Recursion & Backtracking

Principle of Mathematical Induction (PMI)

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

① Trivial Case

$$\sum_{i=1}^{0} i = \frac{0(0+1)}{2} = 0$$

$$\sum_{i=1}^{1} i = 1 = \frac{1(1+1)}{2} = 1$$

② Assume $\sum_{i=1}^{k} i = \frac{k(k+1)}{2}$

③ We have to prove for k+1 RHS should be $\frac{(k+1)(k+2)}{2}$

$$\sum_{i=1}^{k+1} i = 1+2+3+\dots+k+(k+1) = \frac{k(k+1)}{2} + (k+1) = \frac{(k+1)(k+2)}{2}$$

Since LHS = RHS for n=1 & our assumption holds true, by PMI $\boxed{\sum_{i=1}^{n} i = \frac{n(n+1)}{2}}$

Recursion (Example of Printing)

① Recursive function: expectation
void printInc(int n); → 1, 2, 3, ... n

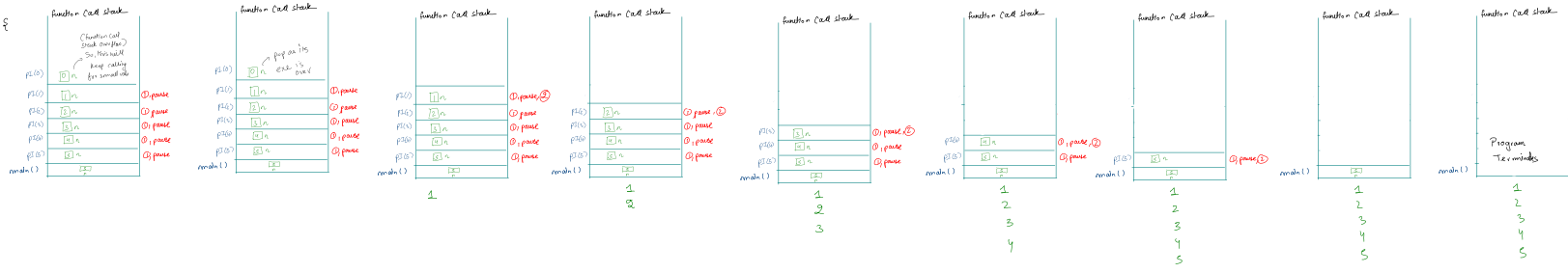
② Recursive function: smaller values :→ faith/call
void printInc(n-1); → 1, 2, 3, ... (n-1)

③ Meeting Expectation with Faith
Sysо(n);

```
Void printInc (int n) {
if (n == 0) return;

① printInc(n-1);
② Sysо(n);
}

3
```

```
public static void printInc(int n){
    if(n == 0) return;
    printInc(n-1);
    System.out.println(n);
}
```

Low-Level Thinking



Que Print Decreasing

Example

Sample Input

5

Sample Output

5
4
3
2
1

① Expectation:

printDec(n) : n, n-1, n-2 ... 1

② Faith:

printDec(n-1): n-1, n-2, n-3 ... 1

```
public static void printDecreasing(int n){

    if(n == 0) return;

    System.out.println(n); → Pre order
    printDecreasing(n-1);
}
```

③ Meet Expectation with faith:

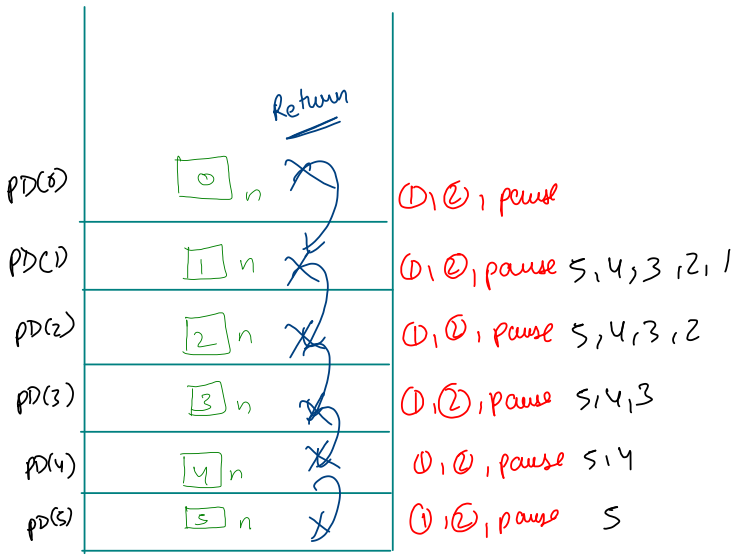
```
Syso(n);
printDec(n-1);
```

acc to HLT

```
void printDec(int n)
{
① Syso(n);
② printDec(n-1);
}
```

Head Rec

work is done before calling functions



Ques Print Decreasing Increasing

Example

Sample Input

3

Sample Output

3
2
1
1
2
3

① Expectation:

pDecInc(n) →

② Faith:

3 → 3, 2, 1, 1, 2, 3

2 → 2, 1, 1, 2

pDecInc(n-1) → n-1, n-2 .. 1, 1, 2.. (n-1)

n
n-1
n-2
:
1
1
2
:
n-1
n

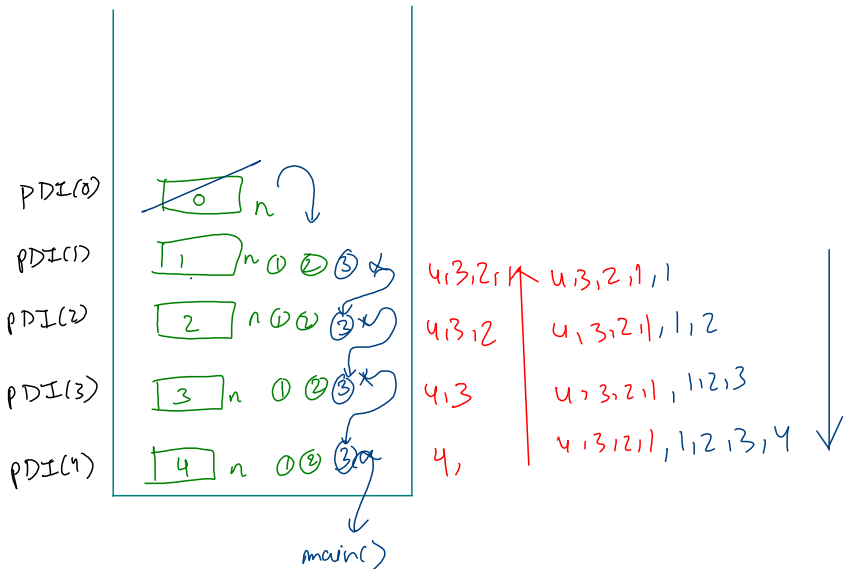
③ Meet Expectation with faith:

Print n

call (n-1)

Print n

```
public static void pdi(int n){

   if(n == 0) return;

   System.out.println(n); -> Pre
   pdi(n-1);
   System.out.println(n); -> Post
}
```



Ques) factorial

Example

Sample Input

5

Sample Output

120

Expectation: fact(n) → n x n-1 x n-2 x n-3 ... 1

Faith: fact(n-1) → n-1 x n-2 x n-3 ... 1

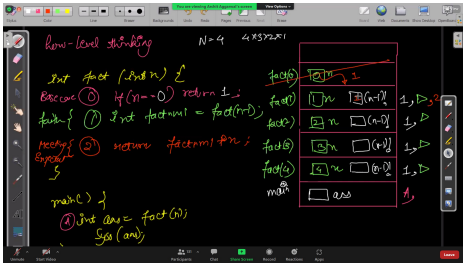
} High Level Thinking

Meet Expectation with faith:

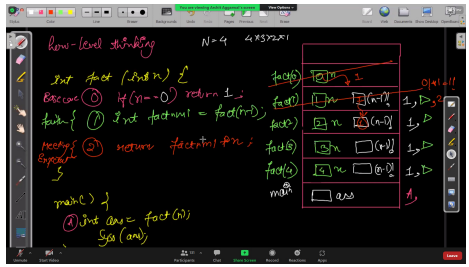
fact(n) = n * fact(n-1)

```
public static int factorial(int n){

    if(n == 0) {
        return 1;
    }

    return n*factorial(n-1);
}
```

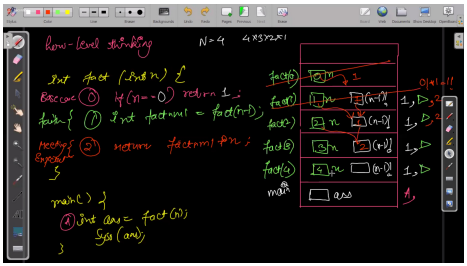
1



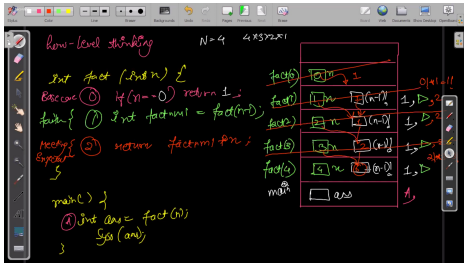
2



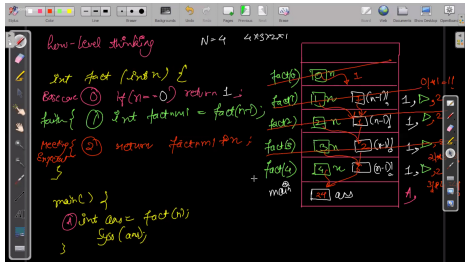
3



4



5



Qus) Power Linear $\Rightarrow O(n)$

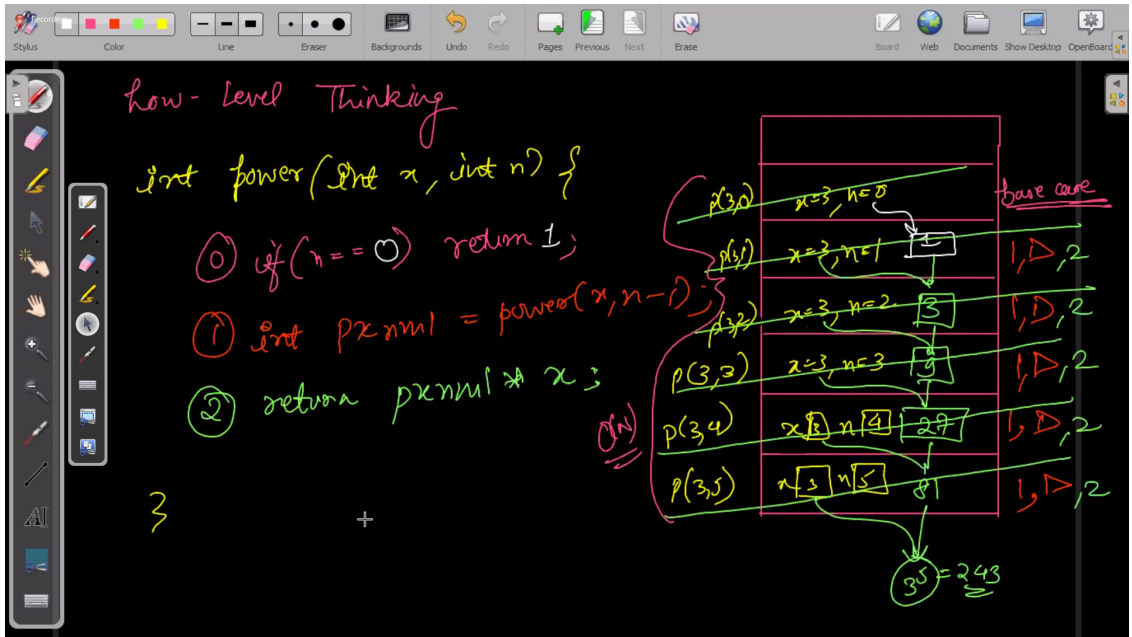
$a^b \rightarrow a * a * a \ldots$ b times

```
public static int power(int x, int n){
    if(n == 0) return 1;
    return x*power(x,n-1);
}
```

① Expectation: power(x, n) → x * x * .... n times

② faith: power(x, n-1) → x * x * x ... (n-1) times

③ Meeting Expectation with faith: x * power(x, n-1)



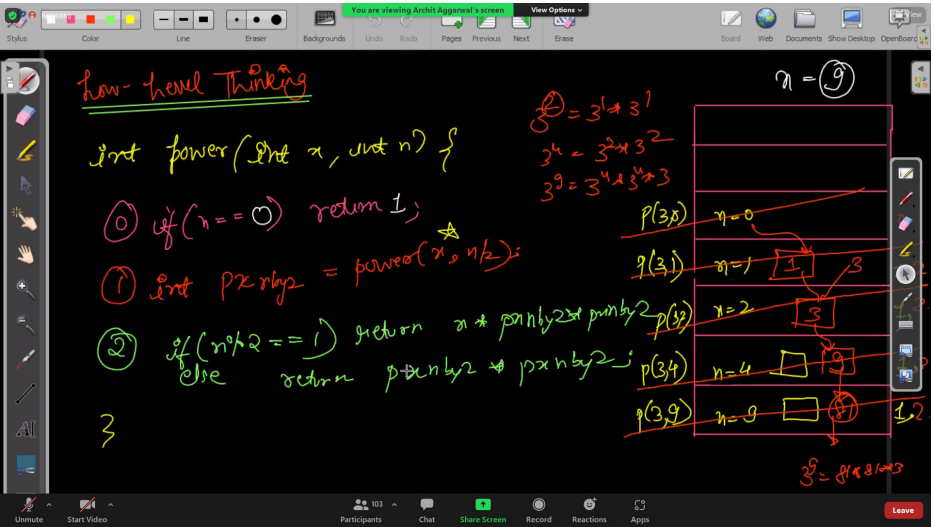
Ques) Power Logarithmic

Expectation: power(x, n) → x * x * x ... n times

faith: x * x * x ... n/2 times

Meeting Expectation with faith: $x^{n/2} * x^{n/2}$

```
if (n%2 == 0)
  return power(x, n/2) * power(x, n/2)
else
  return x * power(x, n/2) * power(x, n/2)
```



$n \to 2^x \Rightarrow x$ no of calls

$\log_2 n = x$

$O(\log_2 n)$

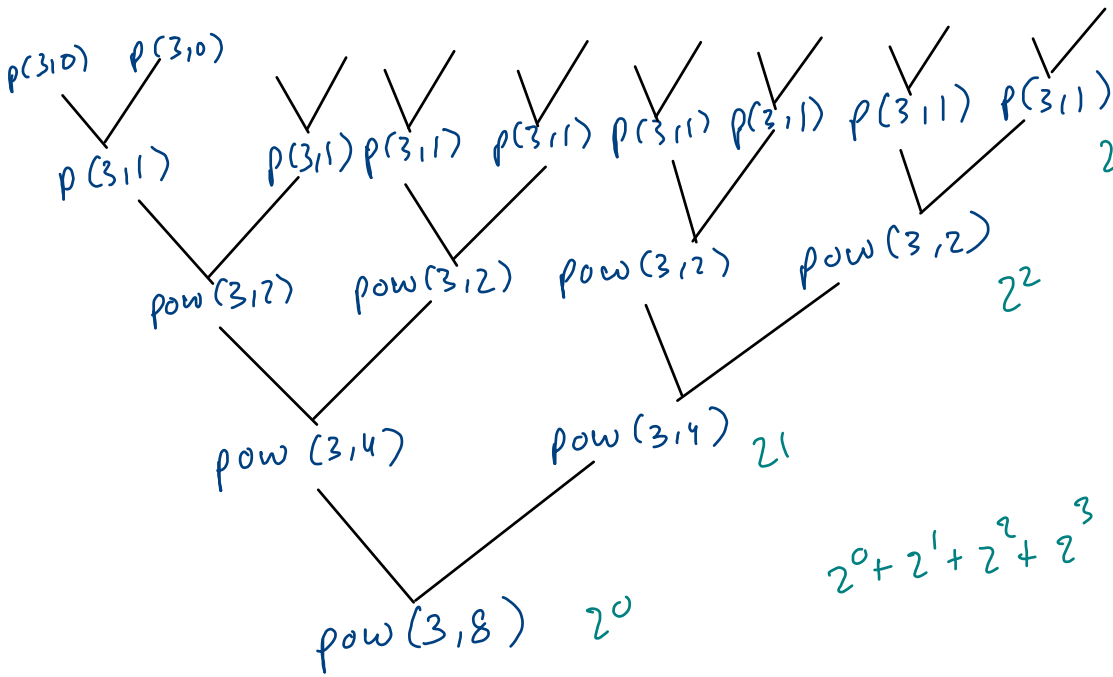
if we do

```
int xpn = power(x, n/2) * power(x, n/2)
```

2 calls

Qus What will be the complexity?

Recursion Tree



→ Even if we ignore base calls

2^3

2^2

21

2^0

$2^0 + 2^1 + 2^2 + 2^3 = \frac{2^4 - 1}{2 - 1} = 2^4$

$\approx (8) \rightarrow n$

Again $O(n)$ {or even worse}

Ques) Print Zig Zag

Sample Input

3

Sample Output

321112111232111211123

① Expectation : pzz(int n)

321112111232111211123

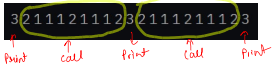
② Faith:

321112111232111211123

pzz(n-1)

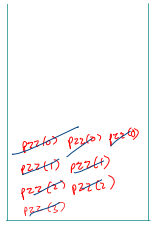
3 → 21112111 2

③ Meet Expectation with faith



* 2 times faith is used.

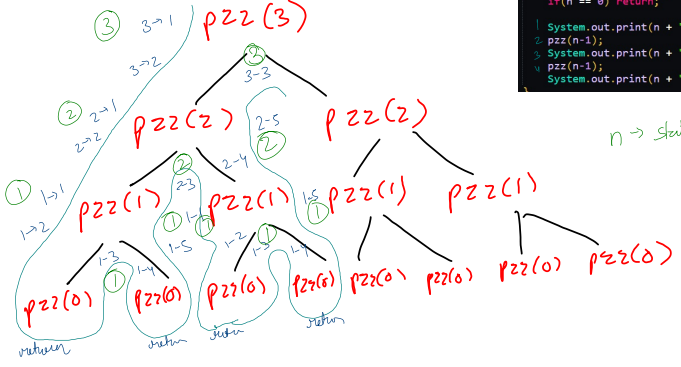
```
public static void pzz(int n){
    if(n == 0) return;

    System.out.print(n + " ");
    pzz(n-1);
    System.out.print(n + " ");
    pzz(n-1);
    System.out.print(n + " ");
}
```



3 2 1 1 2 1 1 1 2 3 2 1 1 1 2 3

```
public static void pzz(int n){
    if(n == 0) return;

1   System.out.print(n + " ");
2   pzz(n-1);
3   System.out.print(n + " ");
4   pzz(n-1);
5   System.out.print(n + " ");
}
```

n → state no



3→1 3→2 2→1 2→2 1→1 1→2 0→r 1→3 1→4 or 1→5 2→3 1→1 1→2 or 1→3 1→4 or 1→5 2→5 3→3

↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓

3 2 1 1 1 2 1 1 2 3

Illy work for 2nd Half of the Tree.

Recursion (Time Complexity formula)

$$(calls)^{height} + \{preorder + inorder + postorder\} * height$$

↑ from any node

for f22

$$(2)^n + \{k + k + k\} * n$$

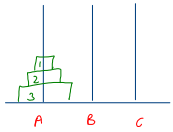
$$= 2^n + 3kn$$

$$O(2^n)$$

Tower of Hanoi

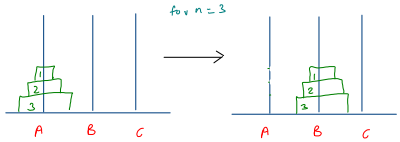
1. There are 3 towers. Tower 1 has n disks, where n is a positive number. Tower 2 and 3 are empty.
2. The disks are increasingly placed in terms of size such that the smallest disk is on top and largest disk is at bottom.
3. You are required to
   3.1. Print the instructions to move the disks.
   3.2. from tower 1 to tower 2 using tower 3
   3.3. following the rules
      3.3.1 move 1 disk at a time.
      3.3.2 never place a smaller disk under a larger disk.
      3.3.3 you can only move a disk at the top.

Note -> The online judge can't force you to write the function recursively but that is what the spirit of question is.Write recursive and not iterative logic. The purpose of the question is to aid learning recursion and not test you.

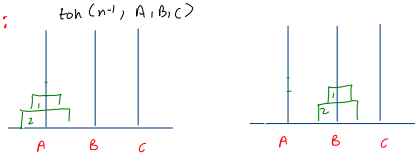


constraints :
↳ Only one disk can be moved at a time
↳ Larger disk cannot be placed on top of smaller disk

① Expectation : toh(n, A, B, C)
(no of disks, src, dest, auxiliary)

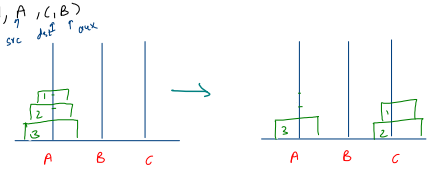


② faith : toh(n-1, A, B, C)

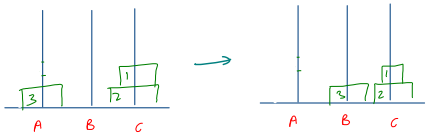


③ Meeting Expectation from faith

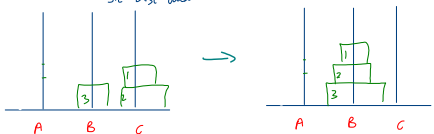
toh(n-1, A, C, B)
src dest aux



nth [A → B]



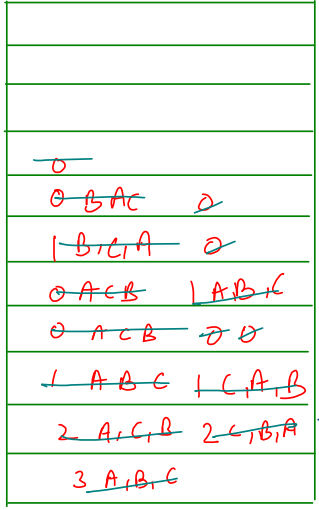
toh(n-1, C, B, A)
src dest aux




```java
public static void toh(int n, int t1id, int t2id, int t3id){

    if(n == 1) {
        System.out.println(n + "[" + t1id + " -> " + t2id + "]");
        return;
    }

    toh(n-1,t1id,t3id,t2id);
    System.out.println(n + "[" + t1id + " -> " + t2id + "]");
    toh(n-1,t3id,t2id,t1id);
}
```

Tower of Hanoi Dry Run

```
public static void toh(int n, int t1id, int t2id, int t3id){

    if(n == 0) {
        return;
    }

    toh(n-1,t1id,t3id,t2id);
    System.out.println(n + "[" + t1id + " -> " + t2id + "]");
    toh(n-1,t3id,t2id,t1id);
}
```



1 ▷ 2 3

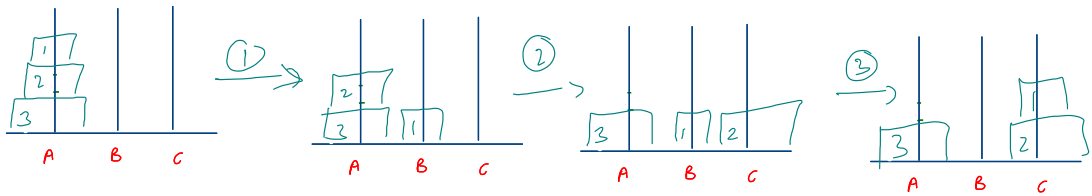
1 ▷ 2 3

1 ▷ 2 3

1 ▷ 2 3 ▷

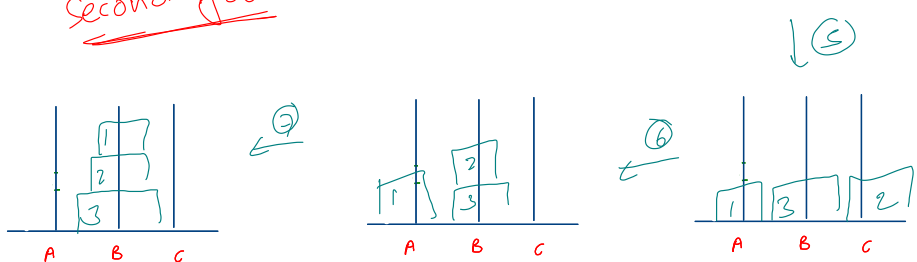
① 1 from A to B
② 2 from A to C
③ 1 from B to C
④ 3 from A to B
⑤ 1 from C to A
⑥ 2 from C to B
⑦ 1 from A to B

first faith is complete



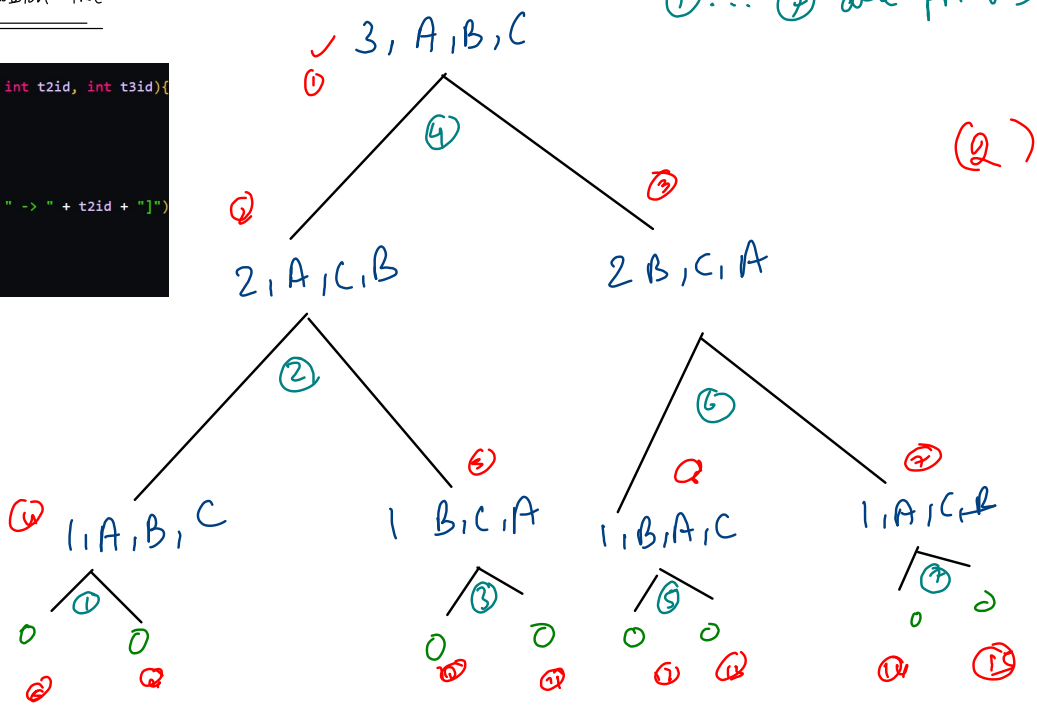
-> own work

Second faith



Tower of Hanoi Recursion Tree

```java
public static void toh(int n, int t1id, int t2id, int t3id){

    if(n == 0) {
        return;
    }

    toh(n-1,t1id,t3id,t2id);
    System.out.println(n + "[" + t1id + " -> " + t2id + "]");
    toh(n-1,t3id,t2id,t1id);
}
```



① ... ⑦ are print statements

$$(2)^{n+1} + (0+k+0)n$$

$$2^{n+1} + kn$$

$$\simeq O(2^n)$$