Time Complexity (using recurrence relation)

Space Complexity

Recursion Quiz

Tower of Hanoi

Print valid Paranthesis

① pow(a,n) = pow(a,n-1) * a

↓

T(n) = T(n-1) + O(1)

TC (using recurrence relation)

1. Factorial

factorial(n) = factorial(n-1) * n

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

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

* $T(n) = T(n-2) + 2$ (After 2 steps)

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

* $T(n) = T(n-3) + 3$ (After 3 steps)

$T(n) = T(n-k) + k$ (After k steps)

$T(0) = O(1)$

$n - k = 0 \Rightarrow k = n$

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

$= T(0) + n$

$T(n) = 1 + n$

TC : $O(n)$

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

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

* $T(n) = 2T(n/2) + 1$ (After 1 steps)

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

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

* $T(n) = 4T(\frac{n}{4}) + 3$ (After 2 steps)

$T(\frac{n}{4}) = 2T(\frac{n}{8}) + 1$

$T(n) = 4[2T(\frac{n}{8}) + 1] + 3$

* $T(n) = 8T(\frac{n}{8}) + 7$ (After 3 steps)

(After k steps)

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

$T(0) \to O(1)$ $\qquad \frac{n}{2^k} = 1$

$\Rightarrow n = 2^k$

$\Rightarrow \log_2 n = k$

$$T(n) = 2^{\log_2 n} T\left(\frac{n}{2^{\log_2 n}}\right) + 2^{\log_2 n} - 1$$

$$= n\, T\left(\frac{n}{n}\right) + n - 1 \qquad T\left(\frac{n}{n}\right) \to O(1)$$

$$T(N) = 2N - 1$$

$$TC : O(N)$$

(3)

```
pow(a,n) {
    BC
    x = pow(a, n/2)
    return x * x
}
```

* $T(n) = T(n/2) + 1$ (After 1 step)

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

* $T(n) = T(n/4) + 2$ (After 2 step)

(After k steps)

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

$T(0) = O(1)$ $\quad \frac{n}{2^k} = 1$

$\Rightarrow n = 2^k \quad \Rightarrow \log_2 n = k$

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

$TC: O(\log_2 n)$

4. Fib (n) = Fib $(n-1)$ + Fib $(n-2)$

* $T(n) = 2T(n-1) + 1$ (After 1 step)

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

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

* $T(n) = 4T(n-2) + 3$ (After 2 step)

* $T(n) = 2^k T(N-K) + 2^k - 1$ (After k step)

$T(0) = O(1)$ $\quad \Rightarrow N - K = 0$

$\Rightarrow K = N$

$$T(n) = 2^N \, T(0) + 2^N - 1$$

$$T(n) = 2^N$$

$$TC : O(2^N)$$

Space complexity

↓

amt of space taken by recursive algo

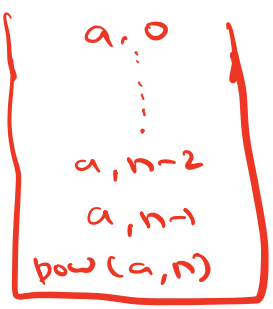
↓ ↓

space taken inside recursive fn + max size of fn call stack at any point

SC of factorial : O(N)

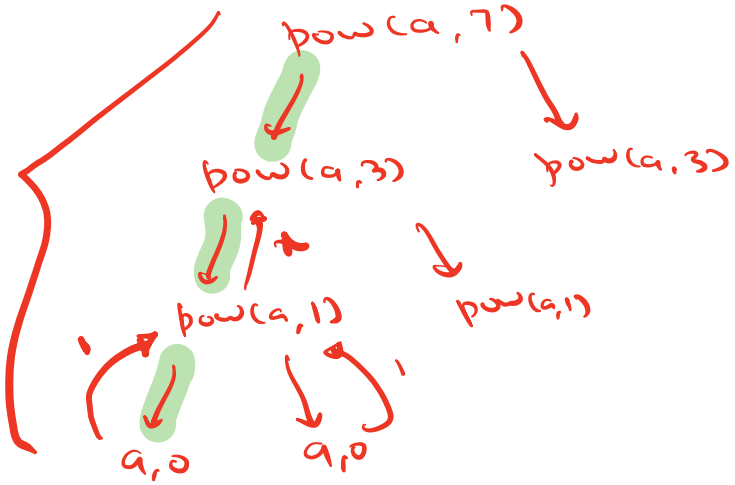
SC of 1st power : O(N)

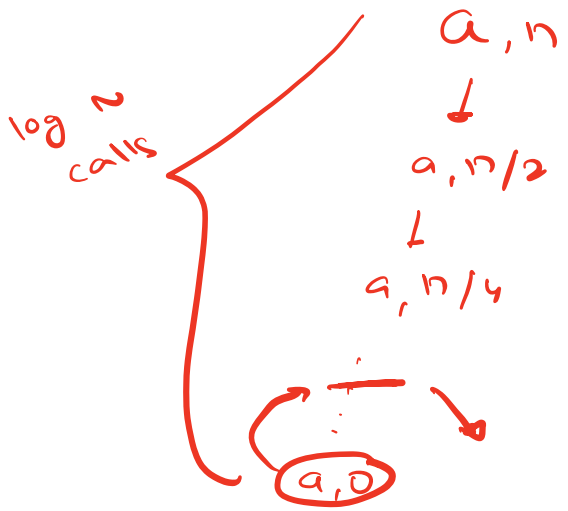
$a^n = a^{n-1} + a$



SC of 2nd power

pow a^n = pow(a, n/2) + pow(a, N/2)





SC : O(log N)

Depth of recursion tree /
Max no. of recursive calls in stack at any point of time

SC of 3rd power

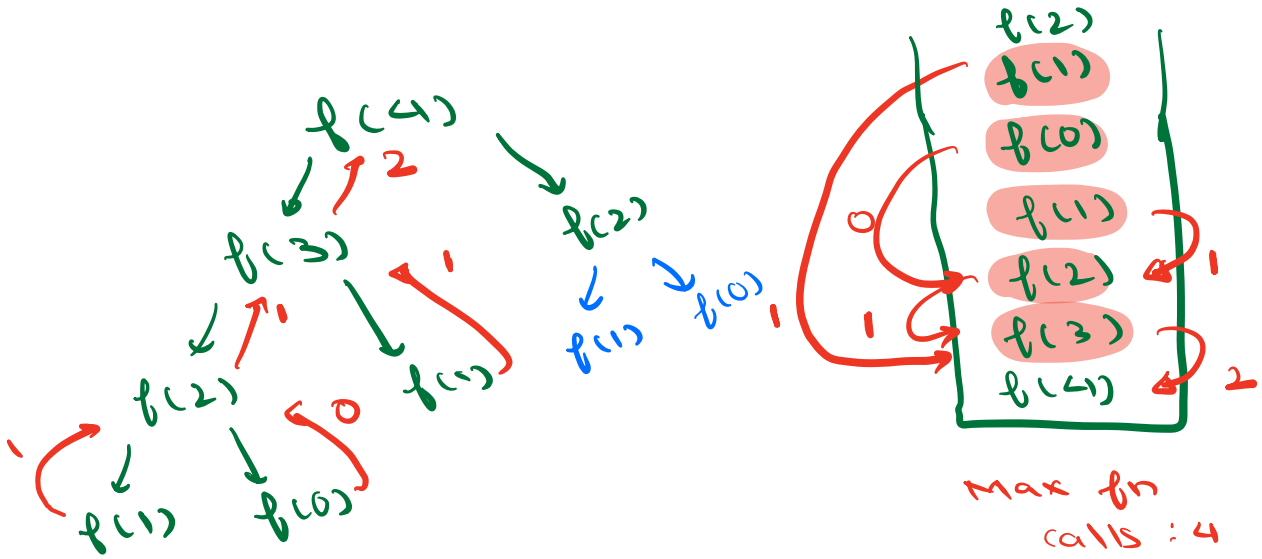
x = pow(a,n/2)

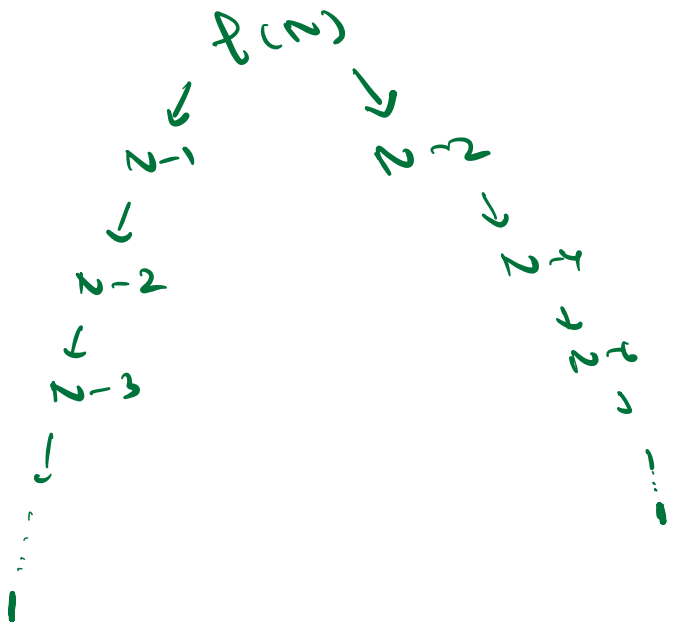
pow a^n = x * x

SC : $O(\log_2 N)$

a,n
a,n/2
a,n/4
a,0

3. Fib (n) = Fib (n-1) + Fib (n-2)



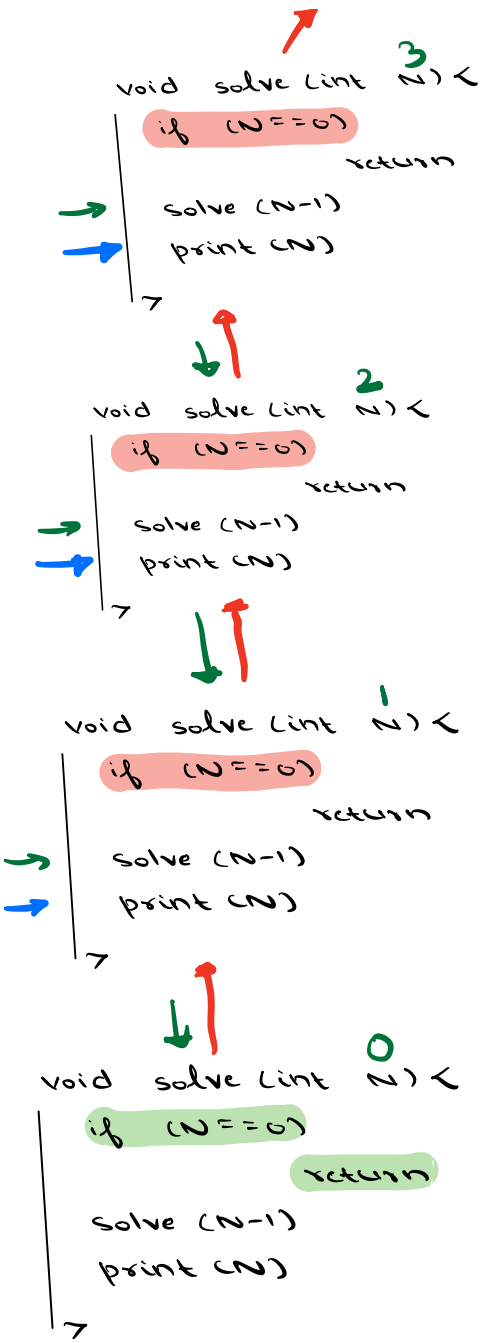


10:13

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

O/P

1 2 3



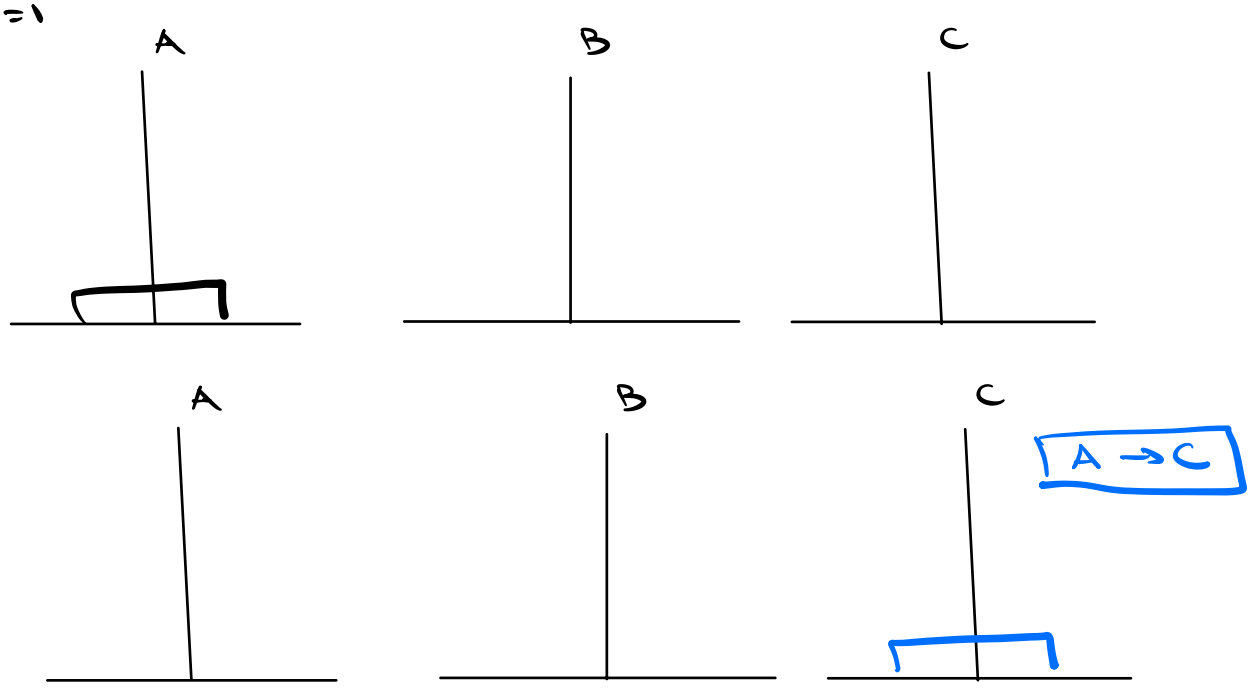
Tower of Hanoi

There are N disks placed on tower A of different sizes.

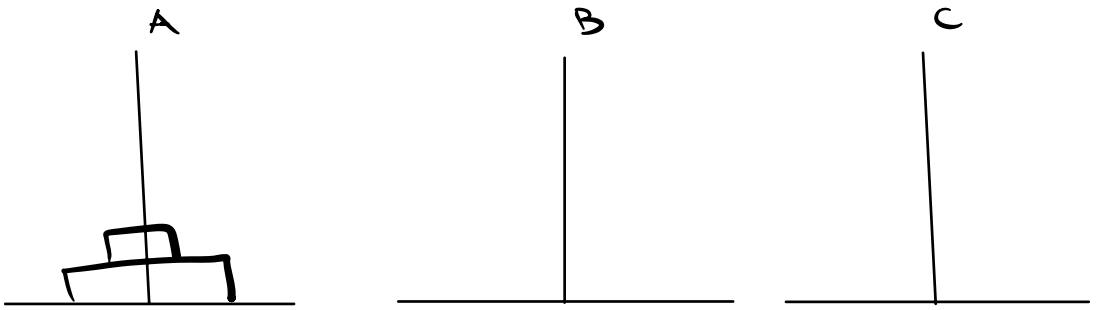
Goal: Move all disks from tower A to C using tower B if needed.
(Print movement of discs)

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

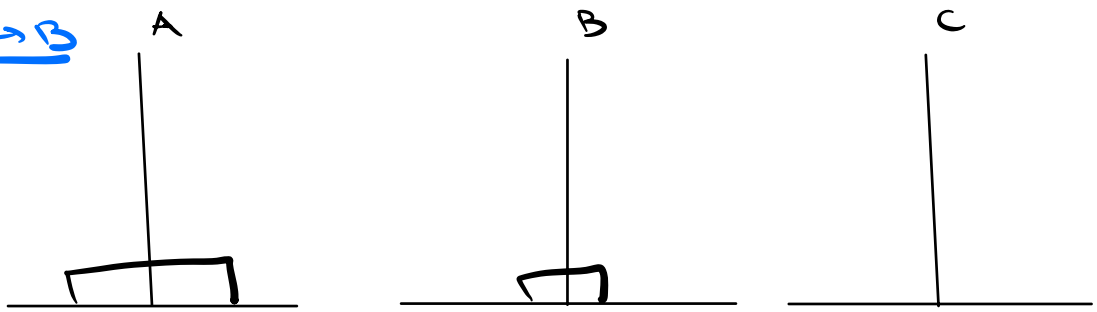
n=1



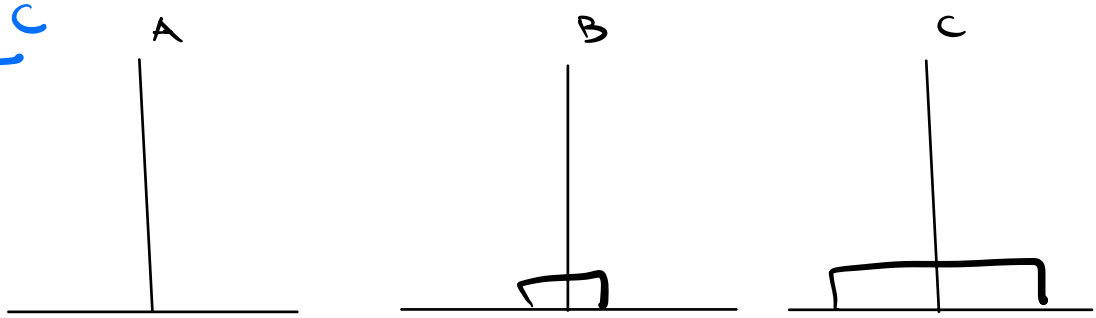
n = 2



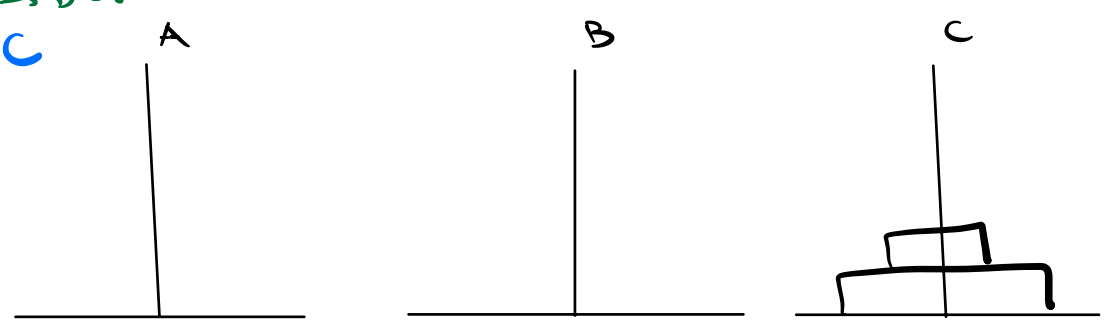
Src → Temp
A → B



Src → Dest
A → C



Temp → Dest
B → C



Move 2 disks from A → C using B

src temp dest
A B C

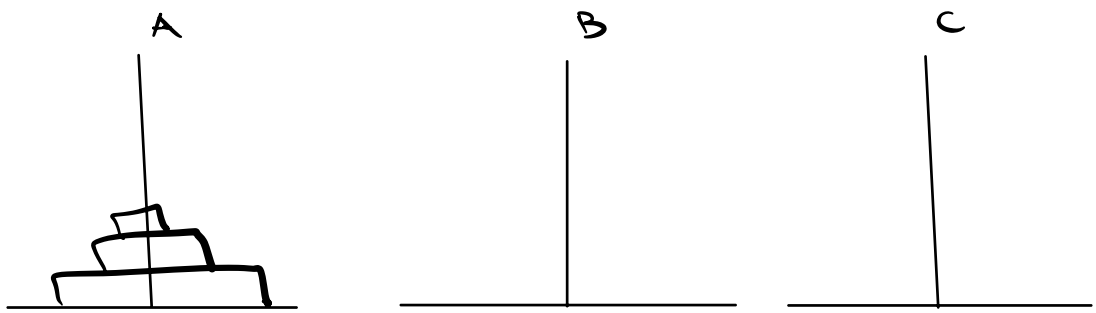
Src → Temp (smaller only)

Src → Dest

Temp → Dest

---

n = 3



① Move 2 disks from A to B using C

Src	temp	dest
A	C	B

Src → Temp

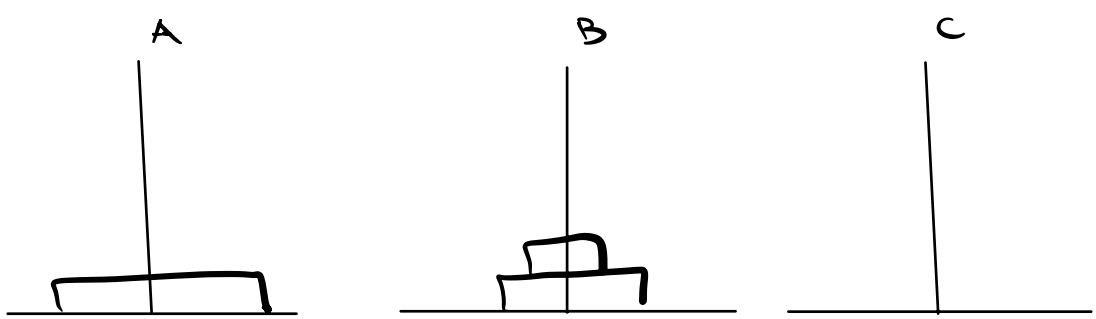
Src → Dest

Temp → Dest

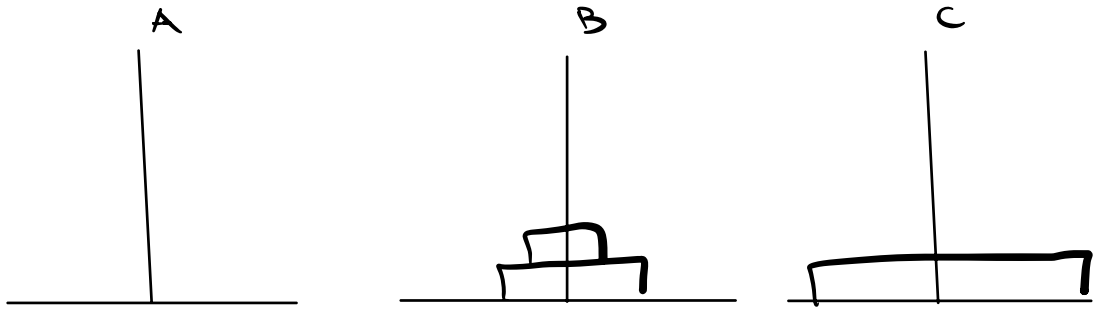
A → C

A → B

C → B



② A → C

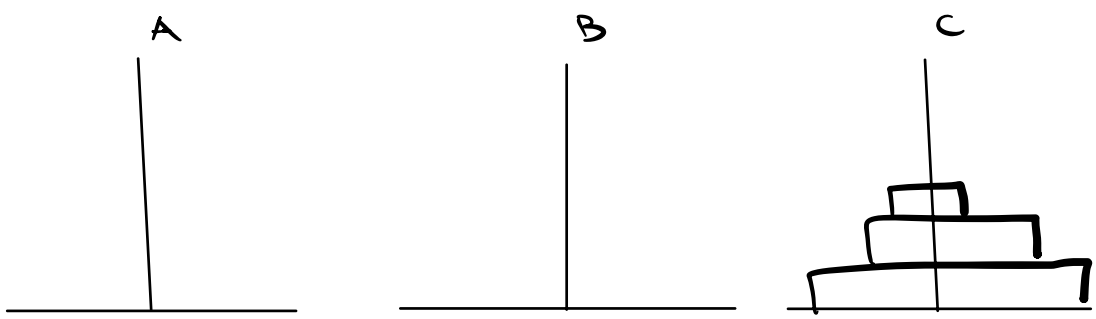


③ Move 2 disks from B → C using A

Src → Temp
Src → Dest
Temp → Dest

B → A
B → C
A → C

Src	Temp	Dest
B	A	C

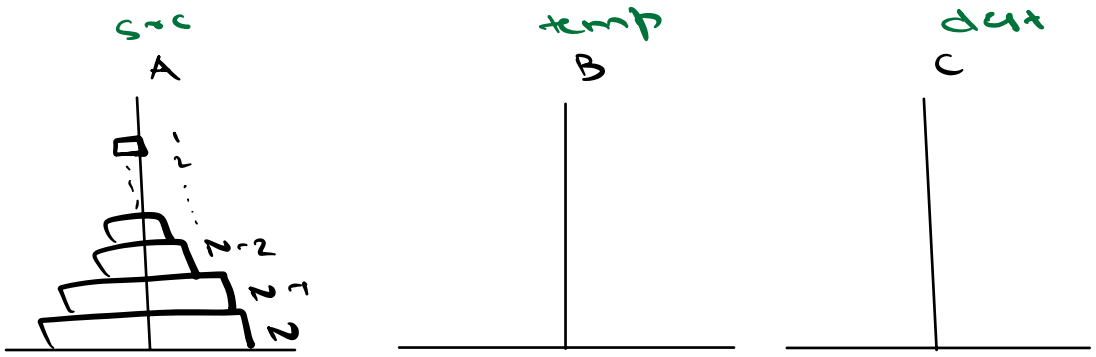


A → C
A → B
C → B

A → C

B → A
B → C
A → C

N Disks



① Move N-1 Disks from src → temp

② print (N : src → dst)

③ Move N-1 Disks from temp → dst

```
// Given all inputs, print moves to transfer
                    N disk from src → dst
void TOH (int n, char src, char temp, char dst){
1 |   if (n == 0) return
2 |   TOH (n-1, src, dst, temp)
3 |   print (n : src → dst)
4 |   TOH (n-1, temp, src, dst)
  }
```

TOH (3, a, b, c) ✗ 2 3 4

2. TOH (2, a, c, b) ✗ 2 3 4

2. TOH (1, a, b, c) ✗ 2 3 4

2. TOH (0, a, c, b) 1 2 3 4

3. print (1: a → c)

4. TOH (0, b, a, c) 1 2 3 4

3. print (2: a → b)

4. TOH (1, c, a, b) ✗ 2 3 4

2. TOH (0, c, b, a)

3. print (1: c → b)

4. TOH (0, a, c, b)

3. print (3: a → c)

4. TOH (2, b, a, c)

1: A → C
2: A → B
1: C → B
3: A → C

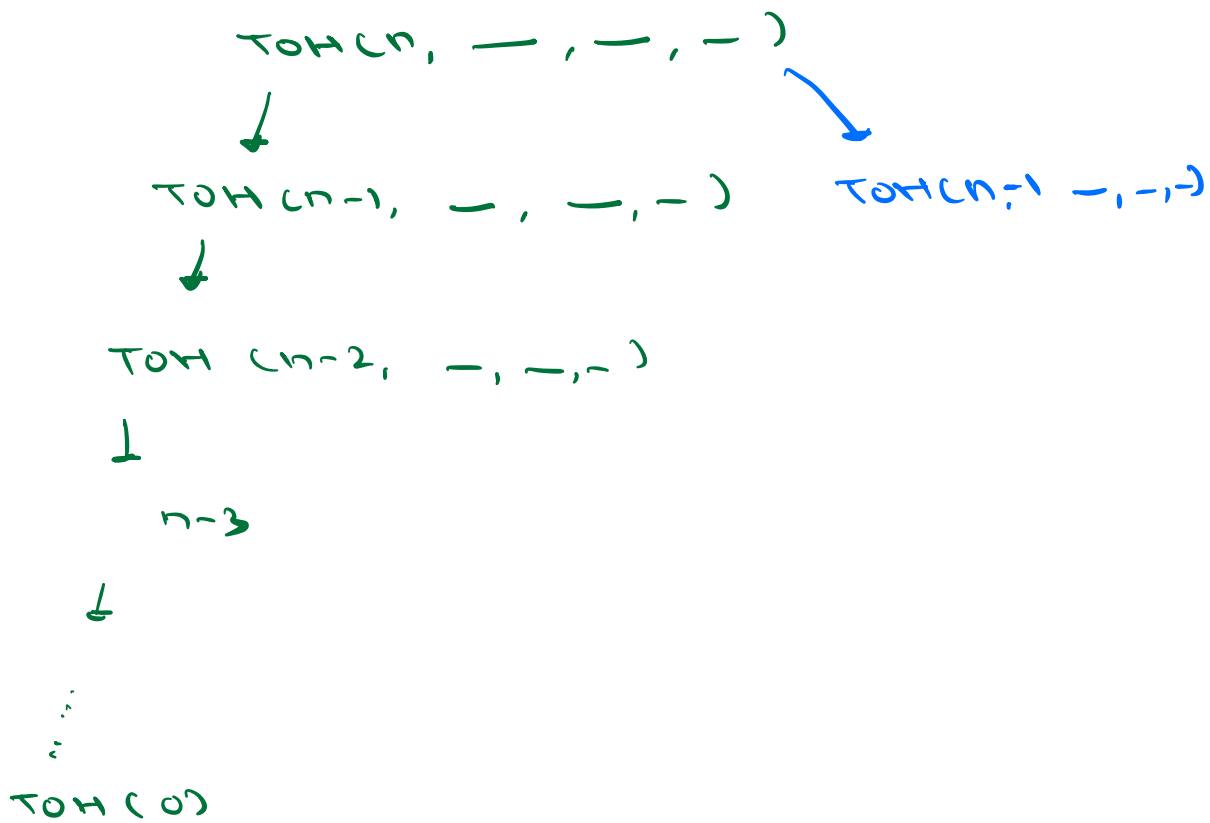
How many recursive calls?

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

TC: $O(2^N)$

SC : fn call stack

SC : O(N)



Print valid Paranthesis

( )

Print all valid paranthesses of length 2N for given value N.

① Valid Paranthesis means equal no. of opening and closing brackets.

② Opening brackets → closing bracket

N = 1 ( )

✗ ((,)) ,)(

N = 2 (( )) , ()()

✗ ((((,)))),
))((,)()(

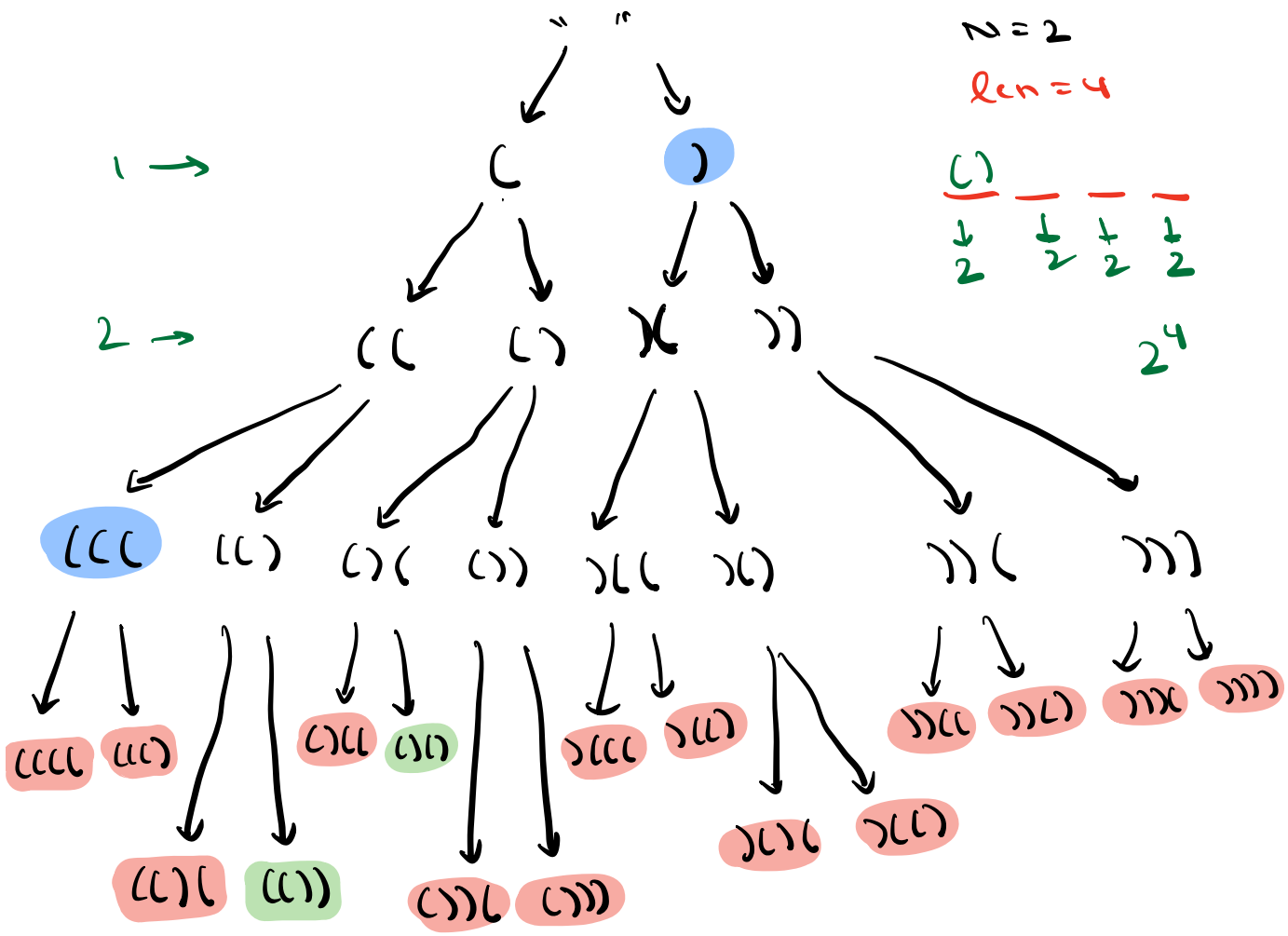
N = 3

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

((((((X
))))))
))((()(

BF : Generate all possible strings

How to check validity ?

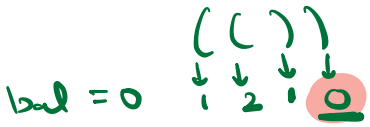


How to check validity ?

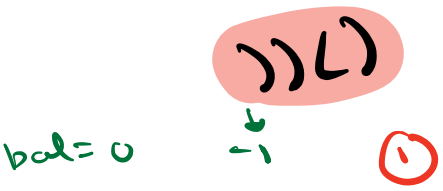
balance = 0

((()(

1 2 1 2



bal = 0 → open == close
+1 -1



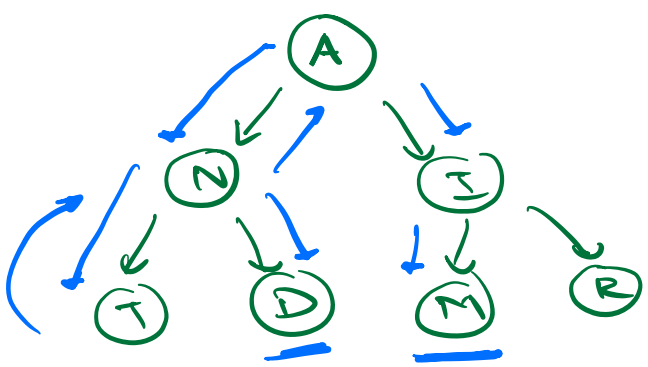
close > open

Seq is invalid
1. bal ever -ve
2. bal → +ve

TC : $O(2^{2N})$ → 2^N
SC : $O(N)$

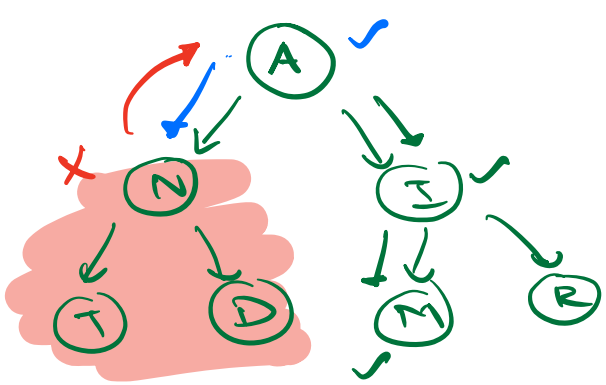
Backtracking

word = AIM



Tree of words

Paths
- ANT
- AND
- AIM
- AIR



word = A I M

This is called pruning our recursion tree

↓

avoid unnecessary paths

① if (opening > closing)
put a ')'

② if (opening < n)
put an '('

```
                                  "", N, 0, 0
void solve (str, N, opening, closing) {
    if (str.length == 2N) {
        print(str)          return
    }
    if (opening < N)
        solve (str+'(', N, opening+1, closing)

    if (opening > closing)
        solve (str+')', N, opening, closing+1)
}
```

HW: Dry run → visualization (log 2 base 2)

TC: $O(2^N)$

SC: $O(N)$

HW

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

N = 3 O/P = ?

N = -3 O/P = ?