- Basics of Recursion
  - Fib. Sequence
  - Output based questions
  - Tower of Hanoi
  - Generate all parenthesis.

Recursion : function calling itself.

→ solving a problem using smaller instances of the same problem. [sub-problem]

Q) sum of first N natural numbers

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

∴ sum(4) is a sub-problem.

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

Write recursive code. =>

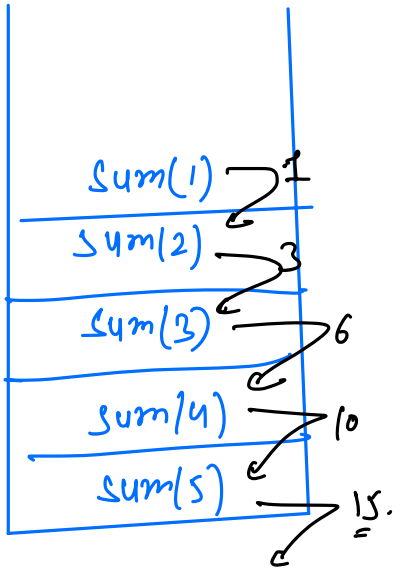
① Expectation/Assumption → What should your function do.

② Main Logic → Solve main problem using sub-problem.

③ Base case → stopping case/condition for recursion.

//Expectation - find & return sum of first N natural no's.

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



Time taken to find sum of first N natural no's → T(N)

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

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

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

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

$T(N) = T(N-3) + 3$

$T(N) = T(N-4) + 4$

⋮

After K steps, recursion stops.

$T(1) = 1.$

$T(N) = T(N-K) + K$

$N - K = 1 \Rightarrow K = N-1.$

$$T(N) = \underbrace{T(N-(N-1))}_{T(1)} + N-1$$

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

T.C → O(N)
S.C → O(N)

Fibonacci Series [N ≥ 0]

0	1	2	3	4	5	6	7	8	9	10
0	1	1	2	3	5	8	13	21	34	55

fib(N) = sum of previous two fibonacci numbers.

fib(N) = fib(N-1) + fib(N-2)

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

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

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

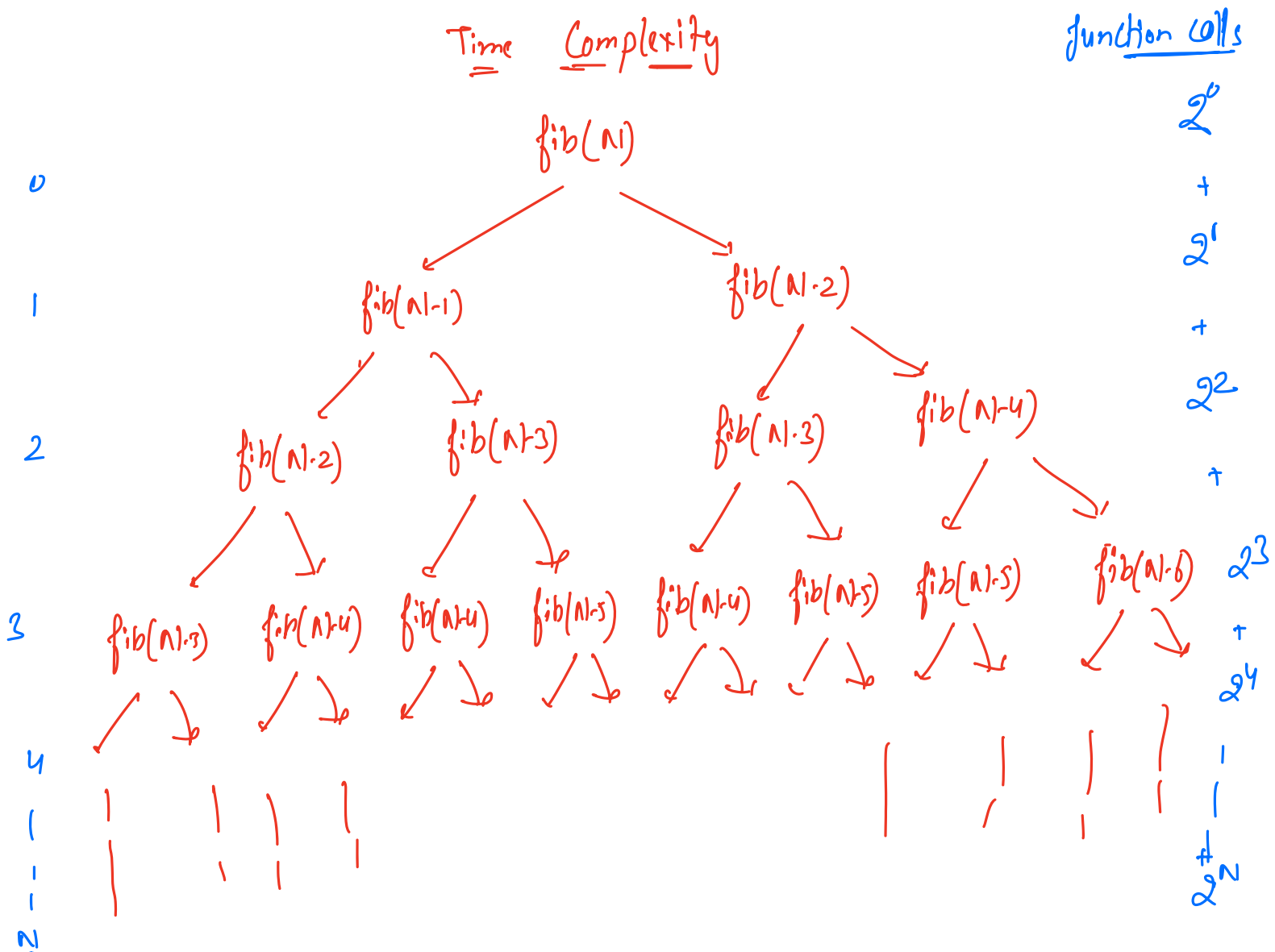
$$T(N-2) = T(N-3) + T(N-4) + 1$$

$$T(N) = T(N-2) + T(N-3) + 1 + T(N-3) + T(N-4) + 1 + 1$$

$$T(N) = T(N-2) + 2T(N-3) + T(N-4) + 3$$

× Substitution method.

Time Complexity



Total function calls: $2^0 + 2^1 + 2^2 + 2^3 + 2^4 + \cdots + 2^N$

$$\text{Sum} = \frac{a(r^{\text{no. of terms}} - 1)}{(r-1)}$$

$a \rightarrow$ 1st term

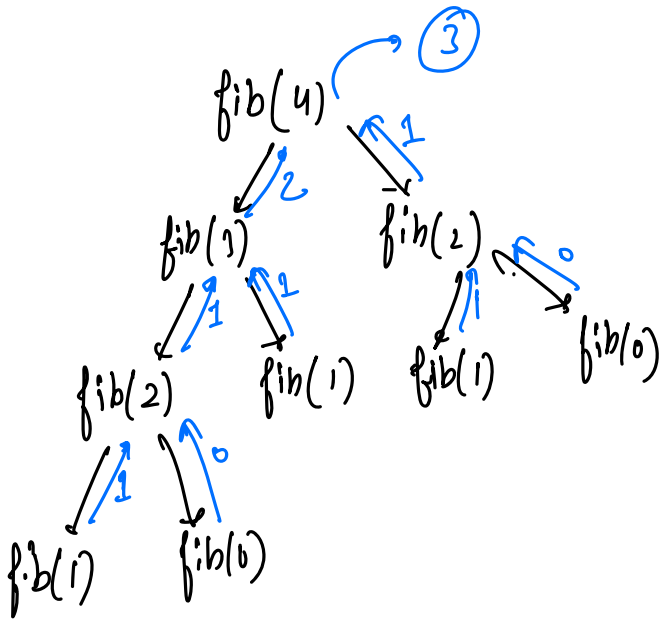
$r \rightarrow$ common ratio

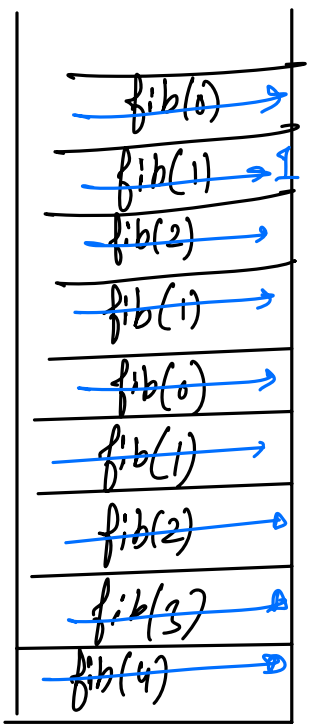
$$= \frac{1(2^{N+1} - 1)}{(2-1)} = 2^{N+1} - 1$$

$$= 2 \cdot 2^N - 1$$

T.C $\rightarrow O(2^N)$

S.C $\rightarrow O(N)$

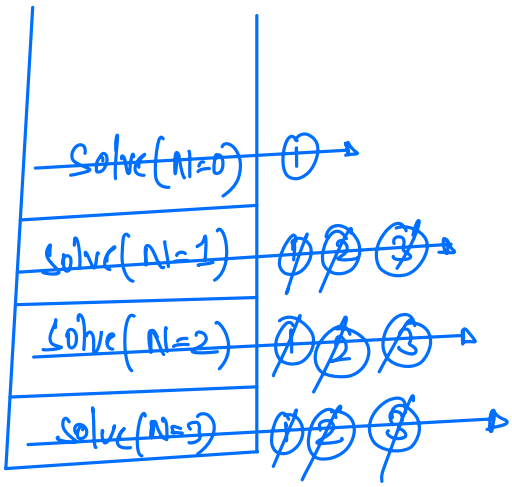




Output based problems

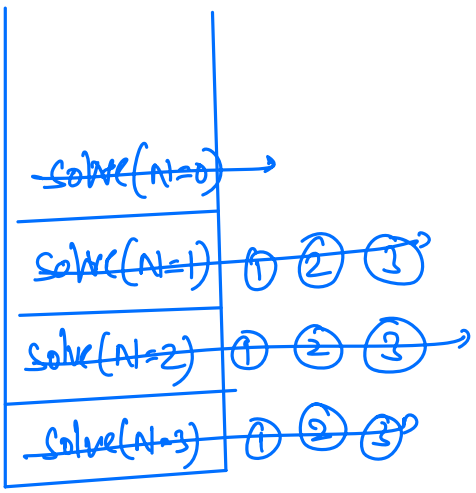
o/p → 1 2 3

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

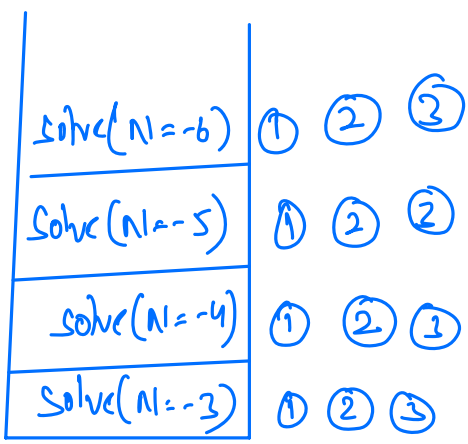


o/p → 3 2 1

②
```
void solve (N) {
    if (N == 0) return;
    print (N)
    solve (N-1)
}
```



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



-3 -4 -5 -6 __ [stack overflow]

Tower of Hanoi

→ Given 3 towers A, B and C

→ There are n-disks placed on tower A.

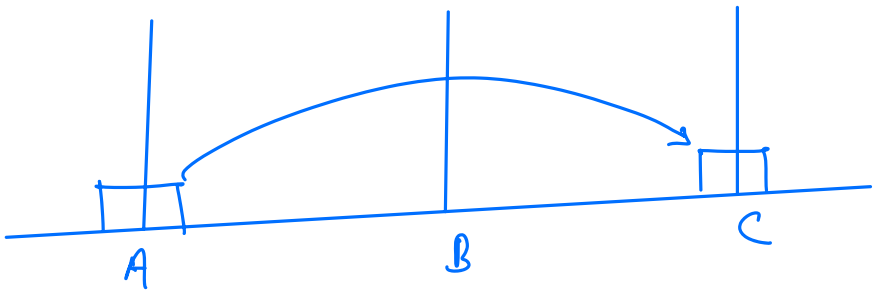
→ Move All the disks from A to C. {using B}

Note: ① Only one disk can be moved at a time. (topmost)

② larger disk cant be placed on a smaller disk.

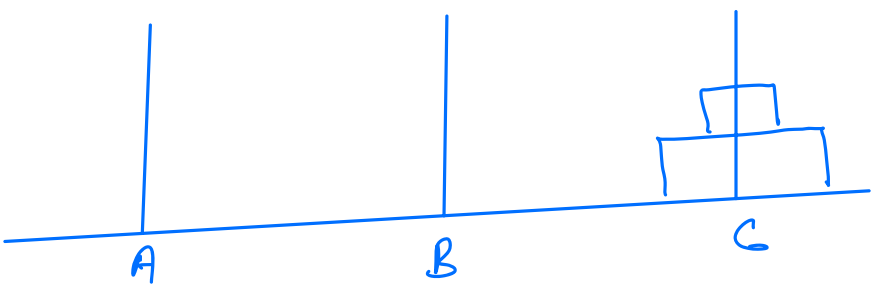
Q → Print movement of the disks.

N=1.



Move disk-1 A → C.

N=2.

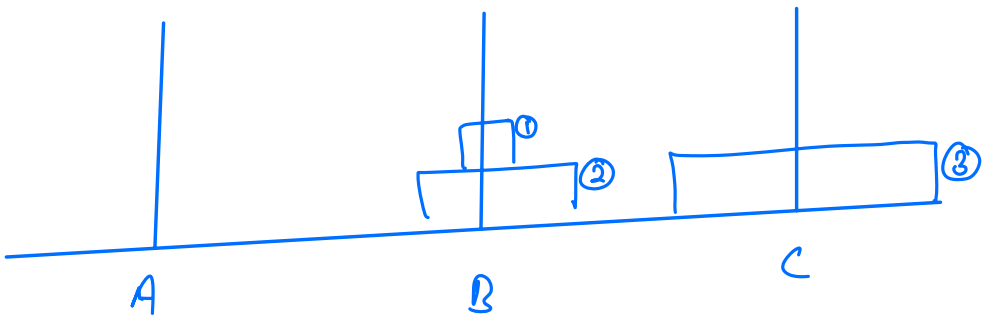


Move disk 1 A → B

Move disk 2 A → C

Move disk 1 B → C.

N=3



- Move disk 1 A → C
- Move disk 2 A → B
- Move disk 1 C → B

} Moving N-1 disks from A to B (using C)

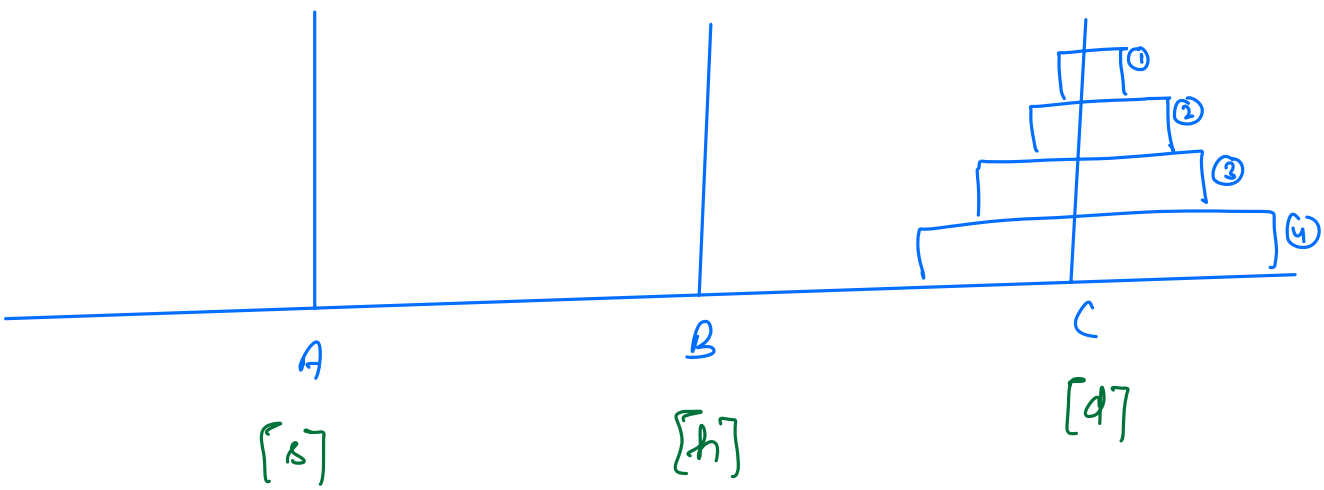
- Move disk 3 A → C

] Move largest disk A → C

- Move disk 1 B → A
- Move disk 2 B → C
- Move disk 1 A → C

} Moving N-1 disks from B to C (using A)

N=4



① Move disk 1 A → B

② Move disk 2 A → C

③ Move disk 1 B → C

④ Move disk 3 A → B

⑤ Move disk 1 C → A

⑥ Move disk 2 C → B

⑦ Move disk 1 A → B

Move N-1 disks from A → B (using C)

⑧ Move disk 4 A → C

Move N^{th} disk from A → C

⑨ Move disk 1 B → C

⑩ Move disk 2 B → A

⑪ Move disk 1 C → A

⑫ Move disk 3 B → C

⑬ Move disk 1 A → B

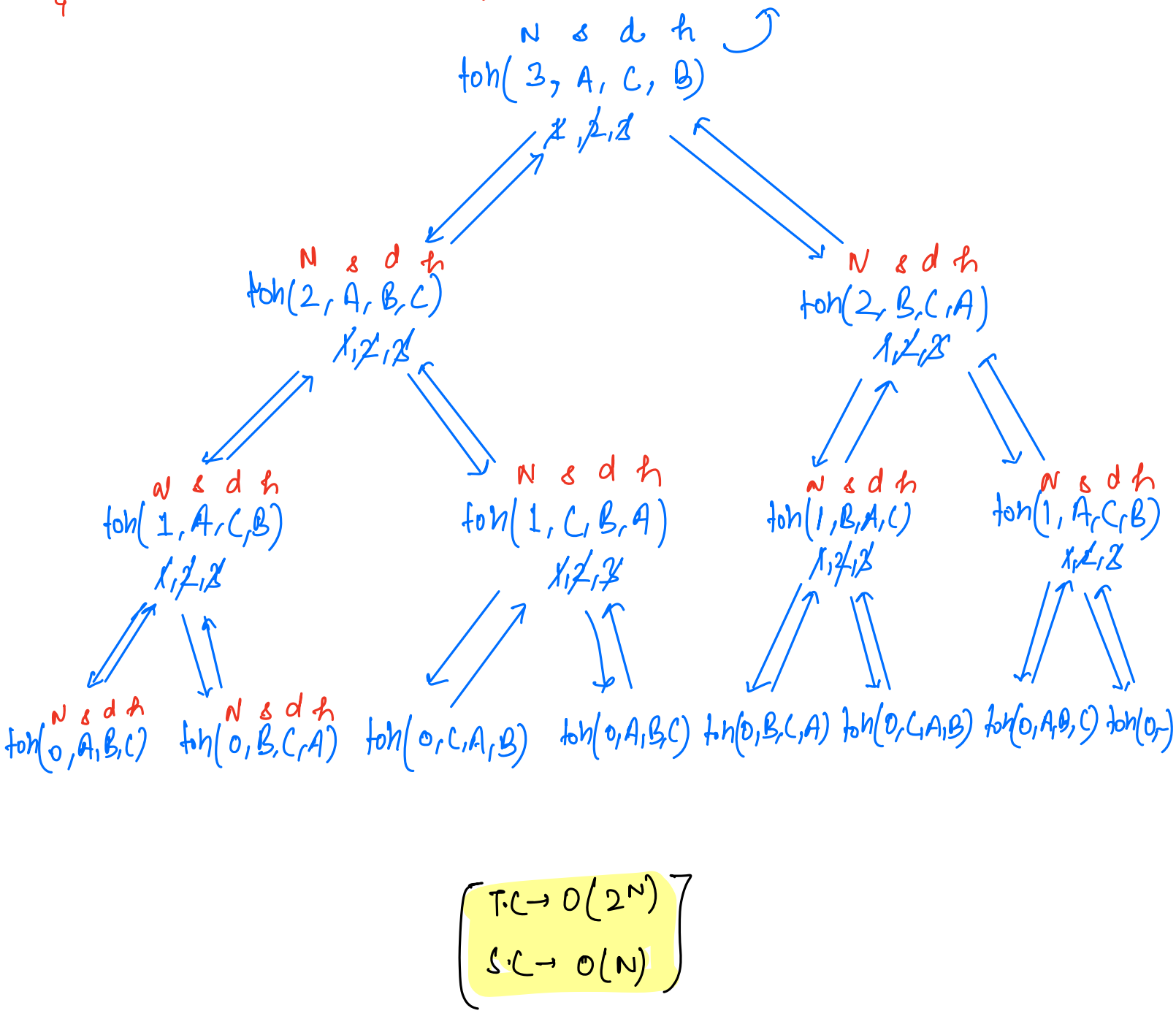
⑭ Move disk 2 A → C

⑮ Move disk 1 B → C

Move (N-1) disks from B to C [using A]

A. C. B.

```
void toh( N, char s, char d, char h){
    if (N==0) { return }
    toh(N-1, s, h, d);  ①
    print("Move Nth disk s→d");  ②
    toh(N-1, h, d, s);  ③
}
```



T.C → $O(2^N)$

S.C → $O(N)$

Move disk 1 A → C

Move disk 2 A → B

Move disk 1 C → B

Move disk 3 A → C

Move disk 1 B → A

Move disk 2 B → C

Move disk 1 A → C

Move disk 1 A → C

Move disk 2 A → B

Move disk 1 C → B

Move disk 3 A → C

Move disk 1 B → A

Move disk 2 B → C

Move disk 1 A → C

{ N = 4. (do dry-run yourself) }

Q Given an integer N. Print all valid parenthesis of length 2N.

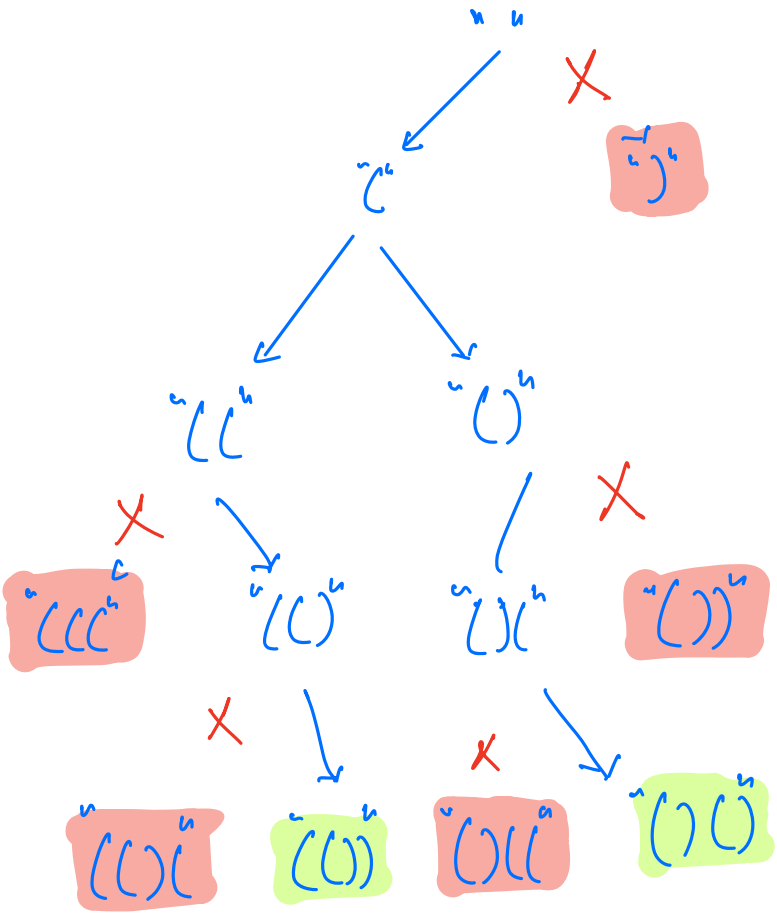
At any point:
no. of closing brackets ≤ no. of opening brackets.

N=1. "()"

N=2. "()()", "(())"

N=3. "((()))", "(()()", "(())()", "()(())", "()()()"

N=2.



```
void solve( N, opening, closing, str){
    if( len of str == 2*N ){ print(str), return}
    if(opening < N){
        solve(N, opening+1, closing, str + "(");
    }
    if(closing < opening){
        solve( N, opening, closing+1, str + ")");
    }
}
```

T.C → $O(2^N)$
S.C → $O(N)$

[dry run] for N=3.

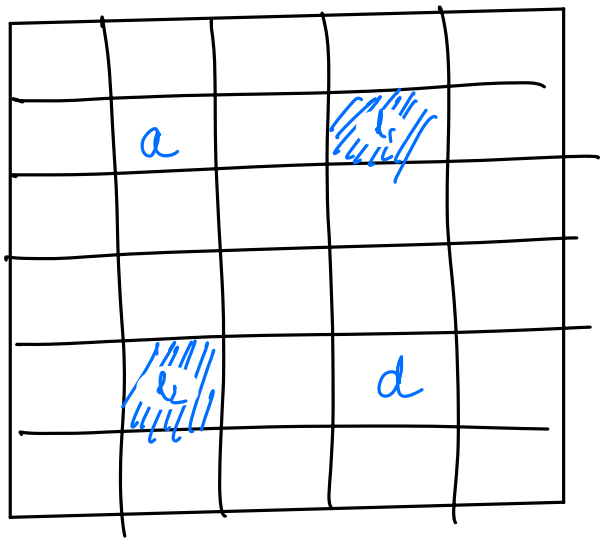
Send s.s of dry run in whatsapp group.

mat[N][M] - find lucky no.

⇓

Smallest in row & largest in column

① How many lucky nos are possible.



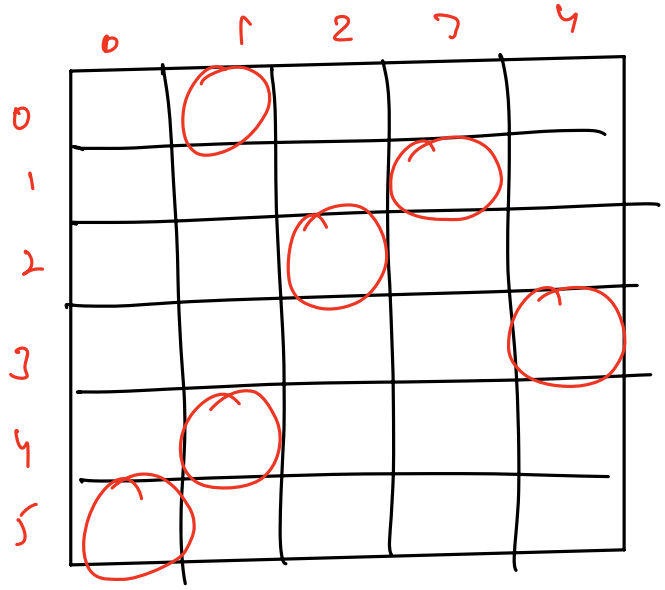
① $l_1 < a$ and $l_1 > d$ $\Rightarrow d < l_1 < a$.

⇓

$d < a$.

② $l_2 < d$ and $l_2 > a$

∴ l_2 can't be lucky no.



Consider minimum of every row & check if it is really lucky no or not

[iterate on column & check if it is maximum or not]