Recursion → High level Understanding.
→ low level Understanding.

High level
① Expectation define,
② faith || Important
③ Merging of faith & Expectation.

High level	low level.
① We establish faith & Expectation to extract Recursive code.	① Base case extraction.
	② flow of code.

Example – Expectation Pd (5) = 5 4 3 2 1 || Expectation

faith Pd(4) = 4 3 2 1

Merging → Pd(5) = (5) Pd(4)

Print by myself

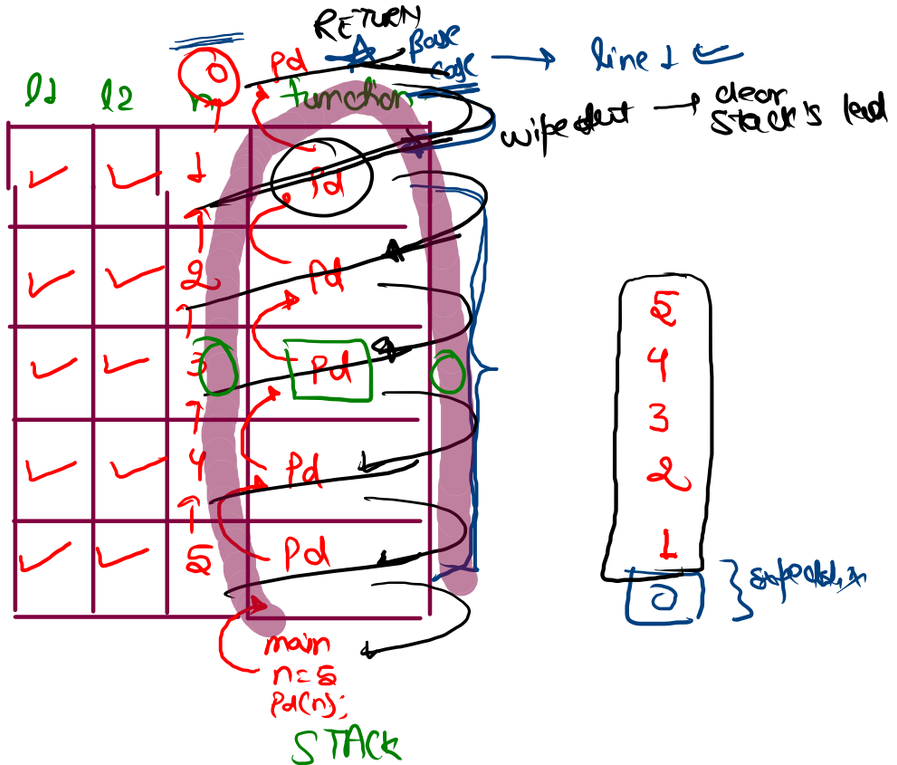
Generic Pd(n) = (n) Pd(n-1)

Pr. myself

```
public static void pd(int n){
    System.out.println(n);
    pd(n - 1);
}
```

if(n==0) return;

Function Return
→ ① RETURN keyword encounter
→ ② all lines of function was executed



Print Decreasing Increasing - High level Analysis

Expectation

Pdi(5) = 5 4 3 2 1 1 2 3 4 5

faith

Pdi(4) = 4 3 2 1 1 2 3 4

Merging → Expectation + faith

Pdi(5) = (5) Pdi(4) (5)

(5): Print by myself

(5): Print by myself

Generic → Pdi(n) = n Pdi(n-1) n] Recursive

low level Analysis

```
public static void pdi(int n) {
    System.out.println(n);
    pdi(n--1);
    System.out.println(n);
}
```

Pre n

call

Post n

if(n==0) return;

Base case

Return

Pdi(0) Pdi(1) Pdi(2) Pdi(3) Pdi(4) main

Before call — Pre Area

After call — Post Area

l1	l2	l3	n	function
✓	✓	✓	0	Pdi
✓	✓	✓	1	Pdi
✓	✓	✓	2	Pdi
✓	✓	✓	3	Pdi
✓	✓	✓	4	Pdi

n=4 Pdi(4) STACK

Program terminate

Base case

Expectation

Pre 4 ✓
Pre 3 ✓
Pre 2 ✓
Pre 1 ✓
Post 1 ✓
Post 2 ✓
Post 3 ✓
Post 4 ✓

Print ZigZag →

Expectation → pzz(1) → 1 1 1

1. Here are a few sets of inputs and outputs for your reference

Input1 -> 1

Output1 -> 1 1 1

Expectation pzz(2) → 2 [1 1 1] 2 [1 1 1] 2

pzz(1) pzz(1)

Experience → understanding → DRY RUN

Input2 -> 2

Output2 -> 2 1 1 1 2 1 1 1 2

Input2 -> 3

Output3 -> 3 2 1 1 1 2 1 1 1 2 3 2 1 1 1 2 1 1 1 2 3

Expectation pzz(3) → 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)

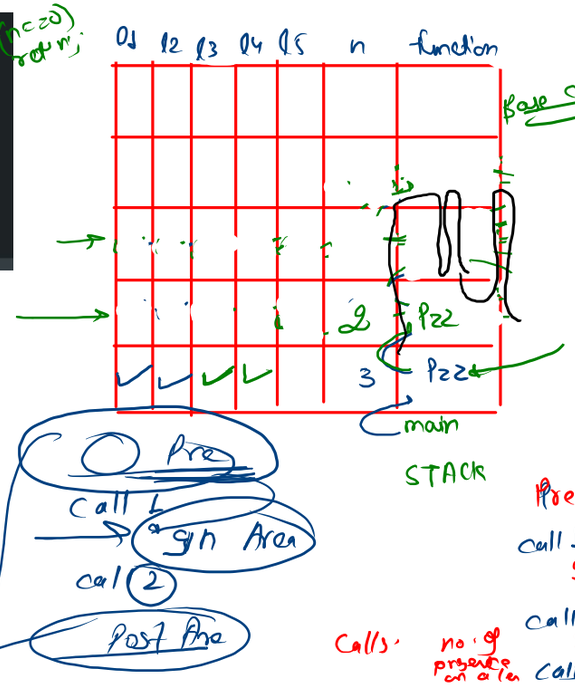
faith pzz(2) = 2 1 1 1 2 1 1 1 2

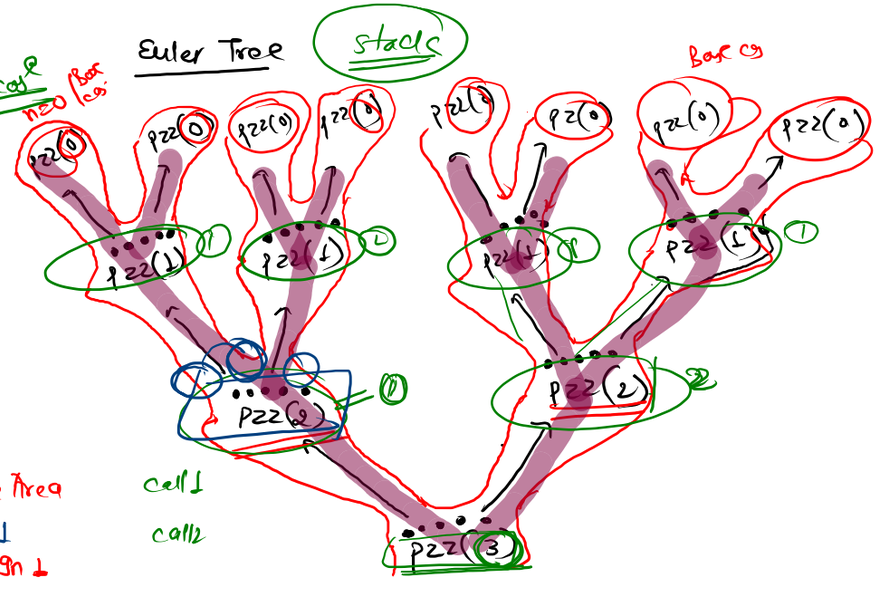
Recursive,

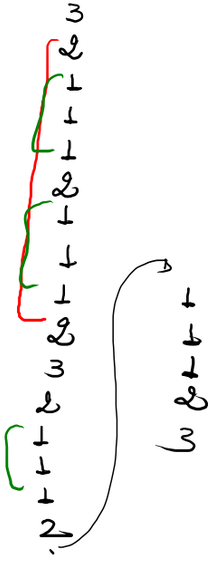
Merging pzz(3) = 3 pzz(2) 3 pzz(2) 3

generic pzz(n) = n pzz(n-1) n pzz(n-1) n

```
//print zigzag
public static void pzz(int n) {
    if(n==0) return;
1.  System.out.println(n);
2.  pzz(n--1);
3.  System.out.println(n);
4.  pzz(n--1);
5.  System.out.println(n);
}
```







3 2 1 1 1 2 1 1 1 2 3

Pre Area

call 1

call 2

Fn 1

call 2 Fn 2 Area

Calls: no. of presence on a len calls

1 → 2

2 → 3

3 → 4

4 → 6

Post Area

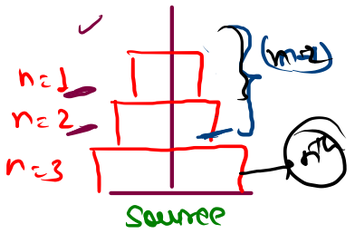
n calls → (n+1) time present

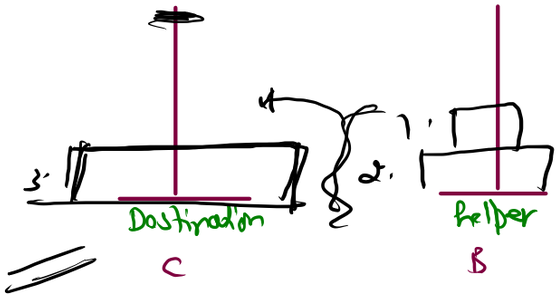
Tower of Hanoi →

→ 3 towers (src, dest, hlp)
→ n disc (n=3)

Target
↳ Print steps to move n discs from source to destination using helper with follow the set of rules.

Rules →
1. You can pick one disc at a time
2. You can't place heavy disc on light disc
3. Print min steps





Expectation → toh(n) → toh(n-1)
print min steps to move n discs from source to destination using helper.

faith - toh(n-1) (n-2)
ye can print min steps to move (n-1) discs from source to helper using destination

Merging

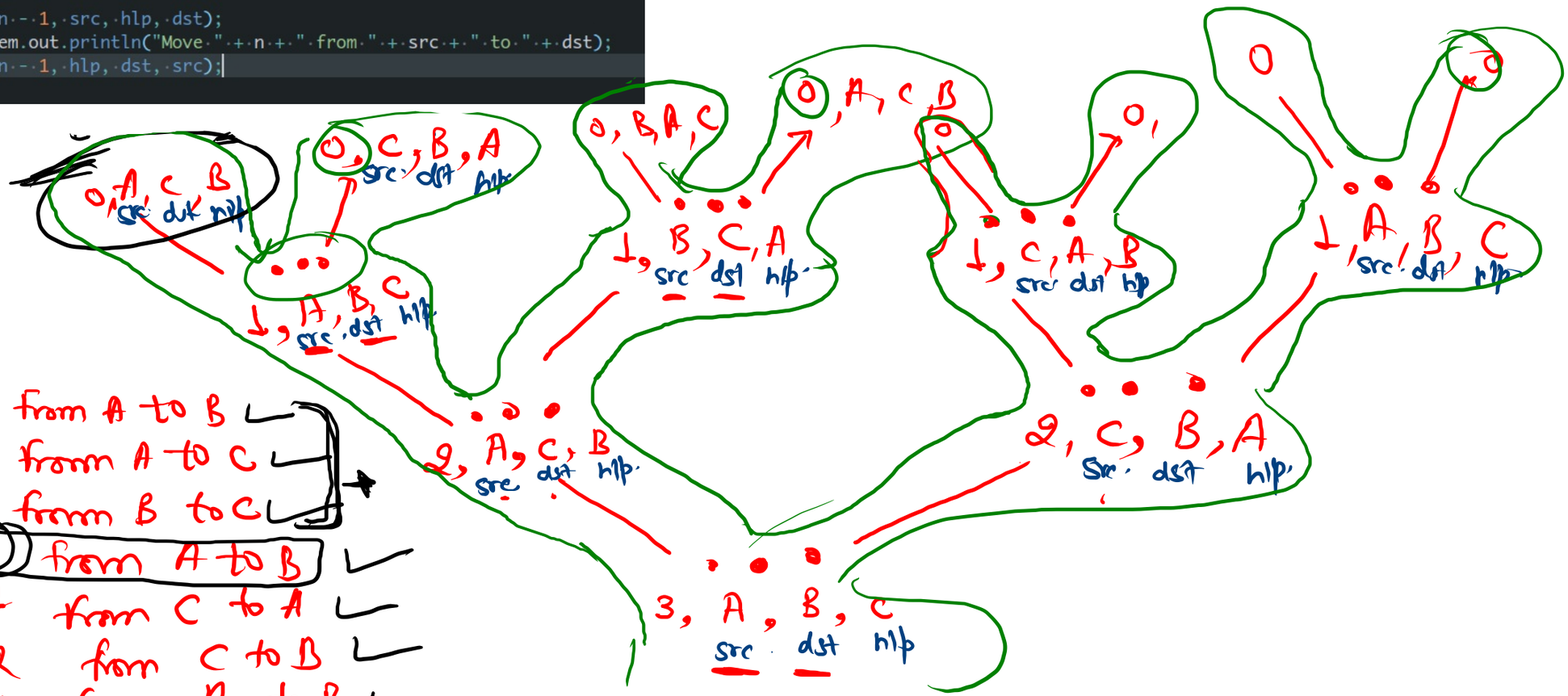
Done by myself →

nth disc move src to destination

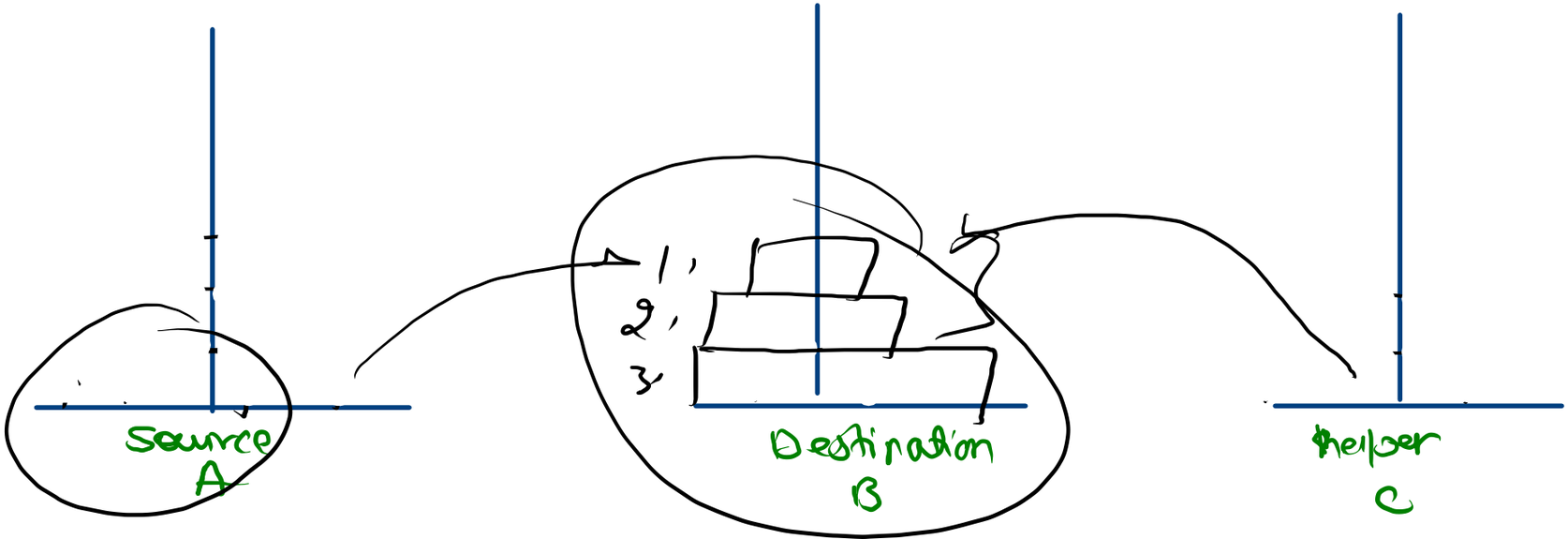
faith toh(n-1) move helper destination using source

```java
public static void toh(int n, char src, char dst, char hlp) {
    ....

1   toh(n - 1, src, hlp, dst);
2   System.out.println("Move " + n + " from " + src + " to " + dst);
3   toh(n - 1, hlp, dst, src);
}
```



move 1 from A to B
move 2 from A to C
move 1 from B to C
move 3 from A to B
move 1 from C to A
move 2 from C to B
move 1 from A to B



Recursion with array →

↳ Traversal

↳ Important → Index ||- print Array

0	1	2	3	4	5
10	20	30	40	50	60

arr, index

Expected

printArr(arr, 0) = 10 20 30 40 50 60

faith printArr(arr, 1) = 20 30 40 50 60

Merging → prr(arr, 0) = arr[0] prr(arr, 1)

print(arr, i) = arr[i] , print(arr, i+1);

print by myself — faith

```
//display array
public static void diasplayArr(int[] arr, int idx) {
    System.out.println(arr[idx]); //done by myself
    diasplayArr(arr, idx + 1); //faith
}
```

Index — 0 1 2 3 4 5

arr = 10, 20, 30, 40, 50, 60

(arr, 4) (arr, 5) (arr, 6)

(arr, 3)

(arr, 2)

(arr, 1)

(arr, 0)

linear Recursion

Power → linear

Power →

10
20
30
40
50
60

arr = { 10, 20, 30, 40, 50 }
0 1 2 3 4

displayReverse -

Expectation , displRev(arr, 0) → 50 40 30 20 10

faith , displRev(arr, 1) → 50 40 30 20

Meger displRev(arr, 0) → displRev(arr, 1) sysO(arr[0])

displRev(arr, 1) — faith

sysO(arr[0]) — done by myself.

Base ~~else~~ arr.length

Generic Base case = idx == arr.length

displRev(arr, idx) = displRev(arr, idx+1) sysO(arr[idx])

max of an array → arr = {10, 20, 30, 17, 16, 7}

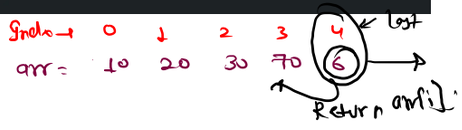
Expectation → maxEle(arr, 0) → max element return (0 to arr.length)

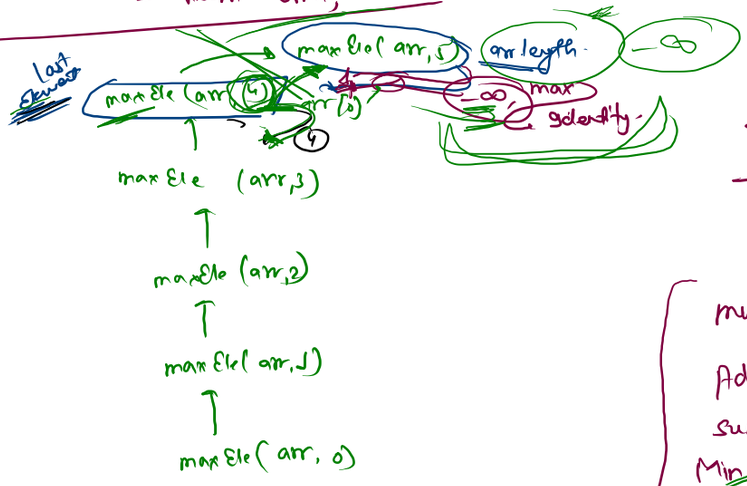
Faith → maxEle(arr, 1) → max (from 1 to arr.length)

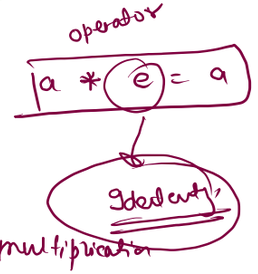
Merging → maxEle(arr, 0) ⇒ Rec.max = maxEle(arr, 1);
= Omax = max(Recmax, arr[0]);
⇒ Return Omax;

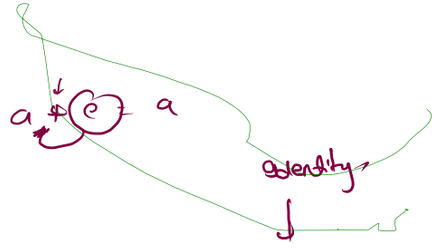
```
//max in array
public static int maxEle(int[] arr, int idx){

    int recMax = maxEle(arr, idx + 1);
    int omax = Math.max(recMax, arr[idx]);
    return omax;
}
```









a * e = a ⇒ e = 1

a + e = a ⇒ e = 0

a - e = a ⇒ e = 0

min(a, e) = a ⇒ e ≥ ∞

max(a, e) = a ⇒ e ≥ -∞