RECURSION

Content

- Recursion Intro
- Sum of natural numbers.
- Fibonacci series
- Output based questions.
- Generate parenthesis
- Tower of Hanoi

Recursion function calling itself.

→ Solving problem using smaller / sub problems.

Q> Sum of first n natural numbers.

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

$$f(4) = 1+2+3+4$$
$$= f(3) + 4$$

$$f(n) = f(n-1) + n$$

Steps for recursive code

1) **Expectation** — what is your code expected to do

2) **Logic** — Solving the expectation

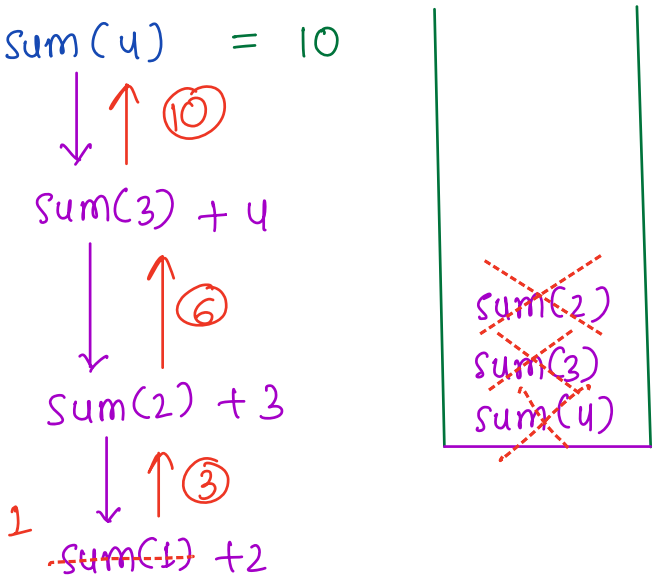
3) **Base Case** → when we have to stop
   → which subproblem you know the answer of.

Pseudocode

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

    return sum(N-1) + N
}
```

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

Sum(4) = 10



Time Complexity

$$f(n) = f(n-1) + O(1)$$

$$f(n-1) = f(n-2) + O(1)$$

$$\vdots$$

$$f(1) + O(1)$$

$O(1)$

$(n-1)$

TC $f(n) = O(n-1) = O(n)$

SC $O(n)$

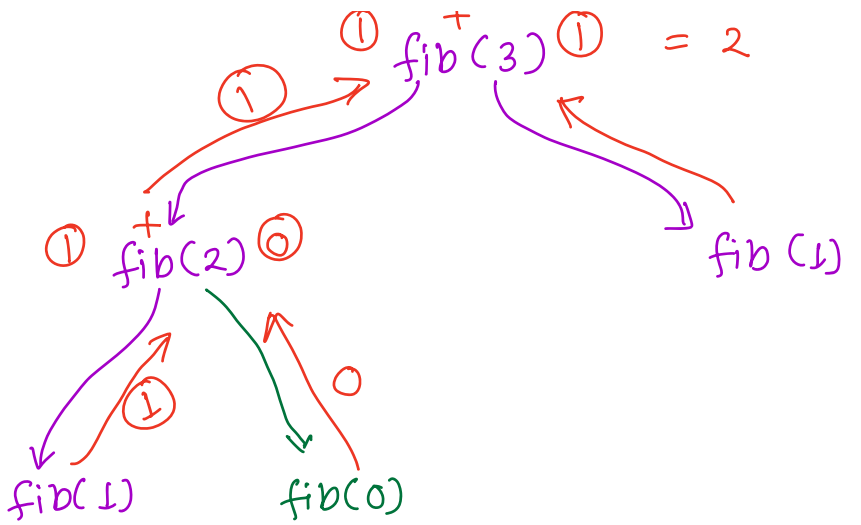
Fibonacci series.

sum of prev two values.

input	0	1	2	3	4	5
output	0	1	1	2	3	5

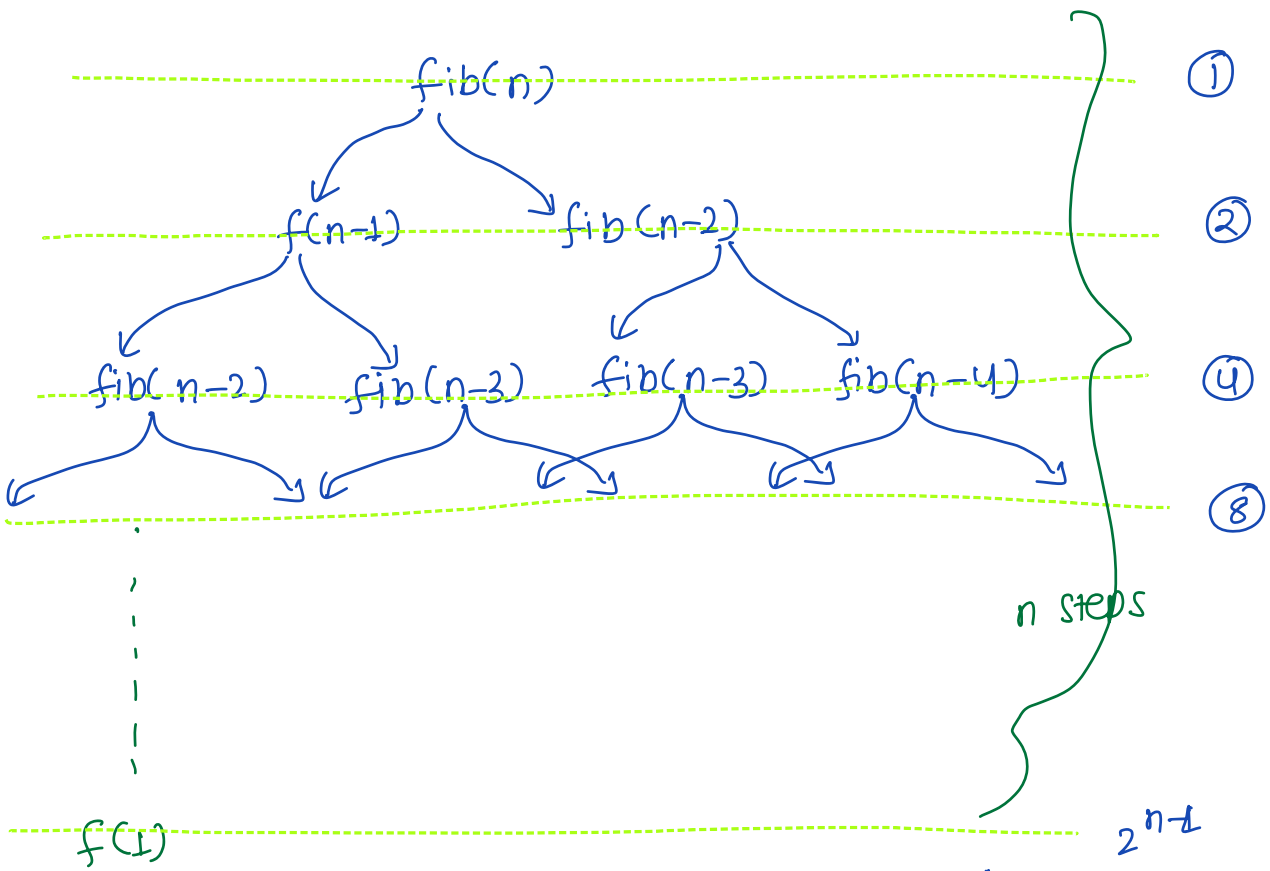
$$fib(2) = fib(1) + fib(0)$$
$$fib(3) = fib(2) + fib(1)$$
$$\vdots$$
$$fib(n) = fib(n-1) + fib(n-2)$$

```
int fib(int N) {
    if (N <= 1) return N

    a = fib(N-1)
    b = fib(N-2)
    return a + b
}
```




```
int fib(int N) {
    if (N <= 1) return N

    a = fib(N-1)
    b = fib(N-2)
    return a + b
}
```

TC



$$TC = 1 + 2 + 4 + 8 + \dots + 2^{n-1}$$

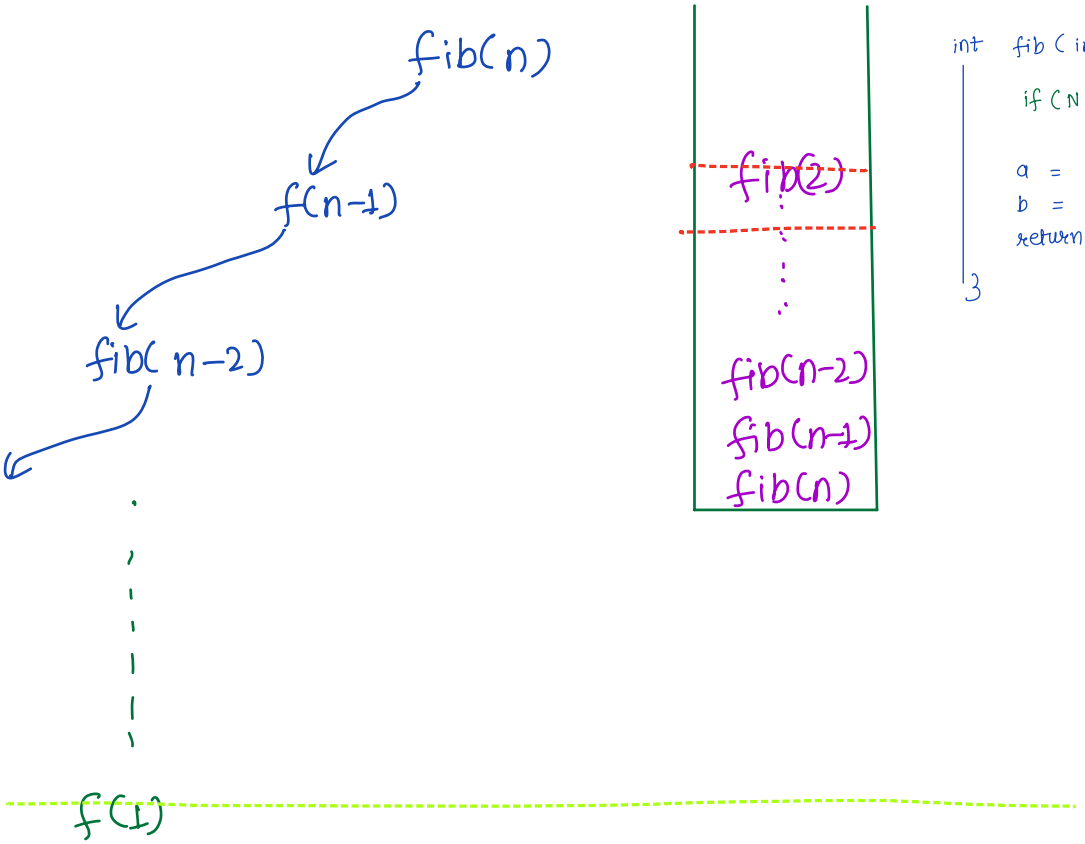
$$= 2^0 + 2^1 + 2^2 + 2^3 \dots + 2^{n-1}$$

$$= \frac{a(r^n - 1)}{r - 1} = \frac{1 \times (2^n - 1)}{2 - 1} = 2^n - 1$$

$O(2^n)$

Space Complexity

max space used at any instant of time by our code



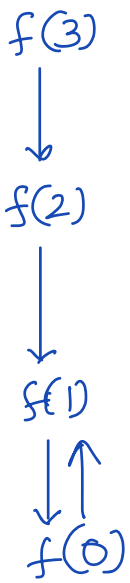
SC = O(N) since the call stack never grows beyond O(n) at any instant.

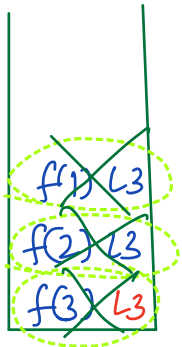
Output Based Quiz

Q1>

```
                    3
        void  solve (n) {
L1          if (n==0) return
L2          solve (n-1)
L3          print (n)
        }
```

1 2 3

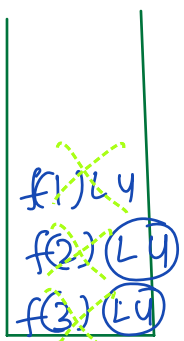




Q2>

```
                    3
        void  solve (n) {
1           if (n==0) return
2           print (n)
3           solve (n-1)
4       }
```

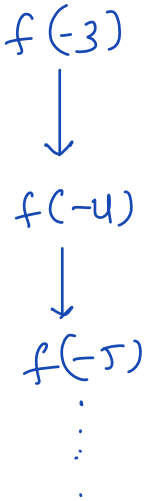
3 — 2 — 1



Q3>

```
void solve (n) {          // n = -3
1   if (n==0) return
2   print (n)
3   solve (n-1)
}
```

Stack overflow



Q4>

```
int solve (n) {
    if (n==0) return 0
    return (solve(n/10) + n%10)
}
```

TODO

Q> Print all valid parenthesis of length 2*N
Given N

N = 1 → ((())( ))

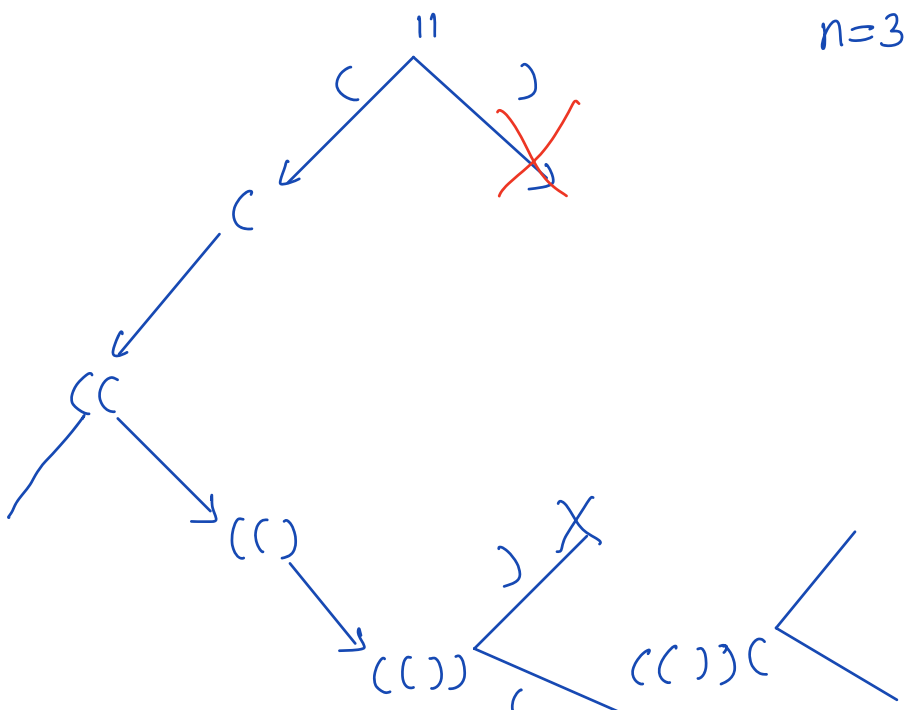
N = 2 → (()) ()()

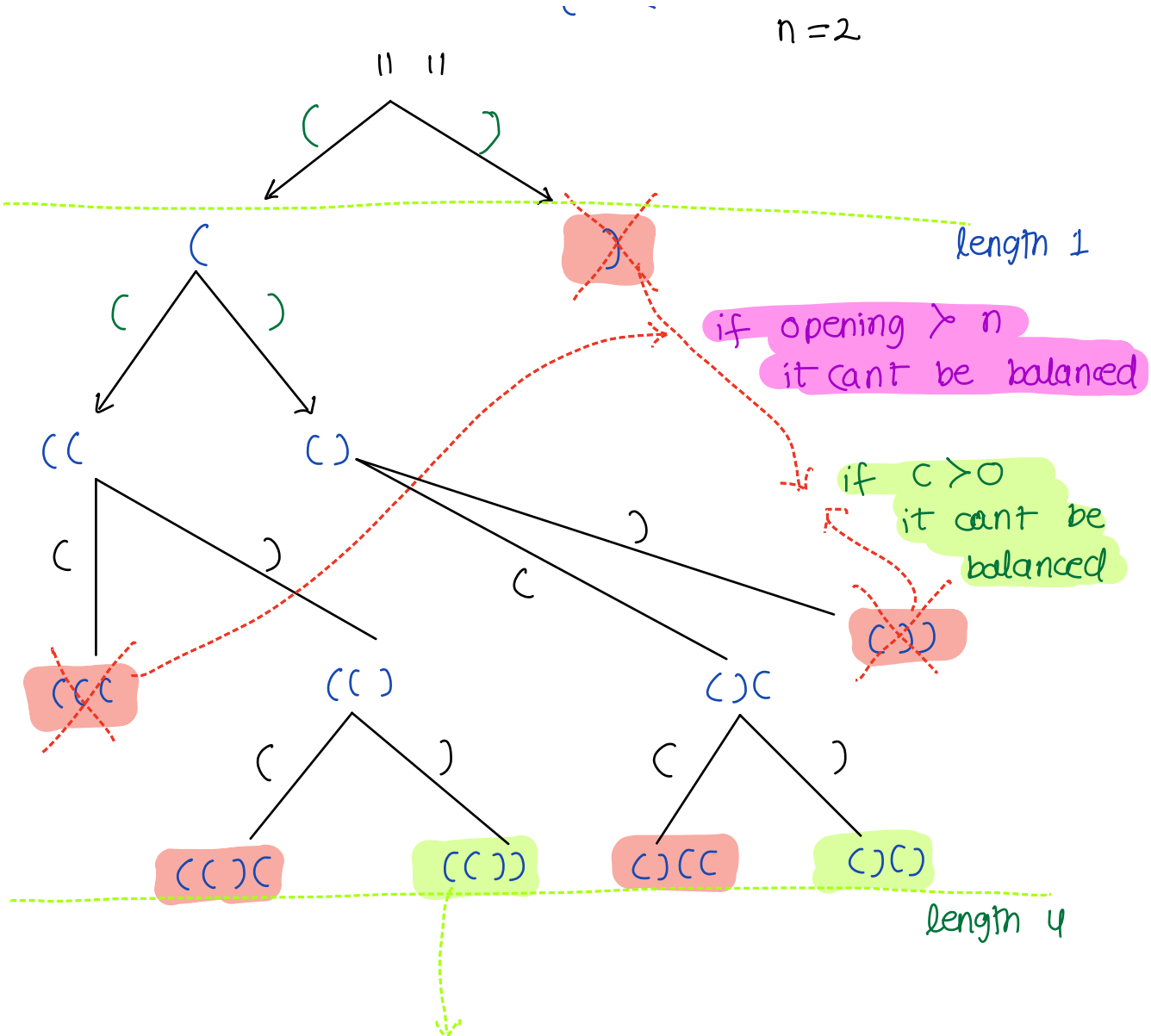
N = 3 → ((())) (()()) ()(()) (())() ()()()

---

idea : Generate all the possible parenthesis
check if the are valid while generating.

Doubt



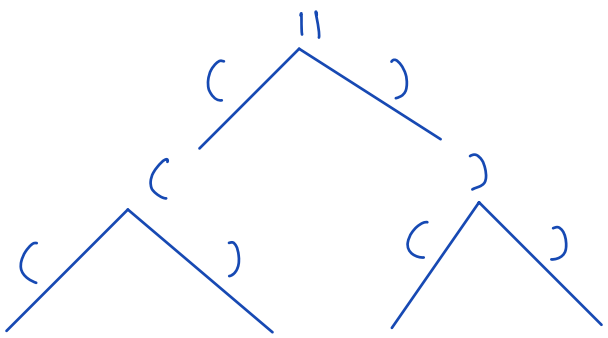

n = 2
length 1
if opening > n
it cant be balanced
if C > O
it cant be
balanced
length 4
At last O == C its balanced.

```
void p(steing s, int o, int c, int N) {
    // Base
    if(o>N) return
    if(c>o) return
    if(o==N && c==N) {
        print(s)
    }
    // Opening bracket
    p(s+"(", o+1, c, N)
    // Closing bracket
    p(s+")", o, c+1, N)
}
```

TC : $O(2^n)$

SC : $O(n)$

10:25



Tower of Hanoi

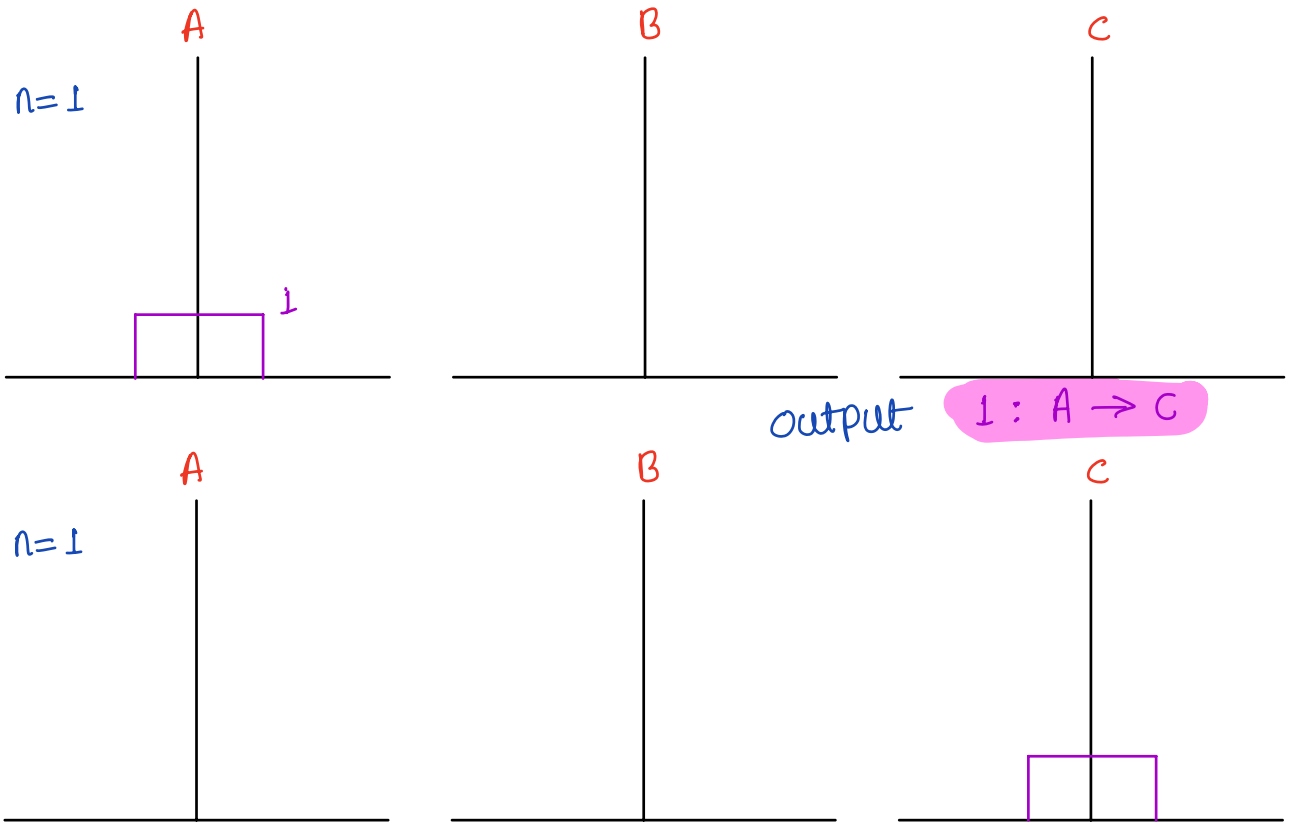
There are N disks placed on tower A of different sizes. Goal is to move all disks from tower A to C using tower B if needed.

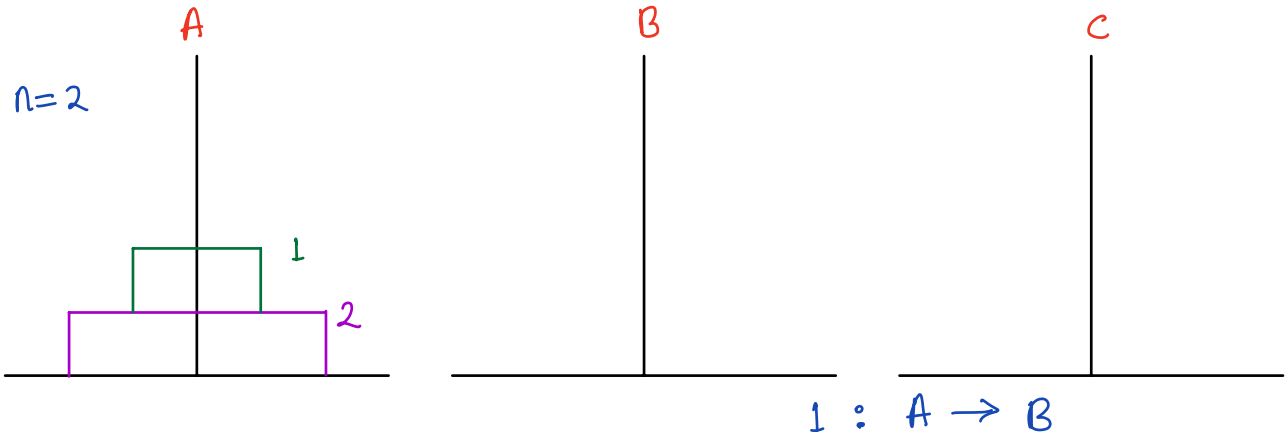
Constraint

01> Only 1 disk can be moved at a time.

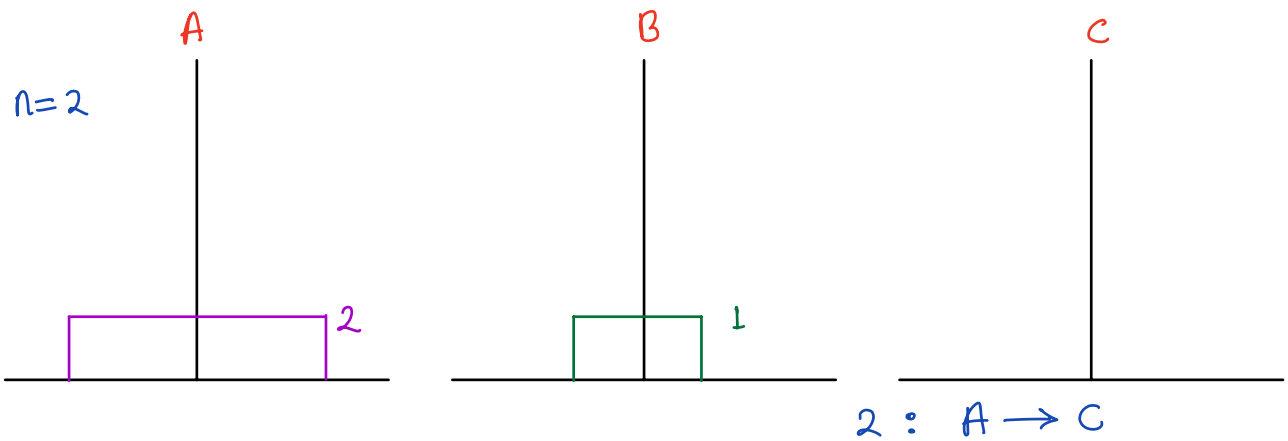
02> Larger disks cannot be placed on smaller disks at any step.

Print the movement of discs from A to C in min steps.

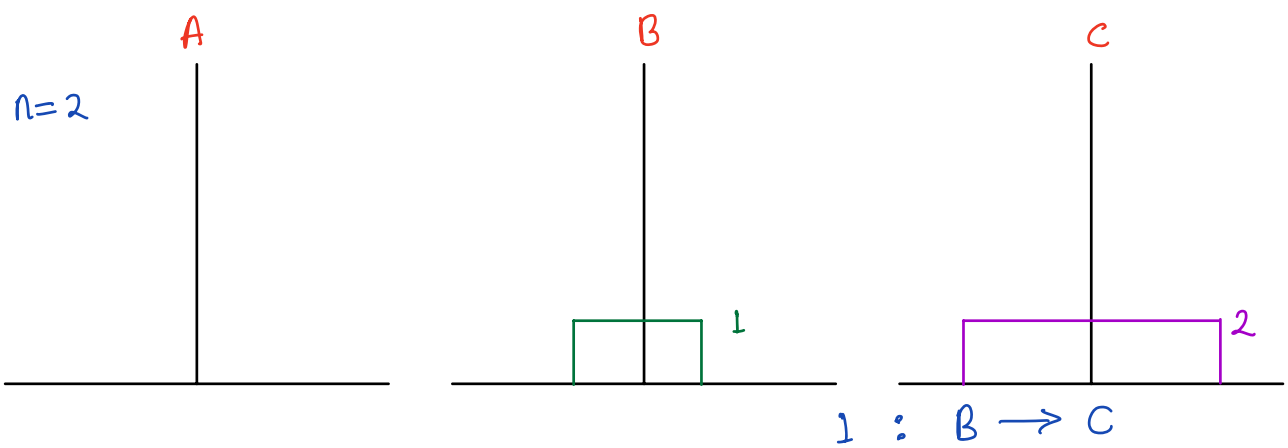




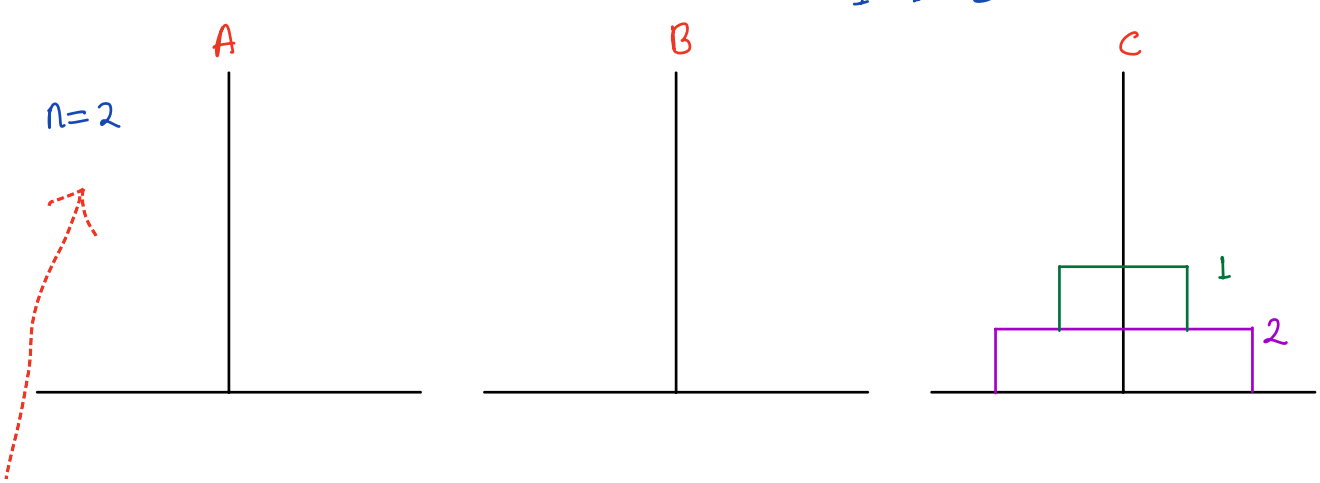
1 : A → B



2 : A → C



1 : B → C

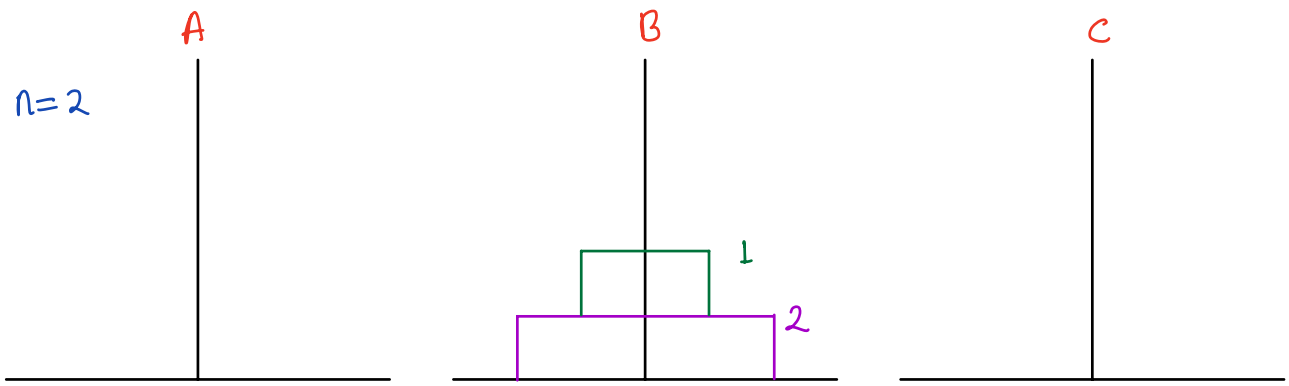


TOH = Tower of Hanoi

TOH(N , A , B , C)

TOH (2 , A , B , C)

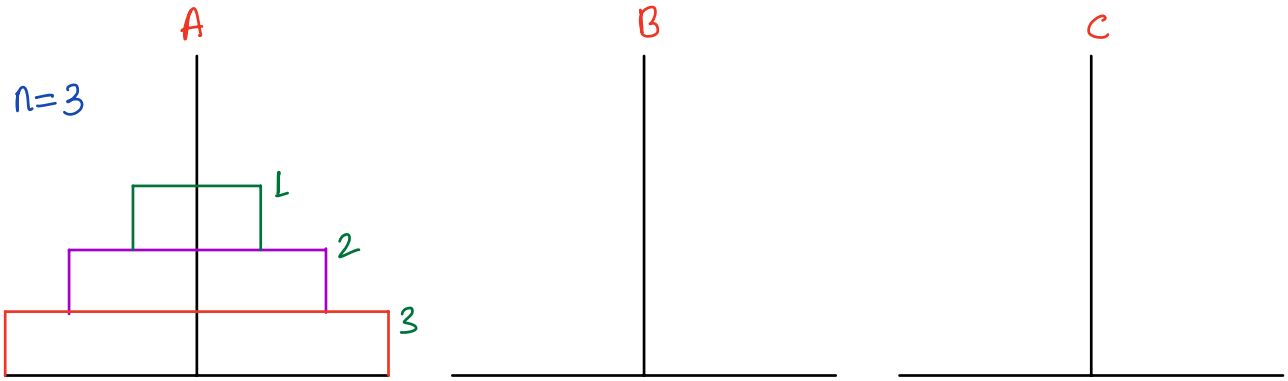
Move from A to B using C



1 : A to C
2 : A to B
1 : C to B

S H t

TOH (3, A, B, C)

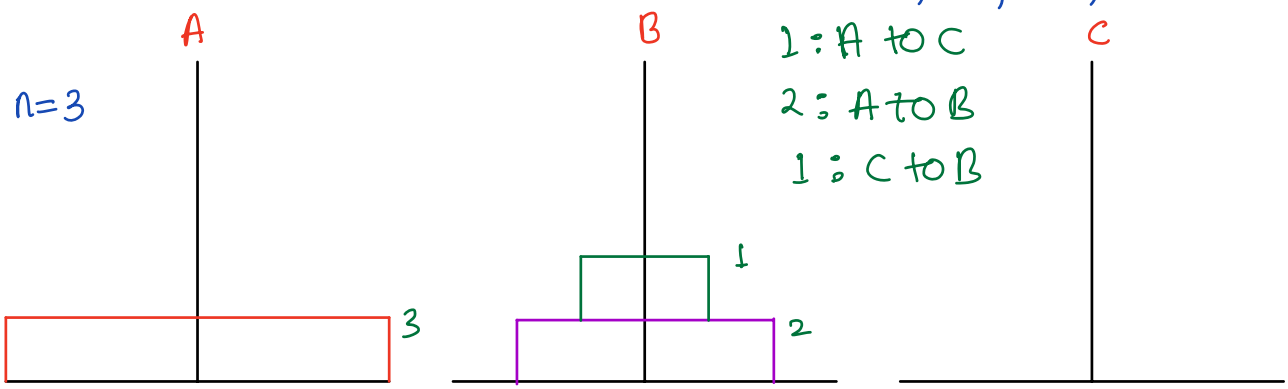


TOH (2, A, C, B)

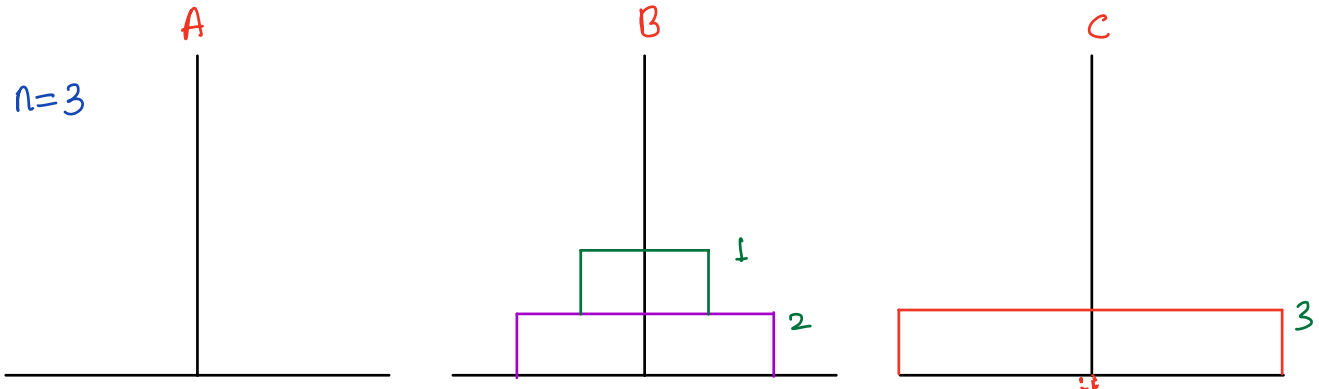
1 : A to C

2 : A to B

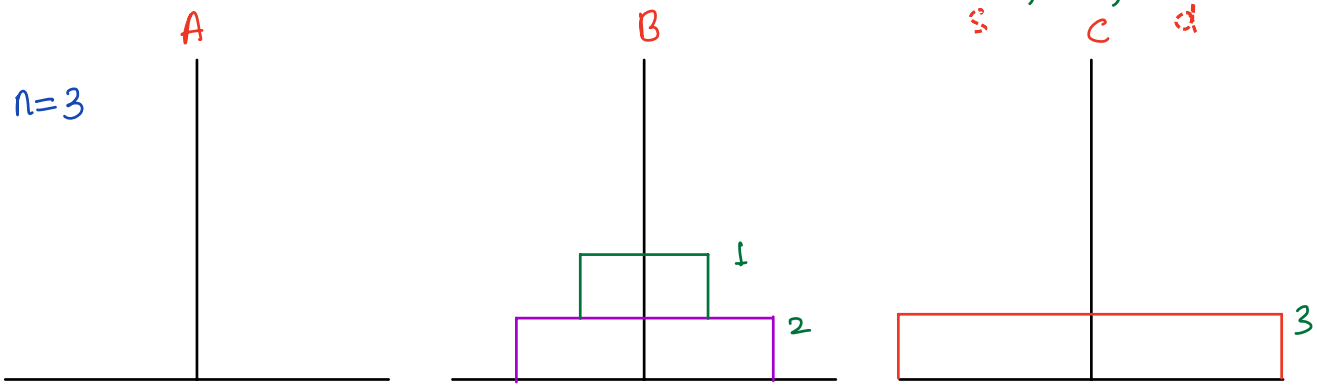
1 : C to B



print (3, A to C)



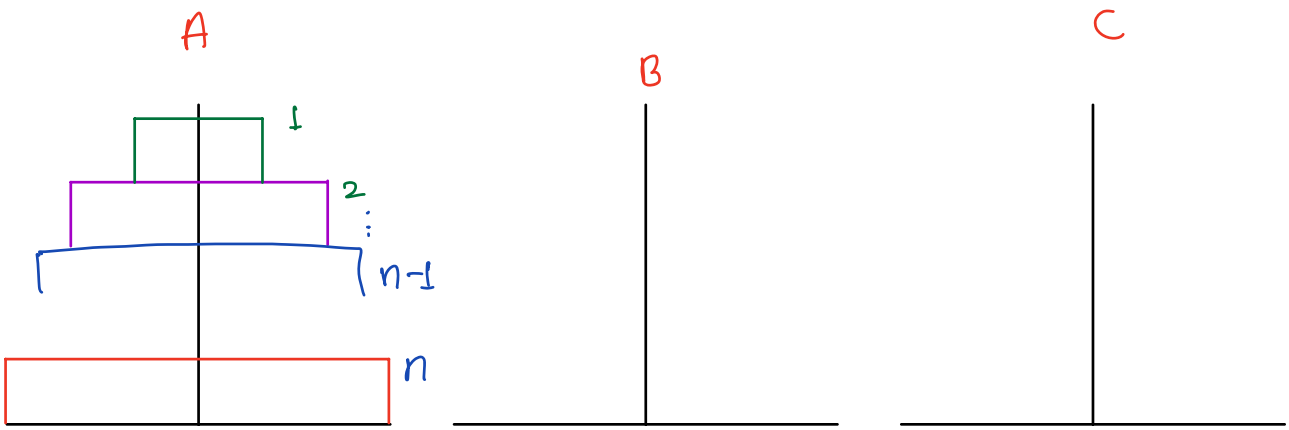
toh(2, B, A, C)



s H t

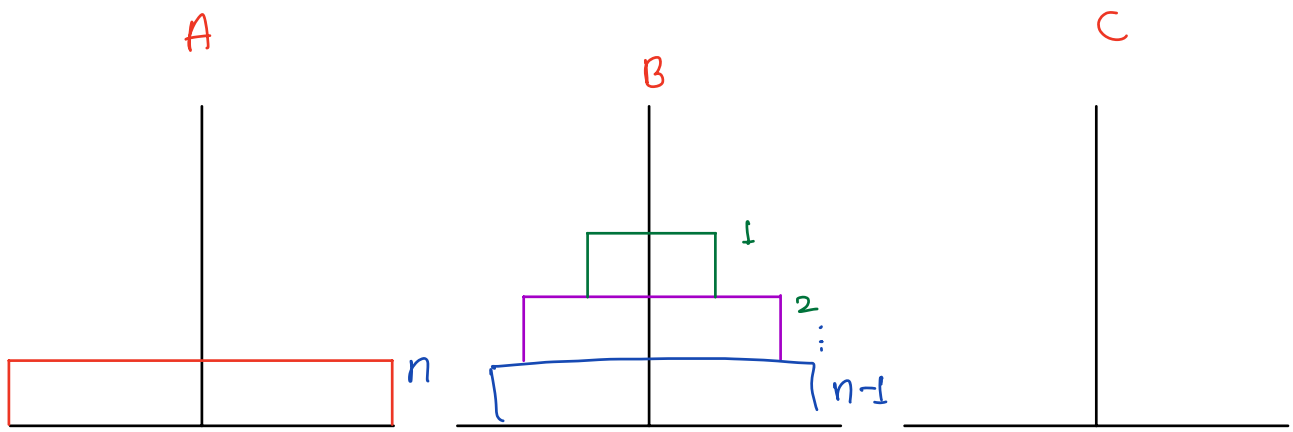
TOH (3, A, B, C)

TOH (2, A, C, B)
print (3, A to C)
toh(2, B, A, C)

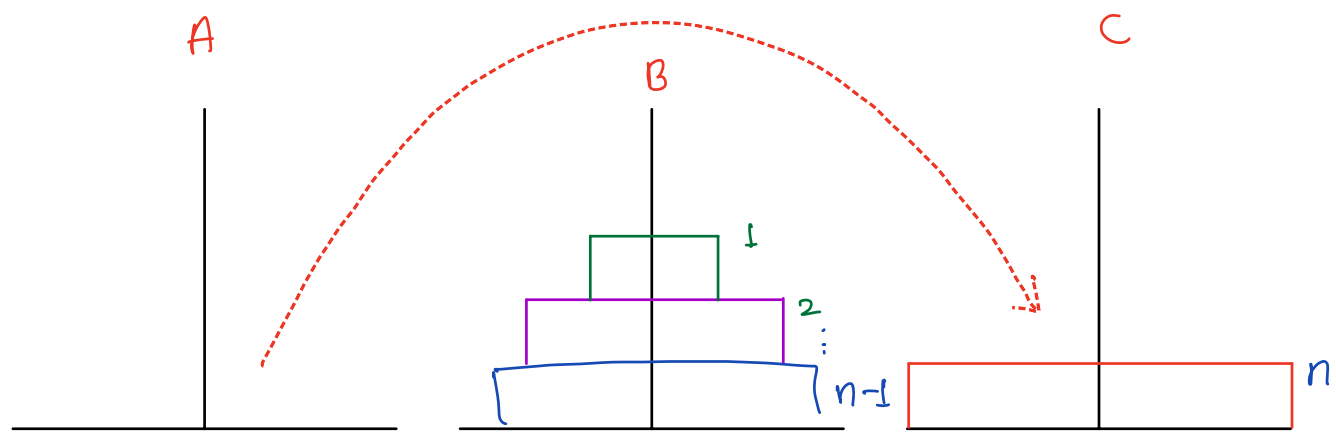


TOH (N, A, B, C)
Move N disks from source A
dest C
Helper or wing B

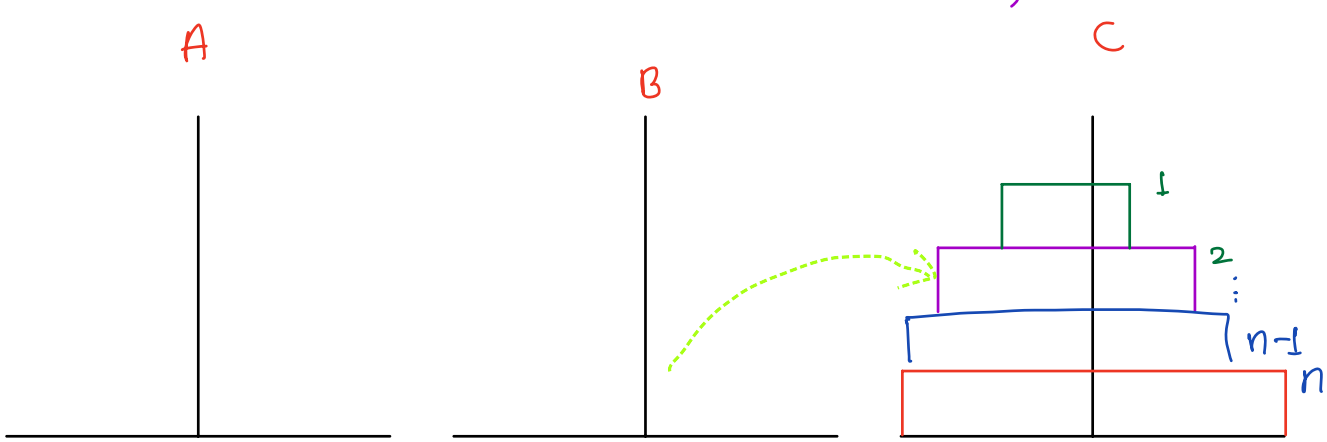
TOH (n-1, A , C , B)



print (n, A to C)



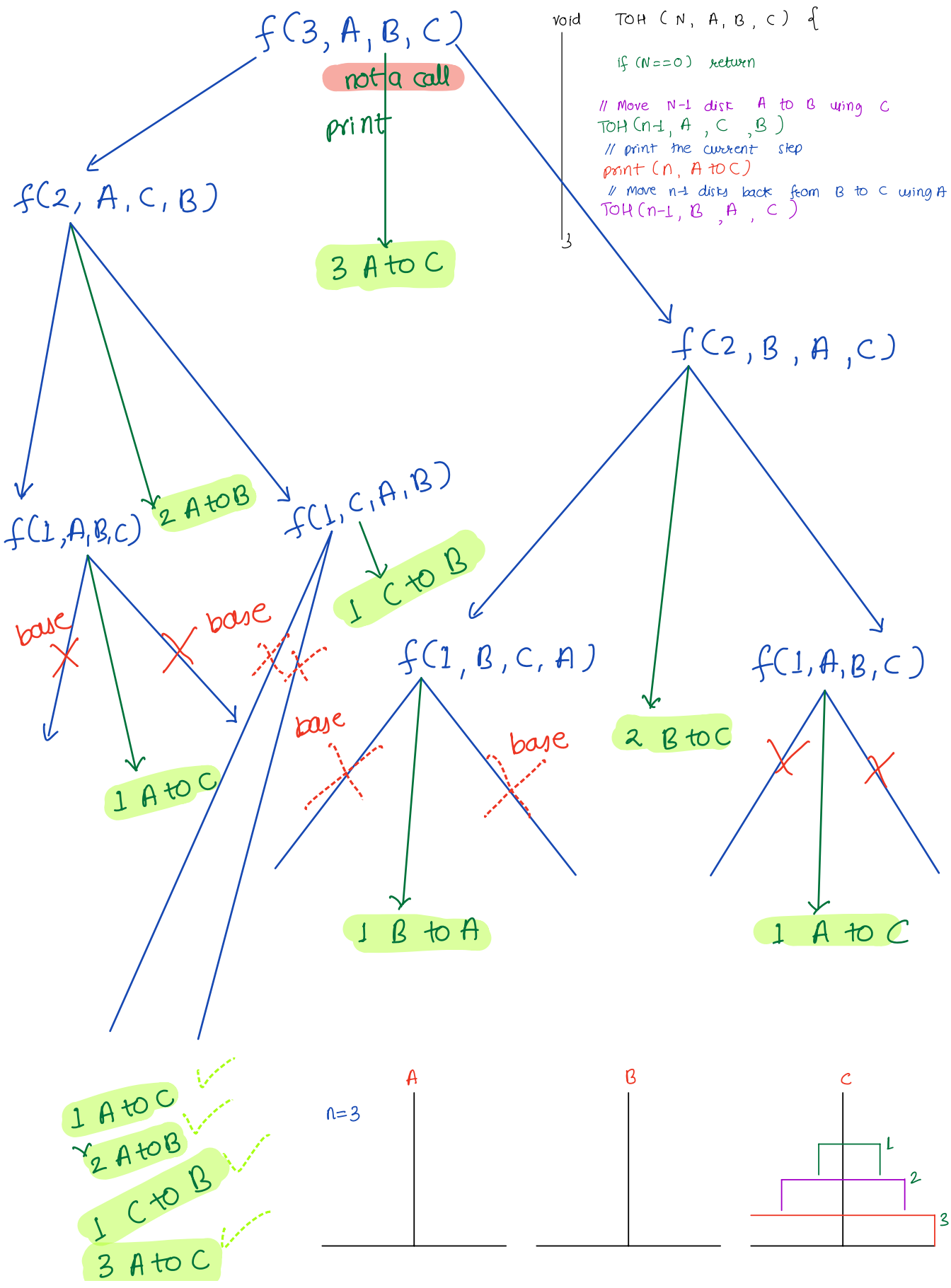
TOH (n-1, B , A , C)



```
void TOH (N, A, B, C) {
                   source  helper  dest
    if (N==0) return

    // Move N-1 disk A to B using C
    TOH(n-1, A, C, B)
    // print the current step
    print(n, A to C)
    // Move n-1 disks back from B to C using A
    TOH(n-1, B, A, C)
}
```

TC : $O(2^n)$

SC : $O(N)$


f(3, A, B, C)
not a call
print
3 A to C
void TOH (N, A, B, C) {
if (N==0) return
// Move N-1 disk A to B using C
TOH (n-1, A , C , B)
// print the current step
print (n, A to C)
// Move n-1 disks back from B to C using A
TOH (n-1, B , A , C)
}
f(2, A, C, B)
f(2, B, A, C)
f(1,A,B,C)
2 A to B
f(1, C, A, B)
1 C to B
base
base
1 A to C
f(1, B, C, A)
base
base
1 B to A
2 B to C
f(1, A, B, C)
1 A to C
1 A to C
2 A to B
1 C to B
3 A to C
n=3
A
B
C
1
2
3

1 B to A

2 B to C

1 A to C

Doubt

```
void p(string s, int o, int c, int N) {
    // Base
    if(o>N) return
    if(c>o) return
    if(o==N && c==N) {
        print(s)
    }
    // Opening bracket
    p(s+"(", o+1, c, N)
    // Closing bracket
    p(s+")", o, c+1, N)
}
```