DAY 44 DSA

20 Oct, 2021

Recursion Continued.....

Agenda for today

1. Print Zigzag → CW
2. Towers of Hanoi → CW

Ques-1. PRINT Zigzag

We are given an integer n and required to print a particular type of sequence of numbers

functionname → Pzz

Eg-1.
Input → 1
Output → 1 1 1
Pzz(0) Pzz(0)

Eg-2
Input → 2
Output → 2 1 1 1 2 1 1 1 2
Pzz(1) Pzz(1)

Eg-3
Input → 3
Output → 3 2 1 1 1 2 1 1 1 2 3 2 1 1 1 2 1 1 1 2 3
Pzz(2) Pzz(2)

Input → 2
Output → 2 1 1 1 2 1 1 1 2

} We need to Build faith that we will receive this Output for n-1

Input → 3
Output → 3 2 1 1 1 2 1 1 1 2 3 2 1 1 1 2 1 1 1 2 3

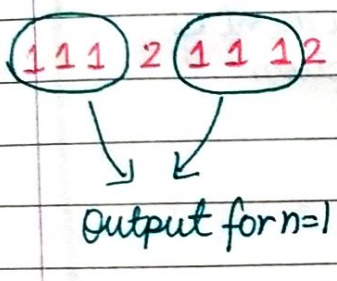
Output for n=2 Output for n=2

Analyzing the output for n=3.

1. Printed 3.
2. Printed the output for n=2
3. Printed 3.
4. Printed the output for n=2.
5. Printed 3.

Page No. | Date

High Level Thinking for n=2.

Expectation	Faith	Expectation meets/ needs faith
for n=2 print : 	for n=1. print 1 1 1 We have a strong faith that our function knows how to print for n=1 i.e n-1.	for n=2. 1) Print 2 (n). 2) Let function pzz print output for n=1 (n-1) 3) Print 2 (n). 4) Let function pzz print the output for n=1 (n-1). 5) Print 2 (n). 2 1 1 1 2 1 1 1 2.

Step 1 Step 2 Step 3 Step 4 Step 5

High Level Thinking

1. → Expectation
   ↳ We expect that our code would the print the correct output for n.
   if n=2
   Output → 2 1 1 1 2 1 1 1 2
   We expect that the correct output will be printed

2. → Faith
   ↳ We form a faith that our code will be able to print the output for n-1.
   हमे नहीं पता कि n-1 का output कैसे print होगा लेकिन हम assume कर लेते है / अपना faith रखते है कि हमे n-1 का Correct Output मिलेगा।
   if n=1
   Output → 111
   We assume that we will receive this output

3. Expectation meets Faith

Page No.
Date

We have 5 steps here.

→ Print n
→ Let function print output for (n-1)
→ Print n
→ Let function print output for (n-1)
→ Print n

return type void है मतलब कुछ return नहीं करेगा fun.
function name is pzz

```
public static void pzz(int n)
{
→ System.out.print(n + " ");
→ pzz(n-1);
→ System.out.print(n + " ");
→ pzz(n-1);
→ System.out.print(n + " ");
}
```

n is passed as the argument of function

हमने High Level Thinking से ये code तो लिखलिया but this code is not complete. because code को अभी ये पता ही नहीं है कि उसे रुकना कब है। So, basically we need to find the Base Case & How do we find the Base Case?

We find the Base Case by doing the

Low Level Thinking.

⇓

Dry run & draw the stack frame to find the Base Case

Love Level Thinking

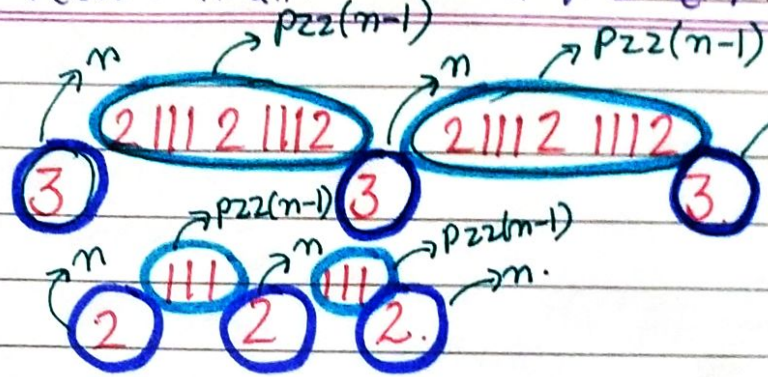
→ इस ques में एक func में 2 calls हो रही हैं.

Page No.
Date :

eg → n=3

2 calls समझाने के लिए ये ques है।

Expectation :-



Faith :-

Expectation meets faith

↳ Pzz(3) को express करना है Pzz(2) की terms में।

```
System.out.print(n+" ");   → Line a
pzz(n-1);                  → Line b
System.out.print(n+"'");   → Line c.
pzz(n-1);                  → Line d.
System.out.print(n+"'");   → Line e.
```

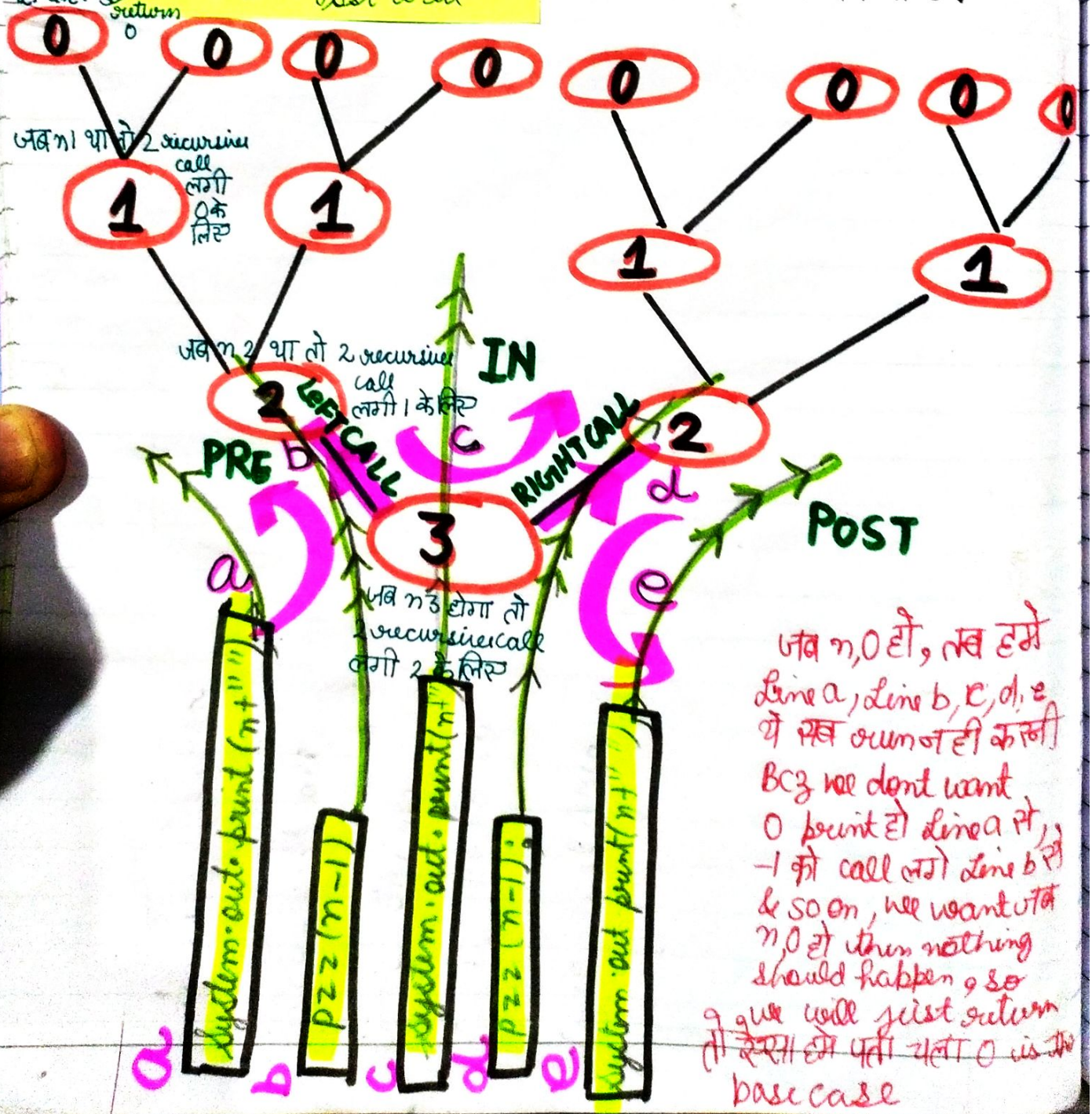
→ ये Line b और d. दोनों Line में pzz function को call line मतलब 2 recursive calls 1 ही function में

```
public static void
  pzz(int n){
System.out.print(n+" ");   → Line a → Prearea
pzz(n-1);                  → Line b → Left Call
System.out.print(n+" ");   → Line c → Inarea
pzz(n-1);                  → Line d → Right Call
System.out.print(n+" ");   → Line e → Postarea
}
```

Line a → Pre area
Line b → Left Call
Line c → In area
Line d → Right call
Line e → Post area

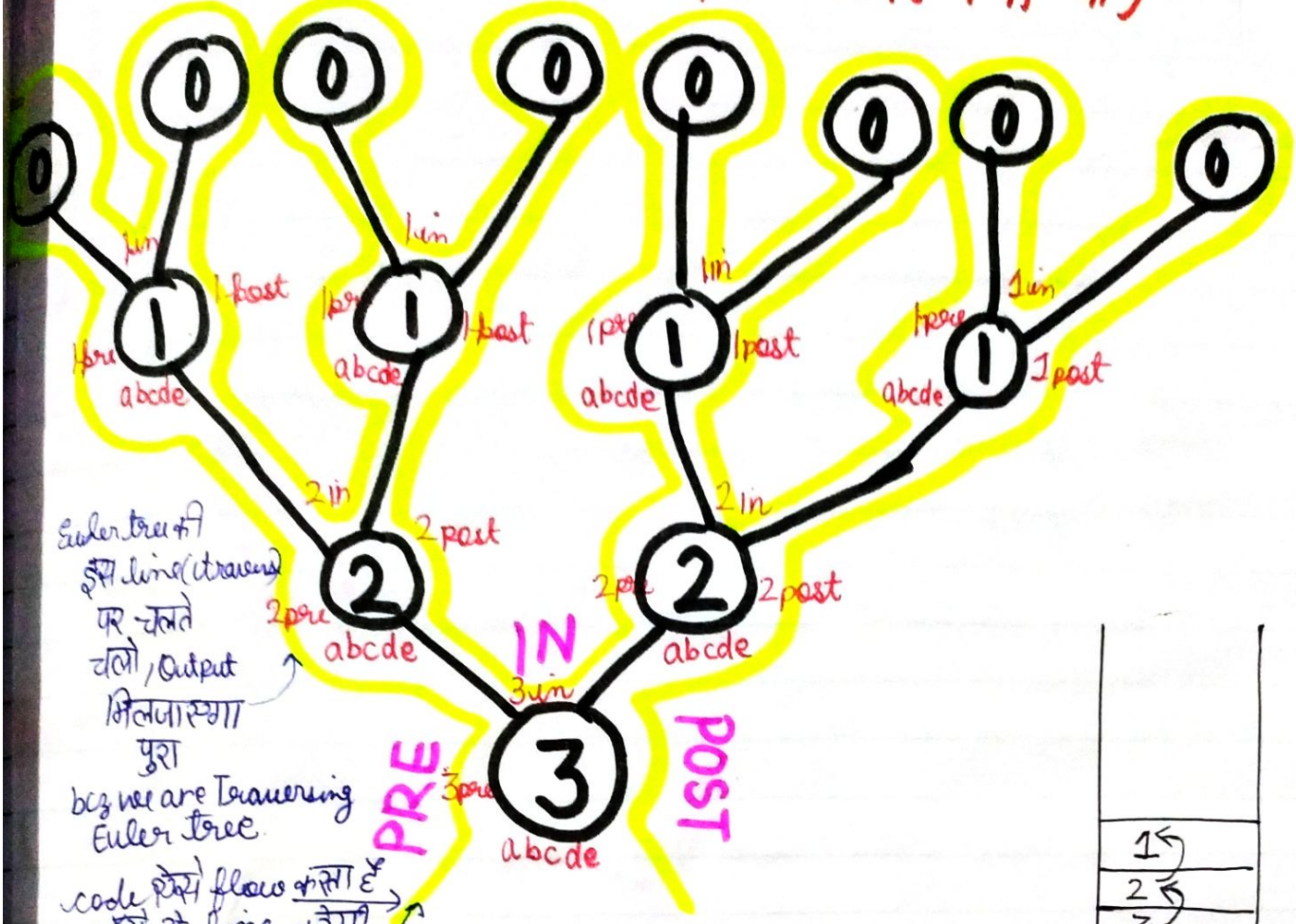
These 2 are recursive calls तो जब भी function में 2 recursive call दिखे जाए तो समझ जाओ कि Euler tree बनाके देखना है।
function में 2 या 2 से ज्यादा recursive call मतलब Euler tree बनाना है।

जब n 0 हो तब हमें कुछ भी output नहीं चाहिए so return



जब n,0 हो, तब हमें Line a, Line b, c, d, e ये सब execute ही करनी Bcz we dont want 0 print हो Line a से, -1 को call लगे Line b से & so on, we want जब n,0 हो then nothing should happen, so we will just return तो ऐसा हो पता चला 0 is the base case

Euler tree हमें stack की timeline देता है
(किस समय stack पर क्या था)




```
public static void pzz(int n)
{ if(n==0)
  {return;}                    → Base Case
System.out.println(n+"pre");   → line a
pzz(n-1);                      → line b
System.out.println(n+"in");    → line c
pzz(n-1);                      → line d
System.out.println(n+"post");  → line e
}
```

जैसी ही ये 5 line चलजाएंगी pzz(n) memory से wipe out होजाता है with the help of Garbage Collection (ये garbage इसलिए बनगया क्योंकि अब उसे कोई use नहीं करेगा)

Output →

```
3pre
2pre
1pre
1in
1post
2in
1pre
1in
1post
2post
3in
2pre
1pre
1in
1post
2in
1pre
1in
1post
2post
3post
```

As soon as the function gets executed, it gets wiped out of the Stack memory & console displays the output as required.

Note:- Binary search या एक तरीके से ही होती है
Inverted Euler Tree क्योंकि उसमें भी एक func 2 calls जाती है।

PRINT Zigzag <code>

```
import java.util.*;
import java.io.*;
public class Main{
  public static void main(String args[])
  {
    Scanner scan = new Scanner(System.in);
    int n = scan.nextInt();
    pzz(n);

// Expectation → pzz(3) ⇒ 3 2 1 1 2 1 1 1 2 3 2 1 1 1 2 1 1 1 2 3
// Faith → pzz(2) → 2
// Expectation meets Faith → 3 pzz(2) 3 pzz(2) 3

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

    System.out.println(n + "pre");
    pzz(n-1);
    System.out.println(n + "in");
    pzz(n-1);
    System.out.println(n + "post");

  }
}
```

Ques-2

Tower Of HANOI

Page No.
Date

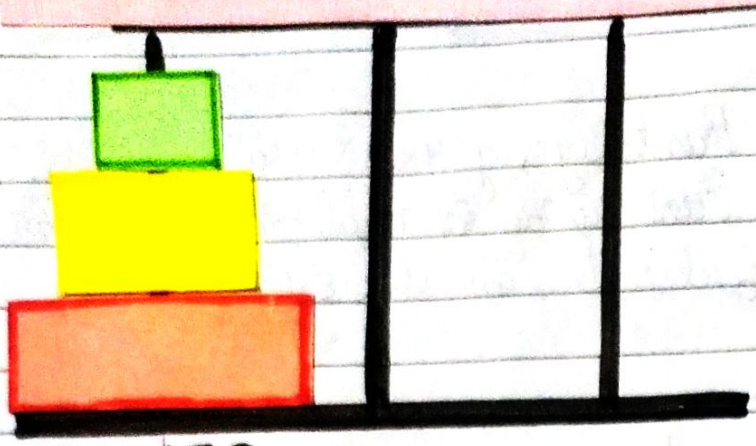
Tower Of Hanoi और Print Zig Zag ques का दोनों same ही है.
Tower Of Hanoi को हम Print Zig Zag की application बोल सकते हैं
इन दोनों ही ques में Euler Tree का use हमें करना होगा।

1 → There are 3 towers.
Tower 1 has n disks, where n is a positive Integer
Tower 2 is empty, means 0 disks
Tower 3 is empty, means 0 disks.

2. → The disks are increasingly placed in terms of size such that the smallest disk is on the top and the largest disk is at the bottom.

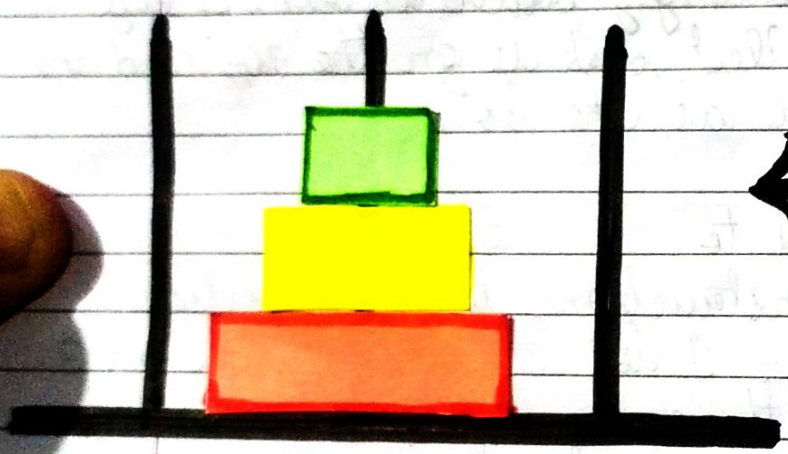
3. → We are required to.
3.1. Print the instructions to move the disks
3.2. from tower 1 to tower 2 using tower 3.
3.3. following the rules :-
3.3.1. move 1 disk at a time
3.3.2. never place a smaller disk under a larger disc
3.3.3. you can only move a disk at the top

Initial & Final Condition of Towers



← INITIAL CONDITION

TOWER 1



← FINAL CONDITION

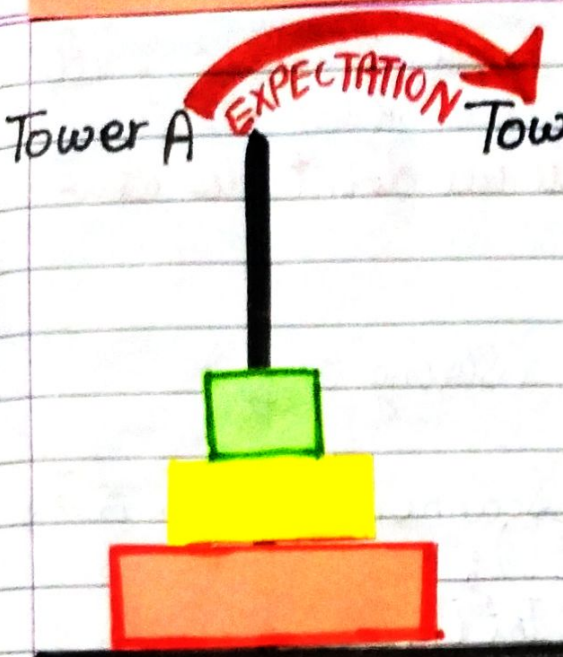
TOWER 2

1. Expectation

HIGH LEVEL THINKING → 2. Faith

3. Expectation meets Faith

High Level Thinking



1. Expectation

toh(3, A, B, C)
↑ all 3 disc
tower of hanoi

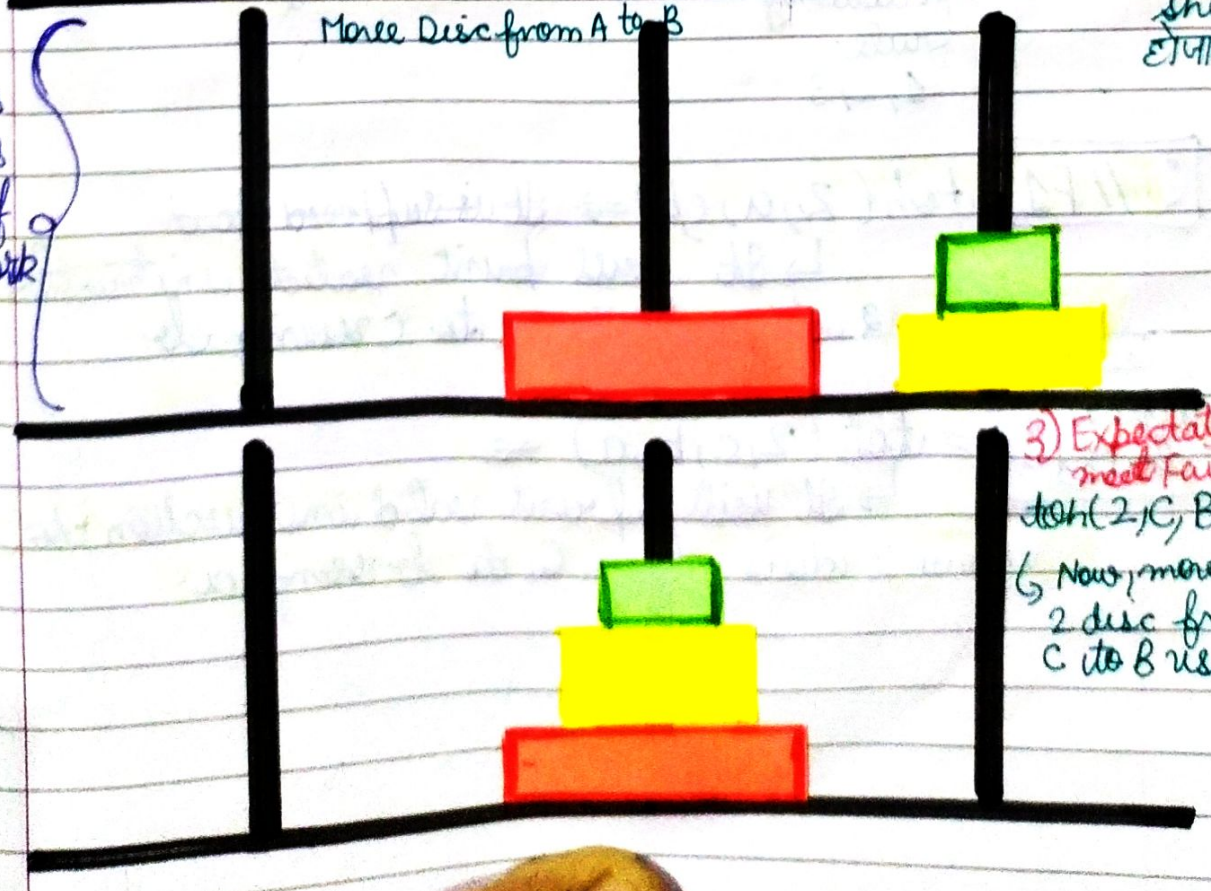
→ It is expected that the func toh(3, A, B, C) needs to print the valid instruction to move 3 disc from A to B using C

2. Faith

toh → (2, A, C, B)

Print all the instructions following which we can move 2 discs from A to C using B

→ ऐसा मान लो की 2 disc सही से shift हो जाएगी



3) Expectation meet Faith

toh(2, C, B, A)

↳ Now, move 2 disc from C to B using A

concepts understood

Page No.
Date

Now, using the High Level Thinking, we must try to code

Tower of Hanoi <code & dry run>

```
public static void main (String[]args)
{
Scanner scan= new Scanner (System.in);
int n= scn.nextInt();
int t1id= scn.nextInt();
int t2id= scn.nextInt();
int t3id= scn.nextInt();
toh( n, t1id, t2id, t3id)
```

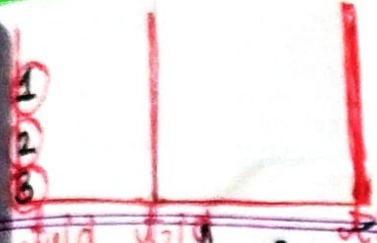
n no. of disks ← | t1id → Name of first Tower | t2id → Name of second Tower | t3id → Name of 3rd Tower

Expectation ⇒ toh(3, a, b, c) ⇒ It is supposed to print valid (or 123) instructions to move 3 discs from a to b using c

instruction following rule 1, 2, 3

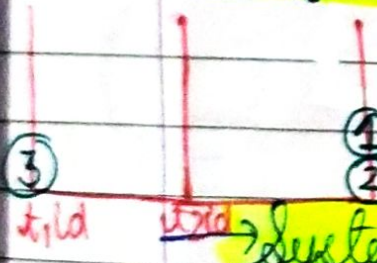
Faith = toh(2, a, c, b) ⇒
↳ It will print valid instruction to move 2 discs from a to c using b.

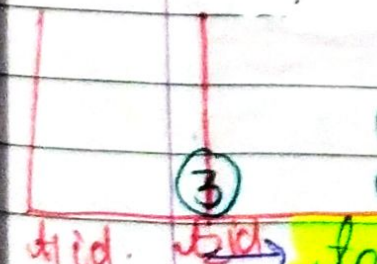
Faith 2 = toh(2, c, b, a) ⇒
↳ It will print valid instruction to move 2 discs from c to b using a.

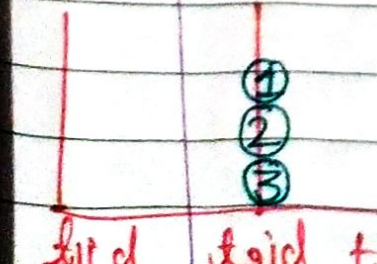


Page No.
Date

```
public static void toh (int n, int t1id,
                        int t2id, int t3id)
{
    toh (n-1, t1id, t3id, t2id);
    // now t1 will have only 1 disc,
    // t2 is now empty,
    // t3 has 2 discs
    System.out.println (n + "[" + t1id + "--> " +
                        t2id + "]");
    // t1 is empty,
    // t2 has 3rd disc,
    // t3 has top two
    toh (n-1, t3id, t2id, t1id);
    // t1 is empty // t3 is emky // t2 has all the disk
}
```







```
public static void toh (int n, int t1id, int t2id, int t3id){
```
→ no. of disc

```
if(n==0)
{ return; }
```
] Base case

] No local area

toh(n-1, t1id, t3id, t2id); → 1st time function calling

जब first call होगी तो function call करते हुए second और third towers की position exchange होगी function के parameter में

```
System.out.println( n+ "[" + t1id + "->" + t2id + "]");
```
↑ In area.

toh (n-1, t3id, t2id, t1id); → 2nd time function calling.

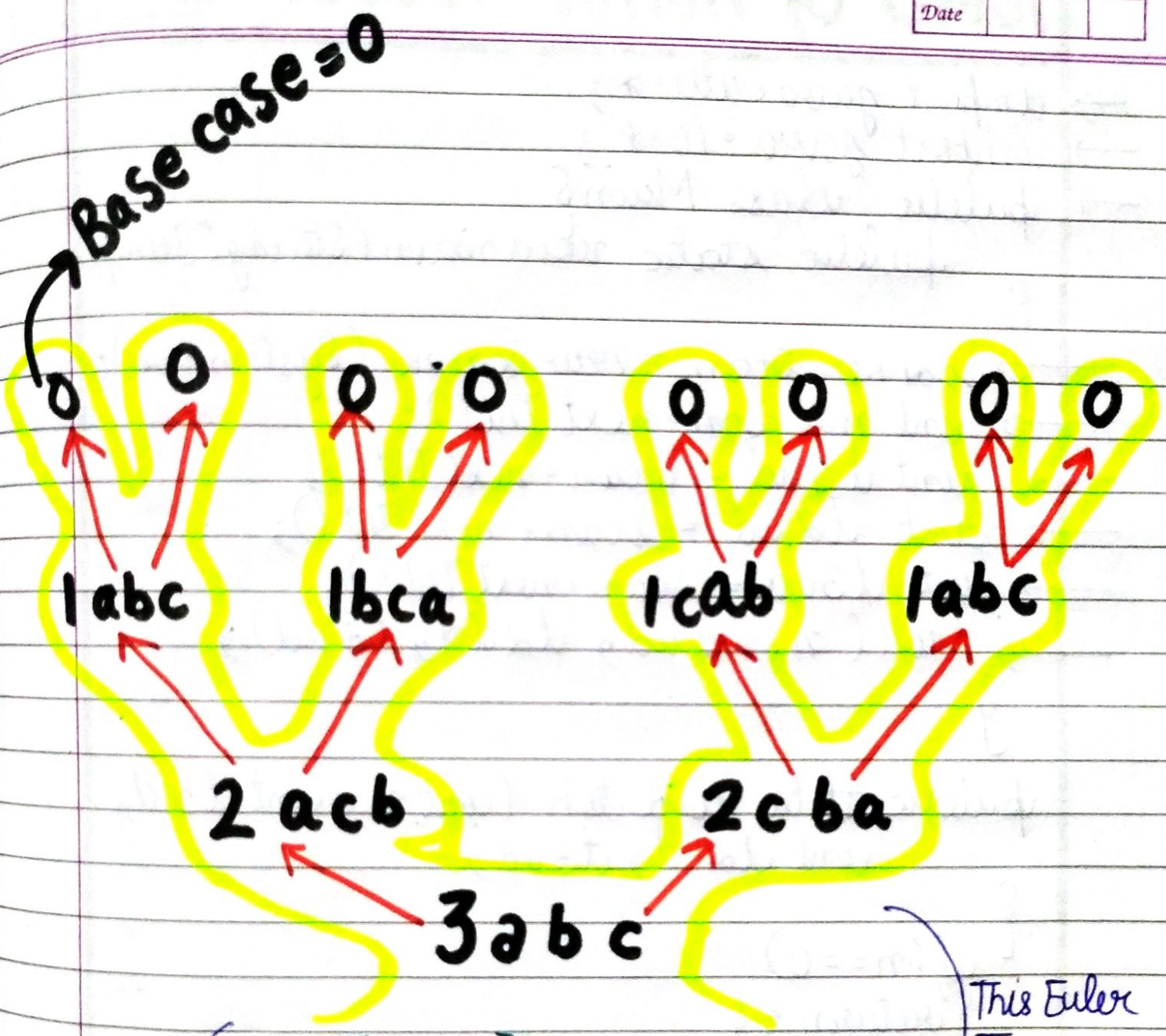
जब second call होगी तो function call करते हुए first और third towers की position exchange होगी. function के parameter में।

] No Past Area.

```
}
```

→ अब ये follow करते हुए Euler tour बनाने में आसानी होगी।

Date



This Euler Tree Traversal gives the following output

Output

1 → [a → b]
2 → [a → c]
1 → [b → c]
3 → [a → b]
1 → [c → a]
2 → [c → b]
1 → [a → b]

Tower Of Hanoi <code>

Page No.
Date

```
import java.util.*;
import java.io.*;
public class Main{
    public static void main(String[]args)
    {
        Scanner scan = new Scanner(System.in);
        int n = scan.nextInt();
        int t1id = scan.nextInt();
        int t2id = scan.nextInt();
        int t3id = scan.nextInt();
        toh(n,t1id,t2id,t3id);
    }
public static void toh(int n, int t1id,
        int t2id, t3id)
{
    if(n==0)
    {return;}

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

    }
}
```

Introduction TO Recursion DONE

— Tamanna Verma