Recursion - 2

N=3

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

output: 1 2 3

```
solve (N=3) {
  solve (N=2) {
    solve (N=1) {
      solve (N=0) {
        return
      }
      print(1)
    }
    print(2)
  }
  print(3)
}
```

N=3

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

output: 3 2 1

```
solve (N=3) {
  print(3)
  solve (N=2) {
    print(2)
    solve (N=1) {
      print(1)
      solve (N=0) {
        return
      }
    }
  }
}
```

```
void solve ( int N) {      → N=-3
    if (N==0) return
    print(N)
    solve (N-1)
}
```

Stack overflow error

solve (N=-3){
print (-3)
solve (N=-4){
print(-4)
solve (N=-5){
⋮
infinity
never become N=0

Tower of Hanoi

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

Move all the disk from A to C. (using tower B)

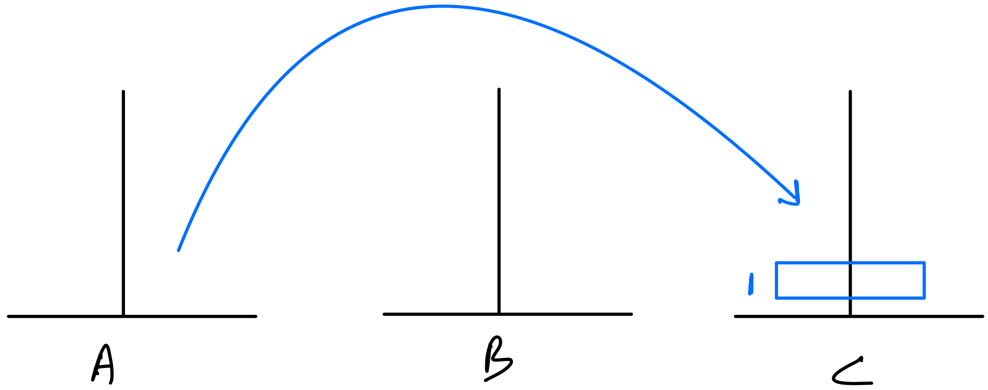
Constraints :

1. Only 1 disk can be moved in 1 step
2. Large disk can't be placed on a small disk at any step.

Print the movement of disk from A to C in minimum steps.

N=1

o/p: 1, A → C



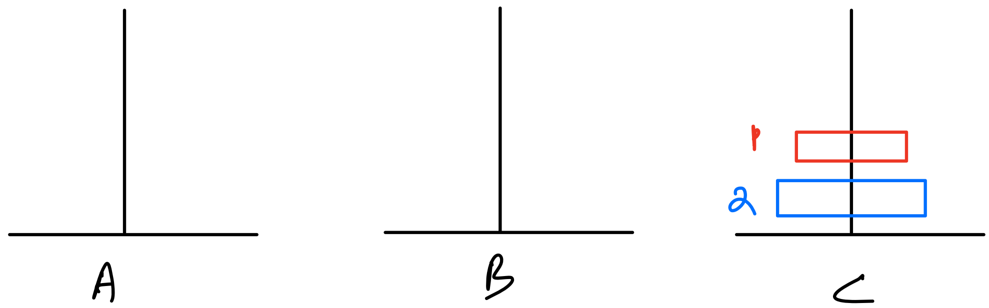
N=2

o/p:

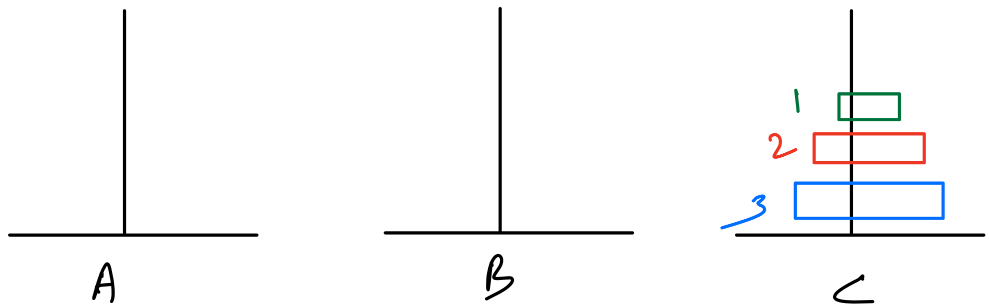
$\boxed{1,\ A \to B}$

2, A → C

$\boxed{1,\ B \to C}$



N=3



o/p:

1, A → C
2, A → B
1, C → B

→ move 2 disks from A to B

3, A → C → 3rd disk, A → C

1, B → A
2, B → C
1, A → C

→ move 2 disks from B to C

if we have N disks to move

1. Move N-1 disks from A → B
2. N^{th} disk , A → C
3. Move N-1 disks from B → C

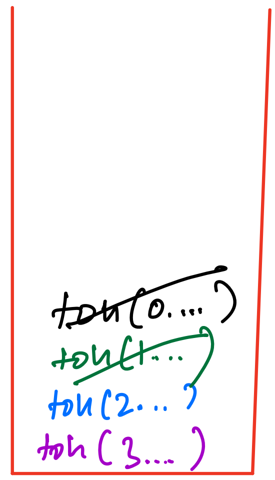
src dest

```
void ToH ( N, A , B, C ) {
    if ( N==0 ) { return }
    ToH( N-1, A , C , B )
    print (N, "A → C")        // move Nth disk from A→C
    ToH( N-1, B , A, C )
}
```

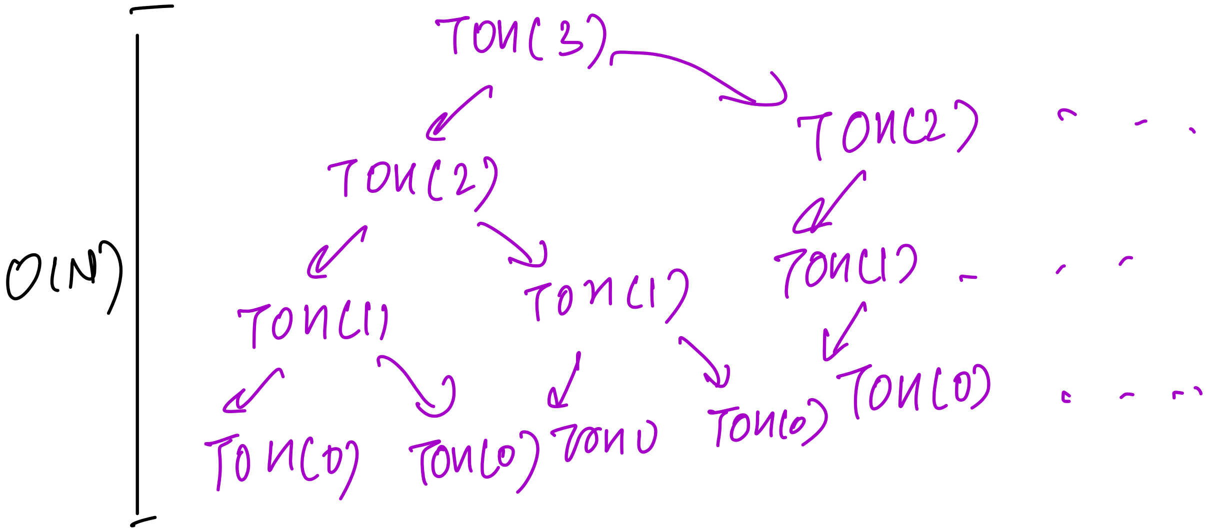
N	# steps			
1	1	= 1	2-1	2^1-1
2	1+1+1	= 3	4-1	2^2-1
3	3+1+3	= 7	8-1	2^3-1
4	7+1+7	= 15	16-1	2^4-1
⋮				
N				2^n-1

TC of recursive code = $O(2^n)$

```
TOH(N=3, 'A', 'B', 'C') {
    TOH(N=2, A, C, B) {
        TOH(N=1, A, B, C) {
            TOH(N=0, A, C, B) { return }
            print(1, A -> C)
            TOH(N=0, B, A, C) { return }
        }
        print(2, A -> B)
        TOH(N=1, C, A, B) {
            TOH(N=0, C, B, A) { return }
            print(1, C -> B)
            TOH(N=0, A, C, B) { return }
        }
    }
    print(3, A -> C)
    TOH(N=2, B, A, C) {
        ⋮
    }
}
```



SC = O(N)



Question

Print all valid parenthesis of length 2N for a given value of N.

1. travel from left to right
   # open >= # close

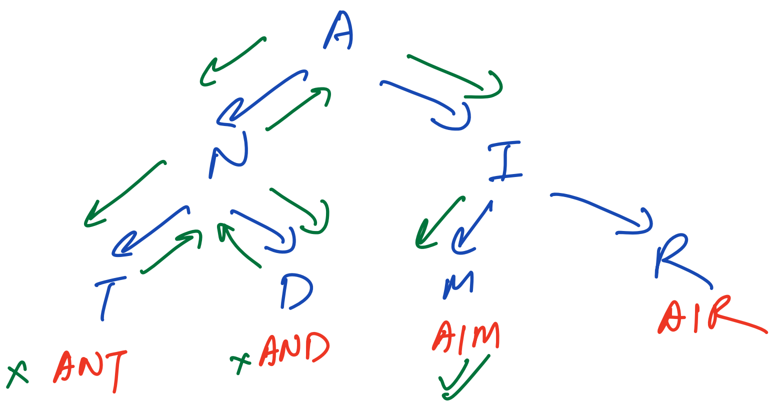
2. total # open = total # close

N=1   ( )   ~~)(~~

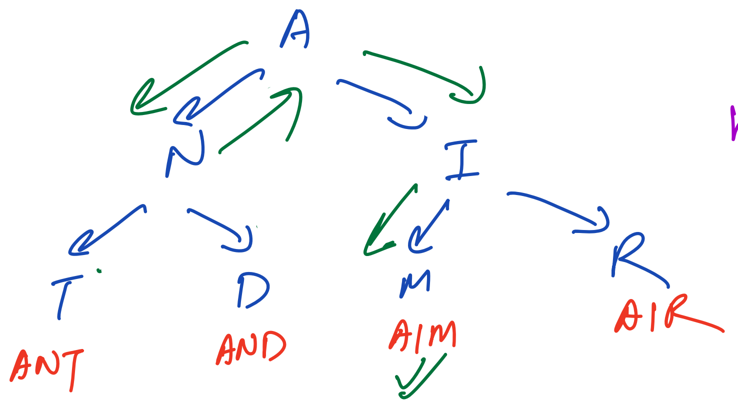
N=2   (()) , ~~(()~~ , ()()

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

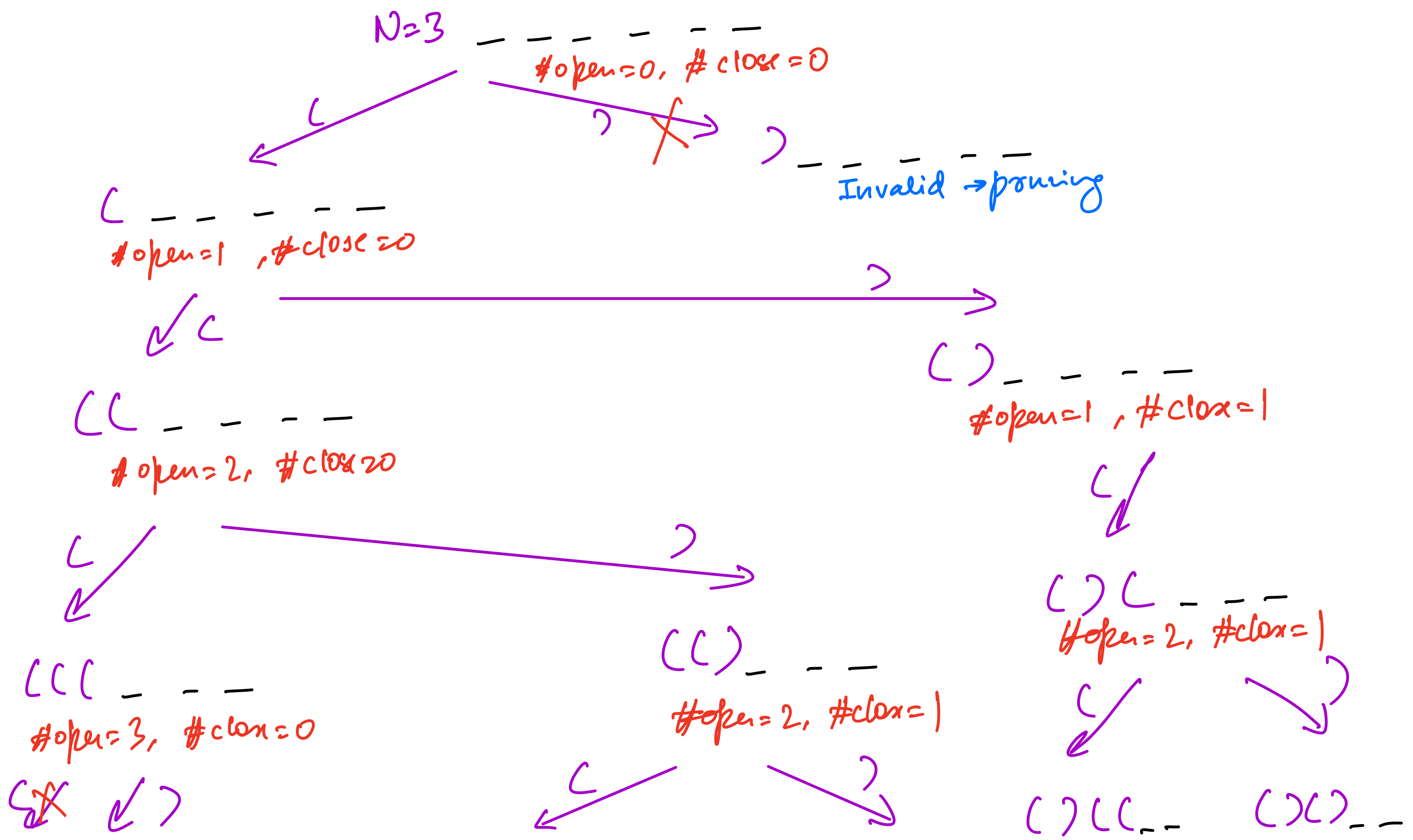
Backtracking ?

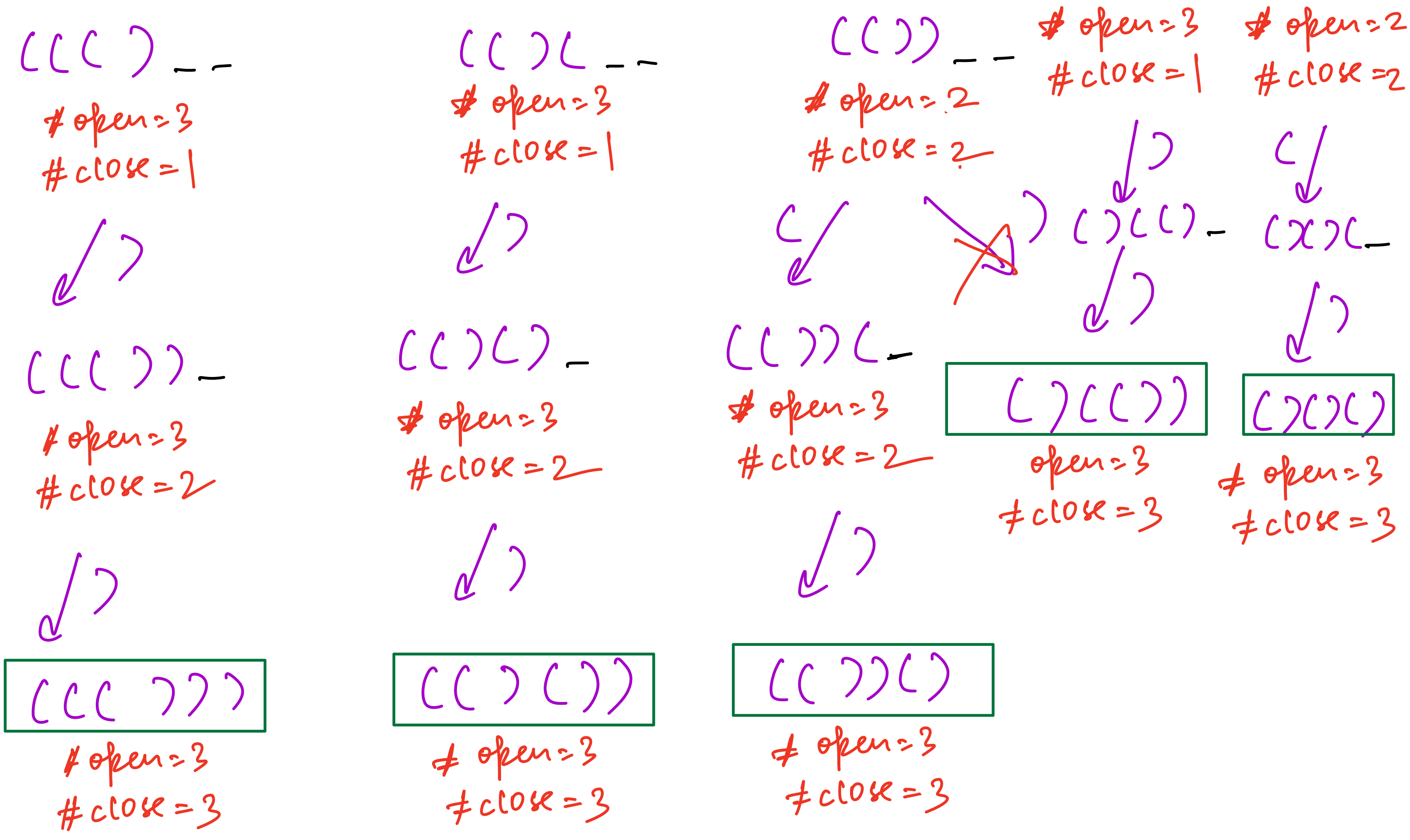


AIM = ?



better approach






```
void solve(N, open, close, str) {
      if( open==N && close==N) {
            print(str)
            return
      }
      if (open < N) {
            solve(N, open+1, close, str + '(')
      }
      if(close < open) {
            solve(N, open, close+1, str + ')')
      }
}
```

TC : ?

at every point there are 2 choices

total calls : $\approx \underbrace{2 \times 2 \times 2 \cdots\cdots \times 2}_{2n \text{ times}}$

$\approx 2^{2n} \approx 4^{n}$

$TC : O(2^n)$

eg

-	2	
--	00 01 10 11	4
---	8	

$SC : O(N)$

Tip not Rule :

if constraint on N is very small like

$N <= 20$, $N <= 15$ etc.

then probably expected $TC = O(2^n), O(n!) \ldots$

which could lead to $\boxed{\text{recursion based sol}^n.}$