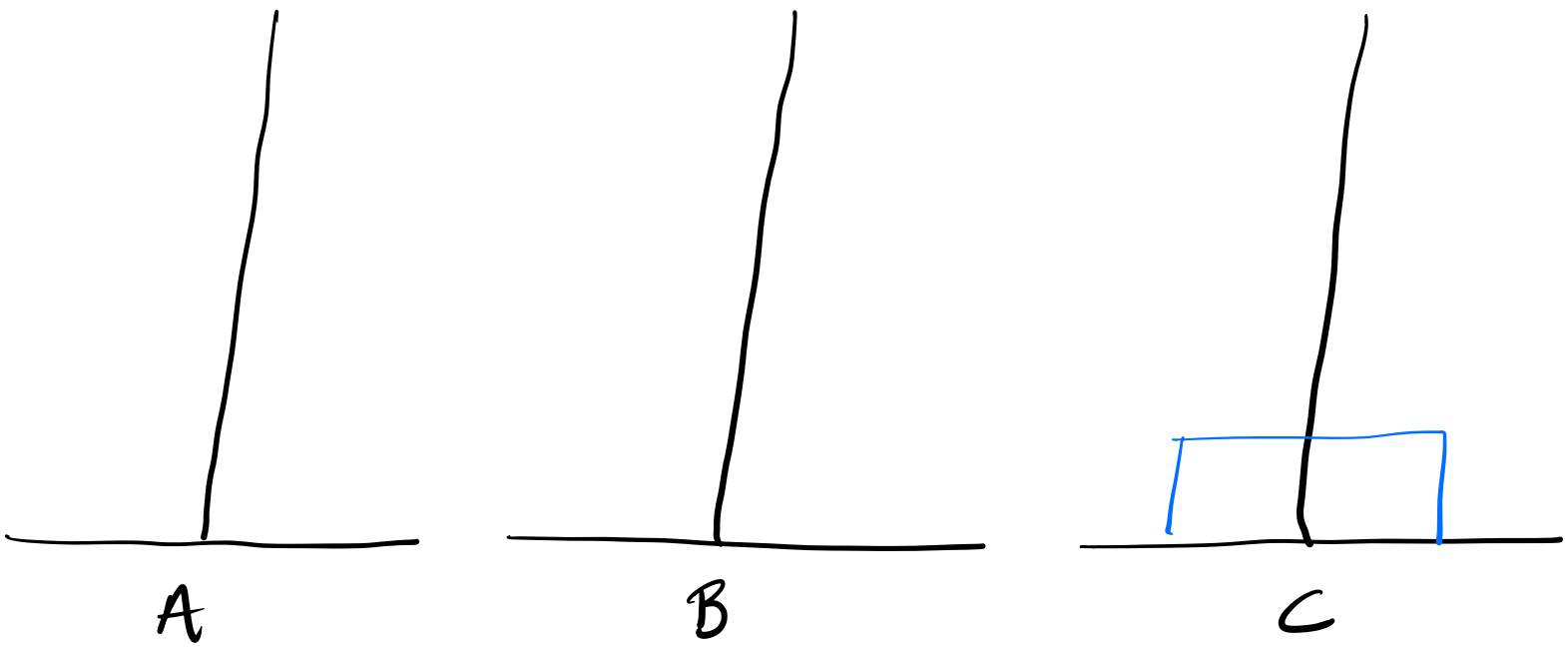


Print the movement of discs

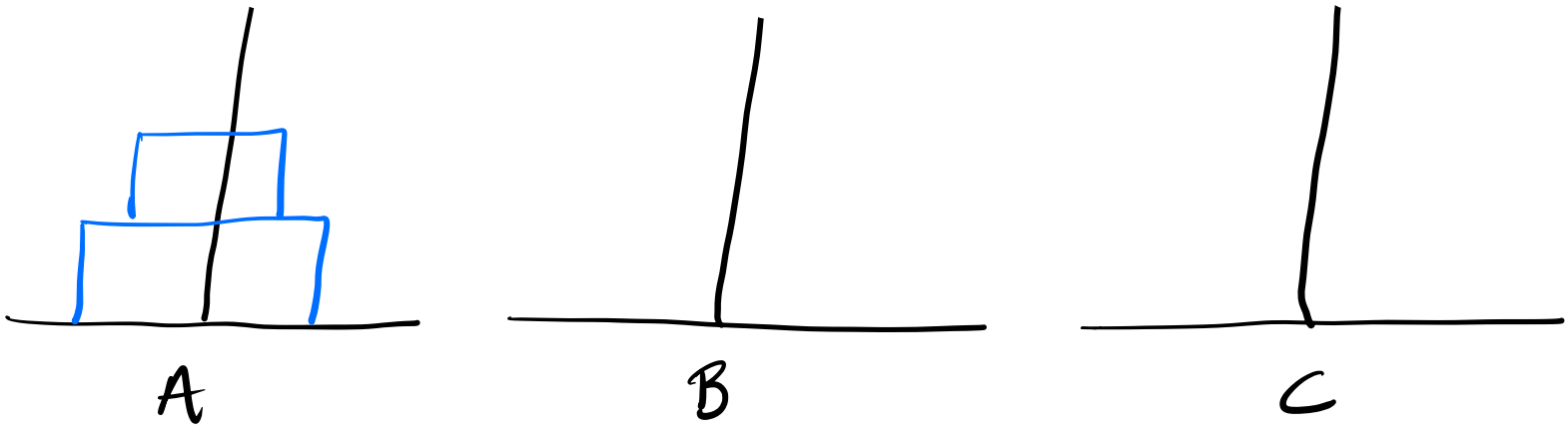
N=1



A → C

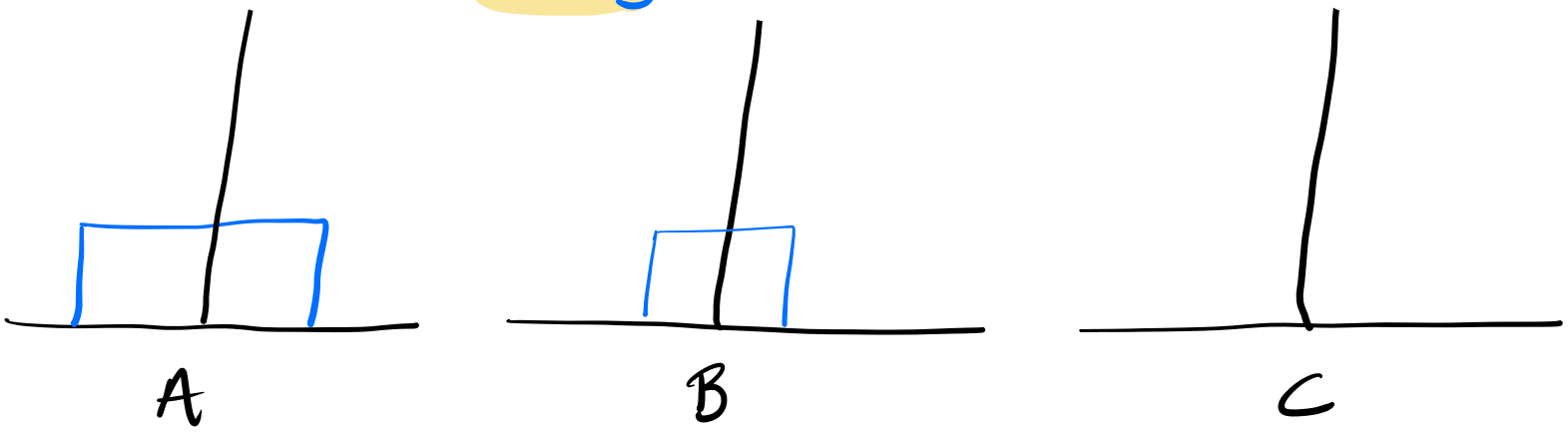


N=2

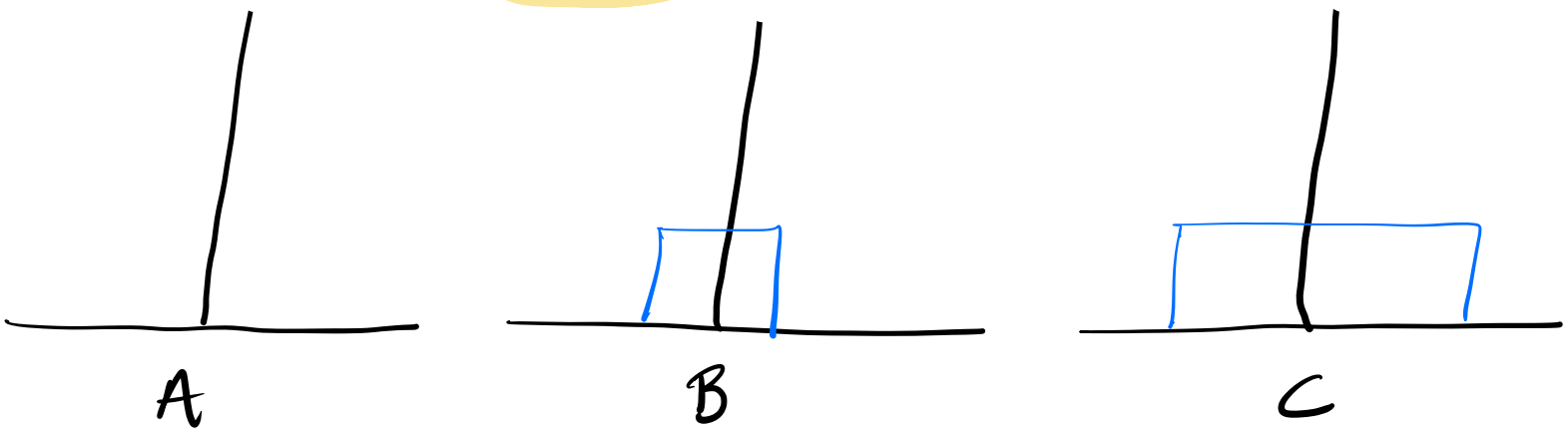


src temp dest

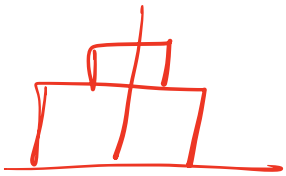
A → B

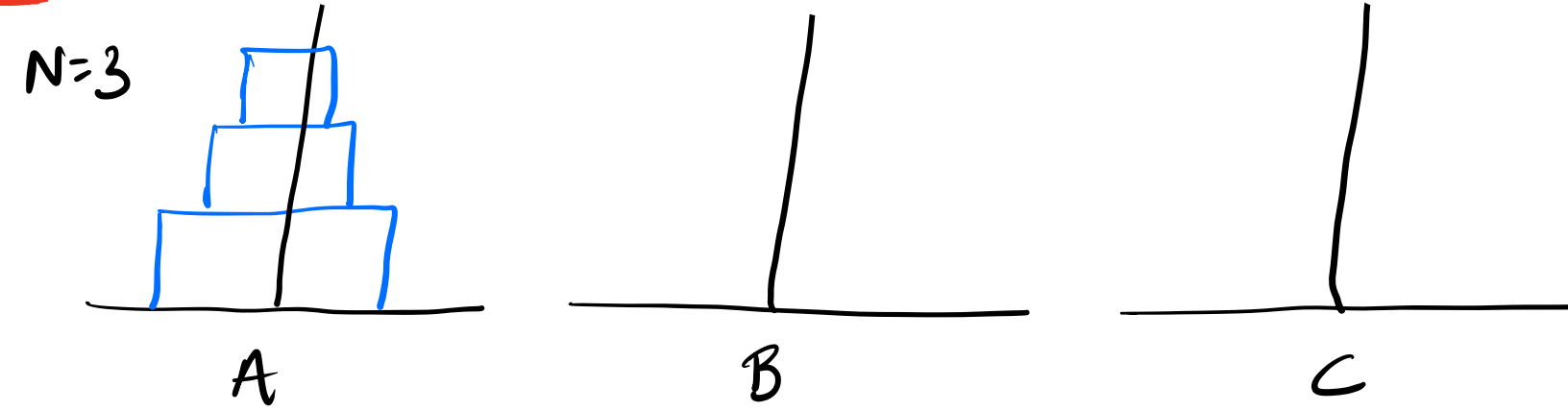


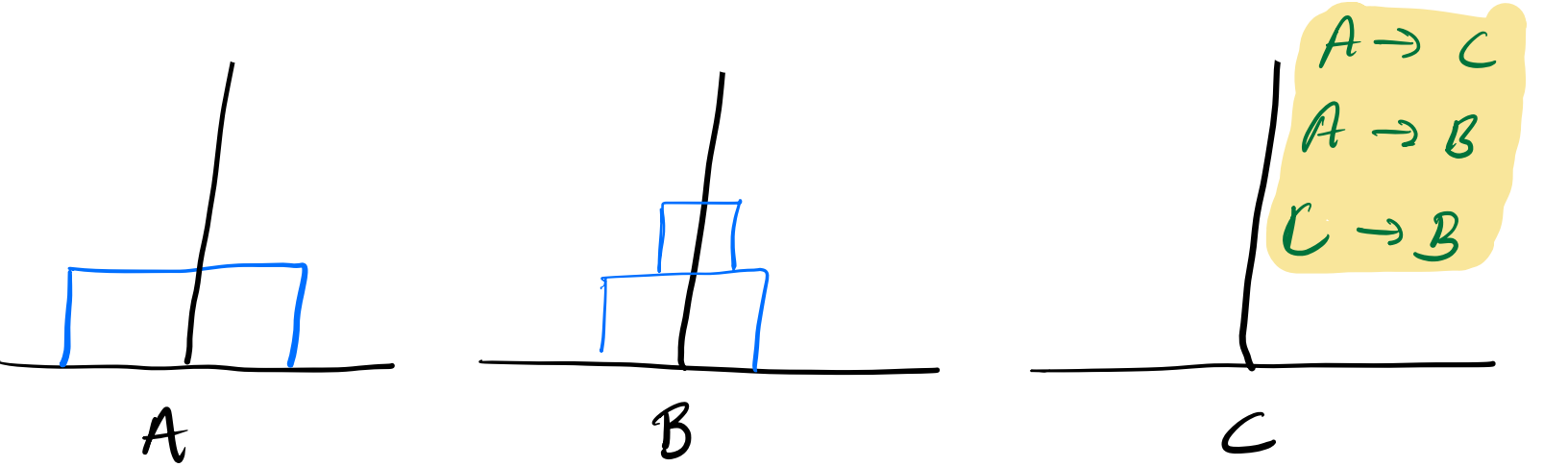
A → C

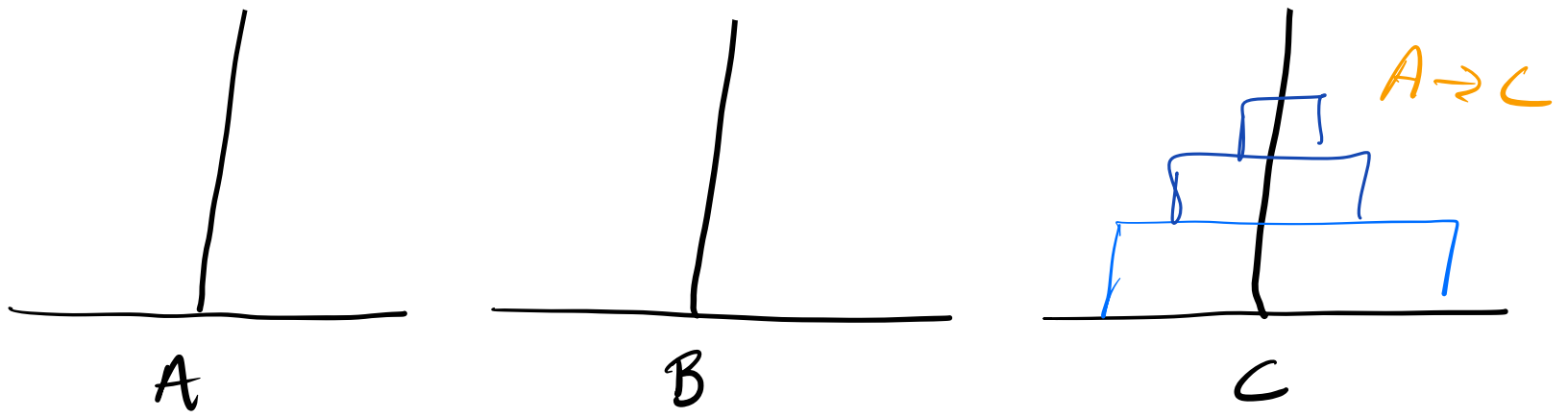


B → C







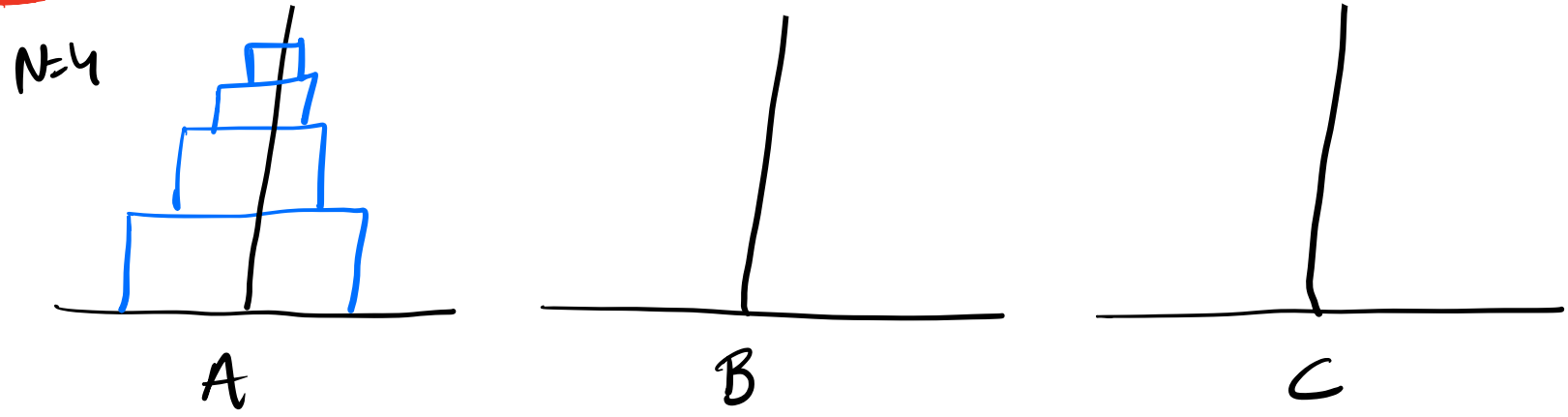


Now move B → C via A

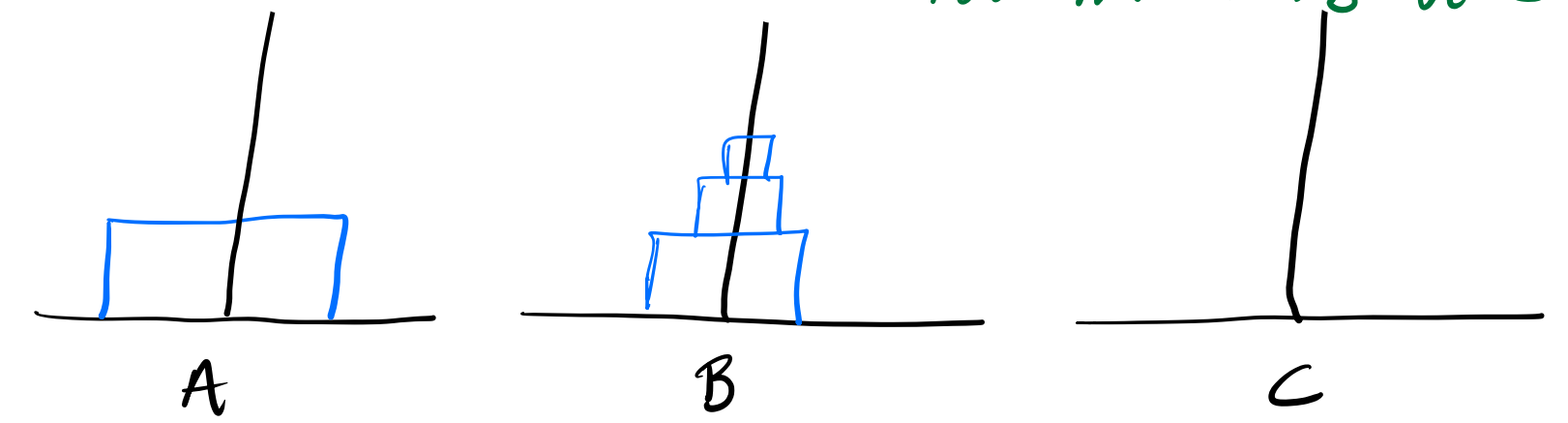
B → A

B → C

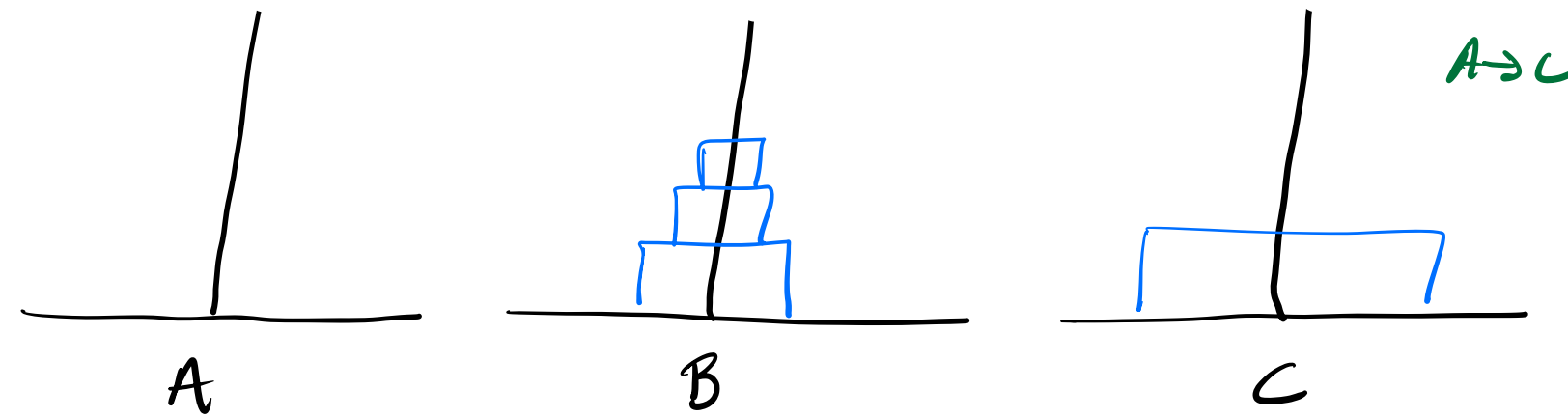
A → C



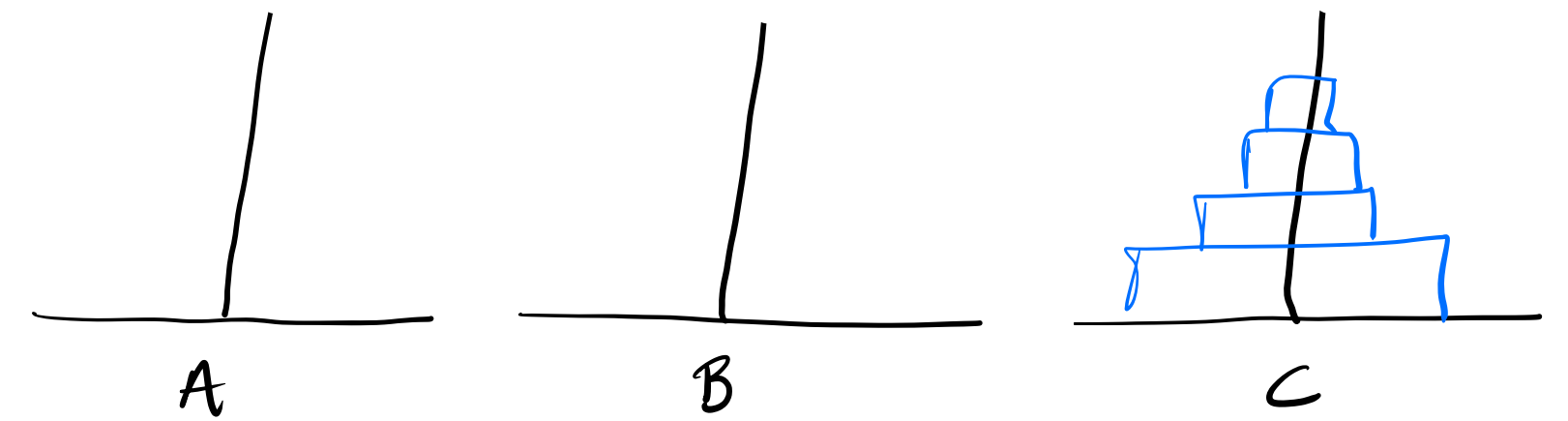
Put n-1 A → B via C



A → C



Now move n-1 B → C via A



- Algorithm

1) Move n-1 from A → B via C
2) Move from A → C
3) Move n-1 from B → C via A

```
void toh (int N, int src, int dest, int temp){
    if (N==1)
        print (src + "-" + dest)
    toh (n-1, src, temp, dest)
    print (src + "-" + dest)
    toh (n-1, temp, dest, src)
}
```

Time Complexity: $2^n - 1 \Rightarrow O(2^n)$

Moves	N = 1	1	$2^1 - 1$
	N = 2	3	$2^2 - 1$
	N = 3	7	$2^3 - 1$
Generalize	N = 4		$2^4 - 1$

toh (3, source A, dest C, temp B)

toh (2, A, B, C)

↳ toh (1, A, C, B)

A → C

A → B

toh (1, C, B, A)

C → B

- A → C

toh (2, B, C, A)

toh (1, B, A, C)

B → A

B → C

toh (1, A, C, B)

A → C

A → C

A → B

C → B

A → C

B → A

B → C

A → C

A → C

A → B

C → B

A → C

B → A

B → C

A → C

Q Print all valid paranthesses of len = 2N

Eg → N = 1 ( )

N = 2 (( )) ( ) ( )

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

(( )) ( ) ( ) ( ) ( ) ( ) (( ) ( ))

Idea: try to create all posible paranthesses recursively

len = 12

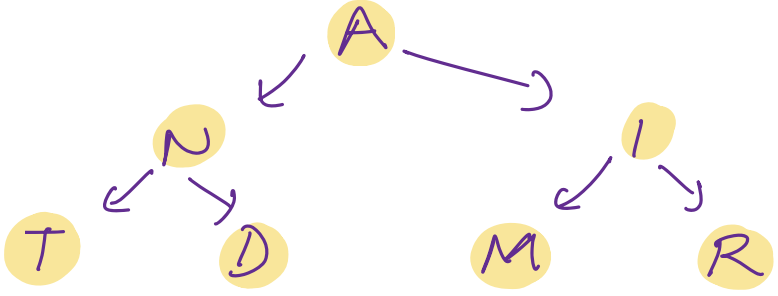
i=7

at all times open ≥ close

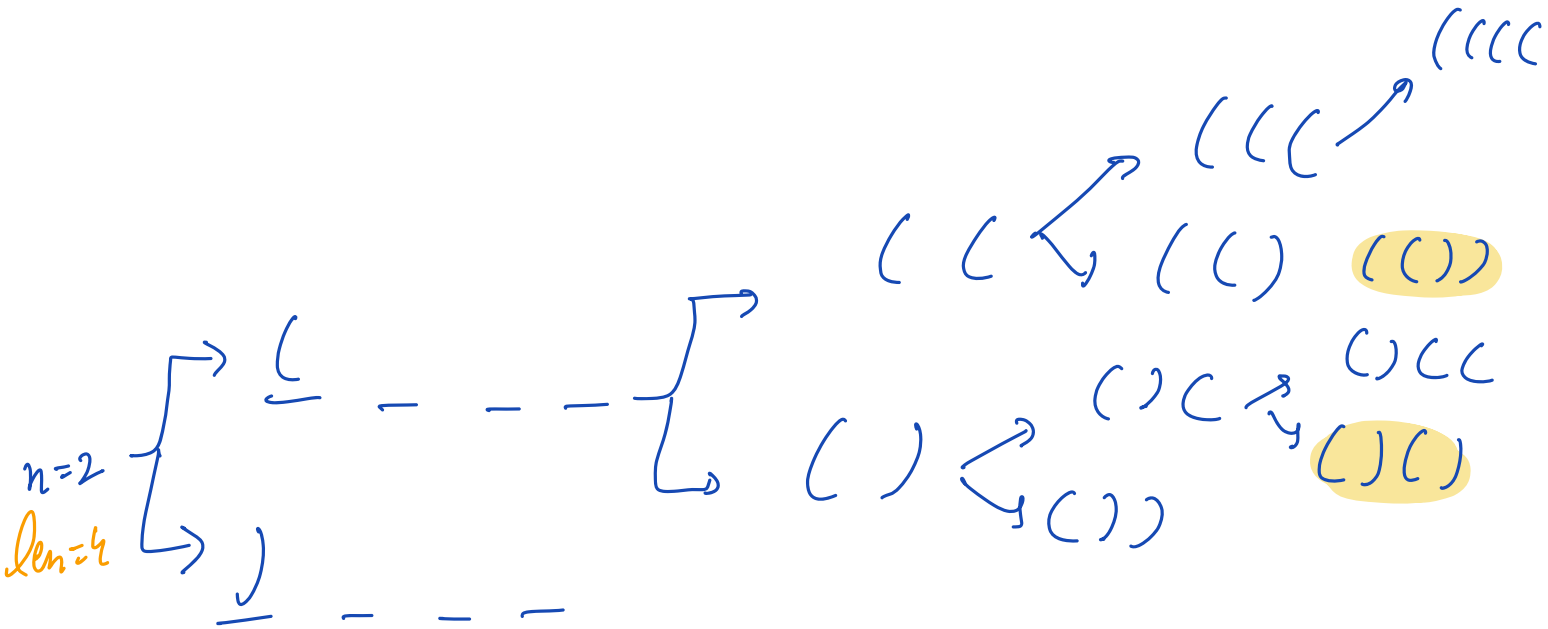
Obs: something like ( )) - - - - is always invalid. How to use this to better the approach.

Backtracking

Find AIM



Palanthesis via backtrack



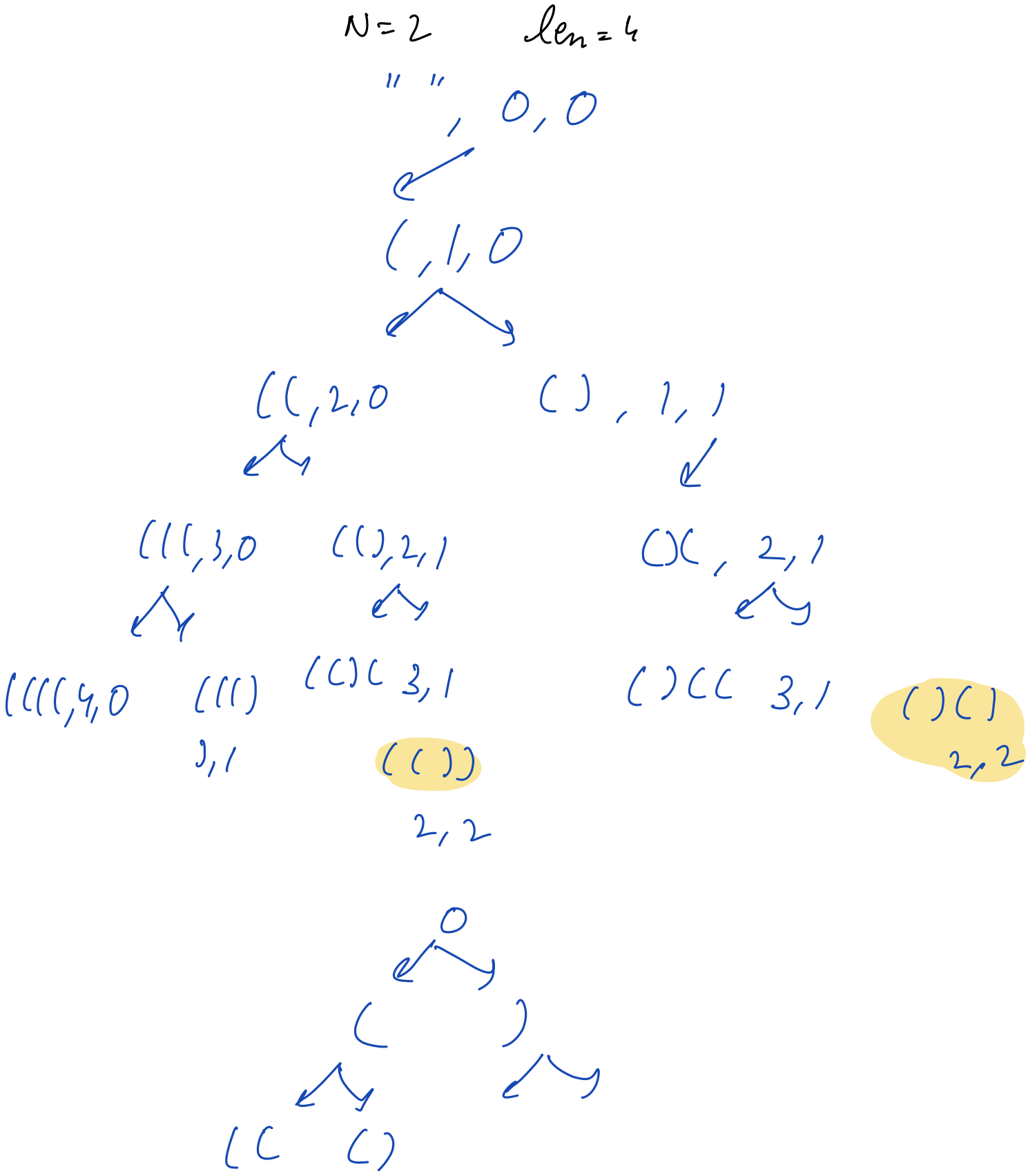
obs: open >= close at all times.

at the end open equal to close

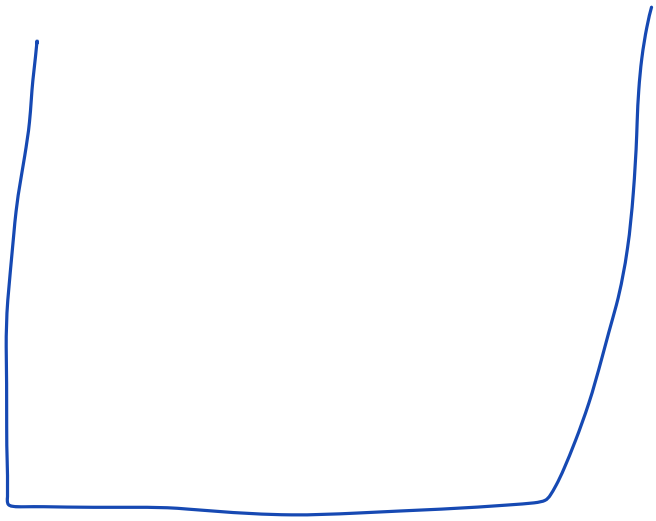
Code

```
void solve(string s, N, open, close){
    if(s.len() == 2N){
        if(open == close){
            print(s)
        }
        return;
    }
    if(open < N)
        solve(s+"(", N, open+1, close)
    if(close < open)
        solve(s+")", N, open, close+1)
}
```

TC: $O(2^n)$


N = 2
len = 4
"", 0, 0
(, 1, 0
((, 2, 0
(), 1, 1
(((, 3, 0
(()
)(, 2, 1
((((, 4, 0
(((
), 1
(()(
3, 1
(())
2, 2
)((
3, 1
()()
2, 2
0
(
)
((
()

```
void solve (n) {
    if (n==0)
        return
    solve (n-1)
    print (n)
}
```



3 2 1

```
void solve (int n) {

    mysolver
```