

Recursion → High level understanding.
↓
Low level understanding.

High level

- ① Expectation defines,
- ② faith // important
- ③ Merging of faith & expectation.

Example - Expectation $Pd(5) = \underline{\underline{5 \quad 4 \quad 3 \quad 2 \quad 1}} \quad || \text{Expectation}$
faith $\underline{\underline{Pd(4) = \quad 4 \quad 3 \quad 2 \quad 1}}$

Merging → $\underline{\underline{Pd(5) = \underbrace{5}_{\downarrow \text{Point by myself}} \underbrace{Pd(4)}}}$

Generic $\underline{\underline{Pd(n) = \underbrace{n}_{\text{f. myself}} Pd(n-1)}}$

High level

- ① we establish faith & expectation to extract recursive code.

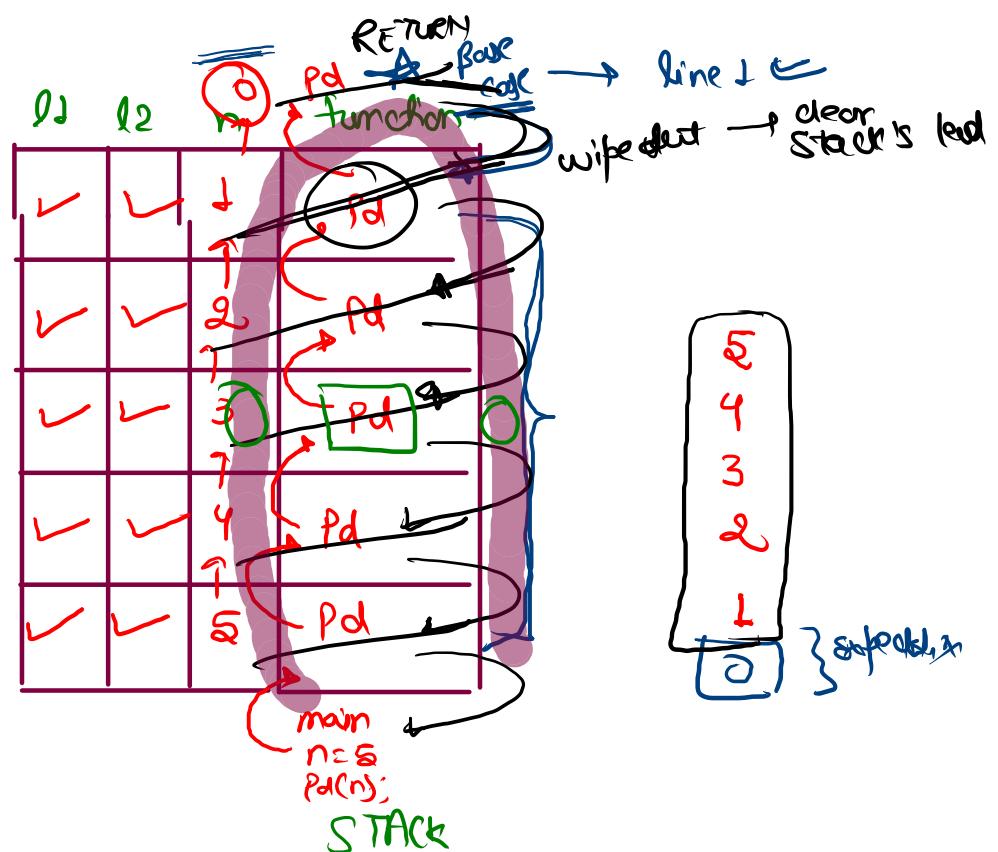
Low level

- ① base case extraction.
- ② flow of code.

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



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



Point Decreasing Increasing - High level Analysis

Expectation

$$Pdi(5) = \underline{\underline{5 \ 4 \ 3 \ 2 \ 1 \ 1 \ 2 \ 3 \ 4}} \ 5$$

fault \Rightarrow

$$Pdi(4) = \underline{\underline{4 \ 3 \ 2 \ 1 \ 1 \ 2 \ 3 \ 4}}$$

Merging - \rightarrow

Expectation \leftrightarrow fault

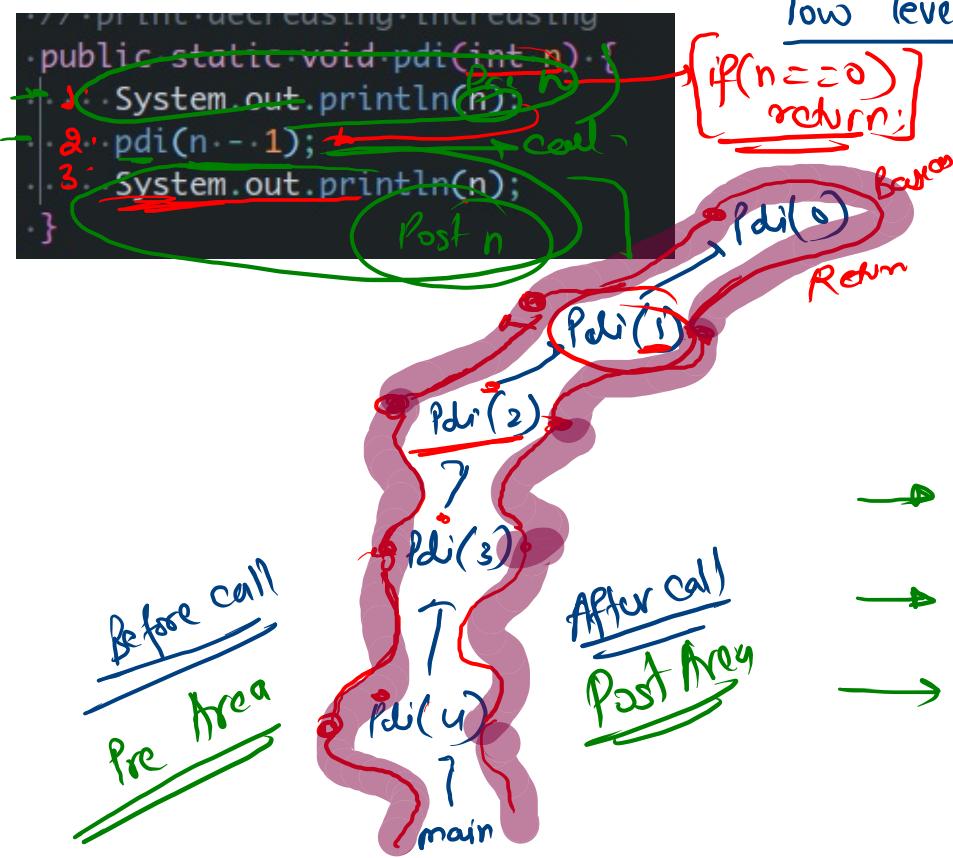
$$Pdi(5) = \circled{5} \ Pdi(4) \circled{5}$$

Point by
myself

Point by
myself

$$\text{Generic} \rightarrow Pdi(n) = \underline{n} \ Pdi(n-1) \underline{n} \] \text{ recursive}$$

low level Analysis



q1 q2 q3 n function

			0	Pdi
✓	✓	✓	1	Pdi
✓	✓	✓	2	Pdi
✓	✓	✓	3	Pdi
✓	✓	✓	4	Pdi

Expectation

pre	4	-
pre	3	-
pre	2	-
pre	1	-
post	1	-
post	2	-
post	3	-
post	4	-

n=4
Pdi(4)
STACK
Program terminate

Point ZigZag →

Expectation → $pzz(1) \rightarrow \downarrow \downarrow \downarrow$

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

Input1 → 1

Output1 → 1 1 1

Input2 → 2

Output2 → 2 1 1 1 2 1 1 1 2

Expectation $pzz(2) \rightarrow 2 \left[\begin{array}{c} \downarrow \\ pzz(1) \end{array} \right] 2 \left[\begin{array}{c} \downarrow \\ pzz(1) \end{array} \right] 2 \left[\begin{array}{c} \downarrow \\ pzz(2) \end{array} \right]$

Experience
understanding
DRY RUN

Input2 → 3

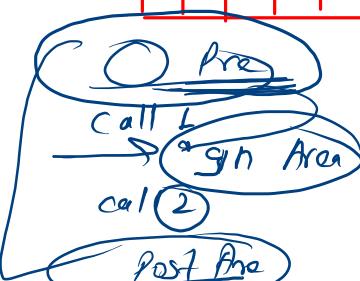
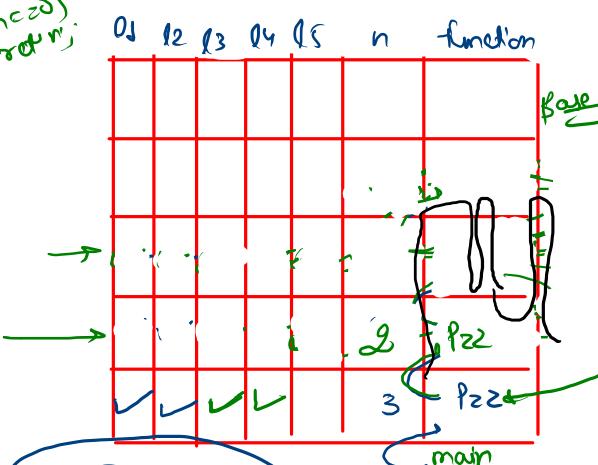
Output3 → 3 2 1 1 1 2 1 1 1 2 3 2 1 1 2 1 1 1 2 3

Expectation $pzz(3) \rightarrow 3 \left[\begin{array}{c} 2 \downarrow \downarrow \downarrow 2 \downarrow \downarrow \downarrow 2 \\ pzz(2) \end{array} \right] 3 \left[\begin{array}{c} 2 \downarrow \downarrow \downarrow 2 \downarrow \downarrow \downarrow 2 \\ pzz(2) \end{array} \right] 3$

faith $pzz(2) = 2 \downarrow \downarrow \downarrow 2 \downarrow \downarrow \downarrow 2$

Merging generic $pzz(3) = \frac{3}{n} pzz(n) \frac{3}{n} pzz(n-1) \frac{3}{n} pzz(n-2) \frac{3}{n}$ recursive,

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



C_1
 C_2
 C_3
 C_4
 C_5

Calls: $\frac{\text{no. of recursive calls}}{\text{parent calls}}$ $\frac{\text{call 1}}{\text{call 2 Area}}$
 $1 \rightarrow 2$ Post Area
 $2 \rightarrow 3$
 $3 \rightarrow 4$
 $4 \rightarrow 5$

Post Area

call 1

call 2

call 3

call 4

call 5

call 6

call 7

call 8

call 9

call 10

call 11

call 12

call 13

call 14

call 15

call 16

call 17

call 18

call 19

call 20

call 21

call 22

call 23

call 24

call 25

call 26

call 27

call 28

call 29

call 30

call 31

call 32

call 33

call 34

call 35

call 36

call 37

call 38

call 39

call 40

call 41

call 42

call 43

call 44

call 45

call 46

call 47

call 48

call 49

call 50

call 51

call 52

call 53

call 54

call 55

call 56

call 57

call 58

call 59

call 60

call 61

call 62

call 63

call 64

call 65

call 66

call 67

call 68

call 69

call 70

call 71

call 72

call 73

call 74

call 75

call 76

call 77

call 78

call 79

call 80

call 81

call 82

call 83

call 84

call 85

call 86

call 87

call 88

call 89

call 90

call 91

call 92

call 93

call 94

call 95

call 96

call 97

call 98

call 99

call 100

call 101

call 102

call 103

call 104

call 105

call 106

call 107

call 108

call 109

call 110

call 111

call 112

call 113

call 114

call 115

call 116

call 117

call 118

call 119

call 120

call 121

call 122

call 123

call 124

call 125

call 126

call 127

call 128

call 129

call 130

call 131

call 132

call 133

call 134

call 135

call 136

call 137

call 138

call 139

call 140

call 141

call 142

call 143

call 144

call 145

call 146

call 147

call 148

call 149

call 150

call 151

call 152

call 153

call 154

call 155

call 156

call 157

call 158

call 159

call 160

call 161

call 162

call 163

call 164

call 165

call 166

call 167

call 168

call 169

call 170

call 171

call 172

call 173

call 174

call 175

call 176

call 177

call 178

call 179

call 180

call 181

call 182

call 183

call 184

call 185

call 186

call 187

call 188

call 189

call 190

call 191

call 192

call 193

call 194

call 195

call 196

call 197

call 198

call 199

call 200

call 201

call 202

call 203

call 204

call 205

call 206

call 207

call 208

call 209

call 210

call 211

Tower of Hanoi →

- 3 towers (src, dest, helper)
- n discs ($n=3$)

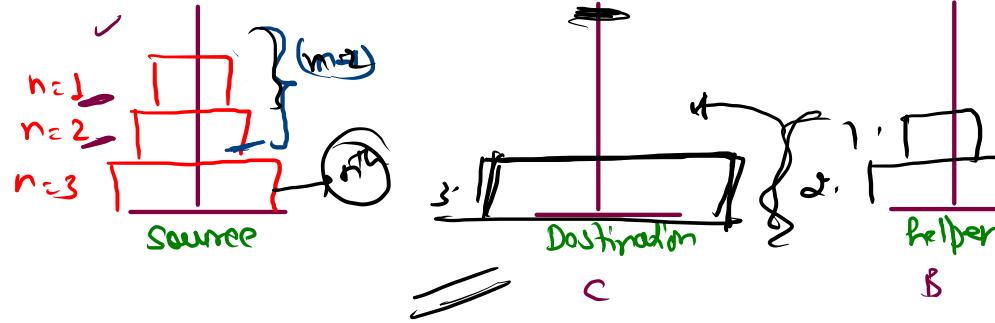
Target

↳ print steps to move n discs from source to destination using helper with follows 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



$$\text{expectation} \rightarrow \underline{\text{toh}(n)} \rightarrow \underline{\text{toh}(n-1)}$$

print min steps to move n discs from source to destination using helper.

$$\text{faith- } \underline{\text{toh}(n)}$$

set can print

min steps to move $(n-1)$ discs from source to helper using destination

Moving

Done by myself

n^{th} disc move src to destination

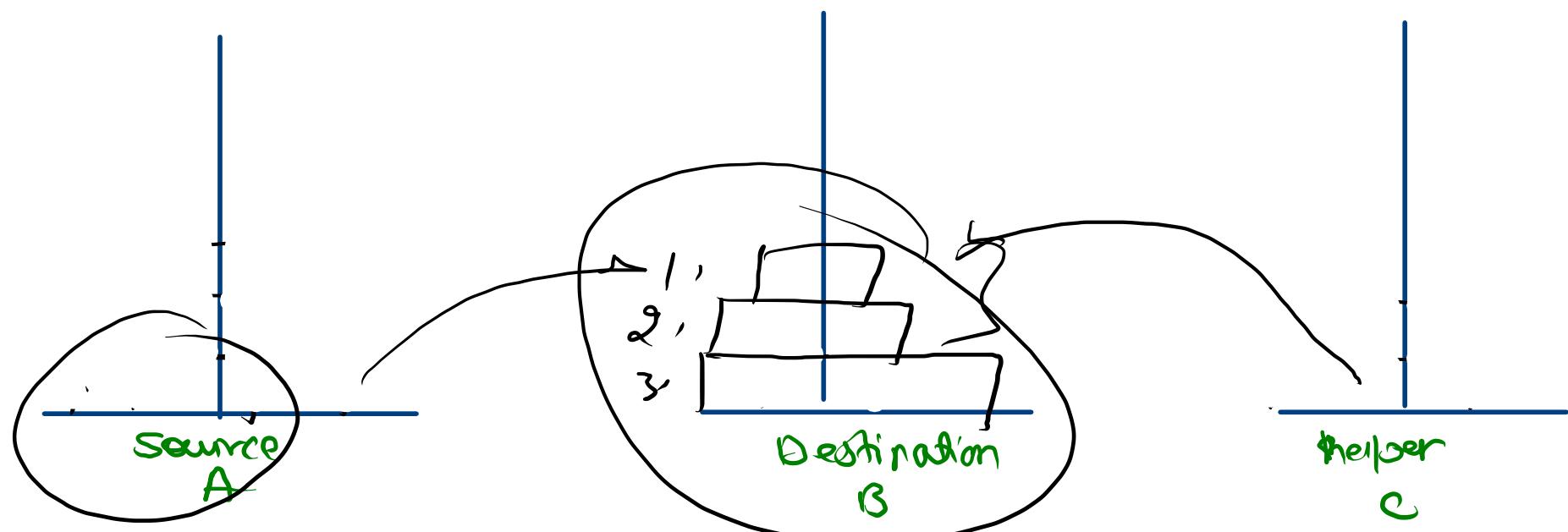
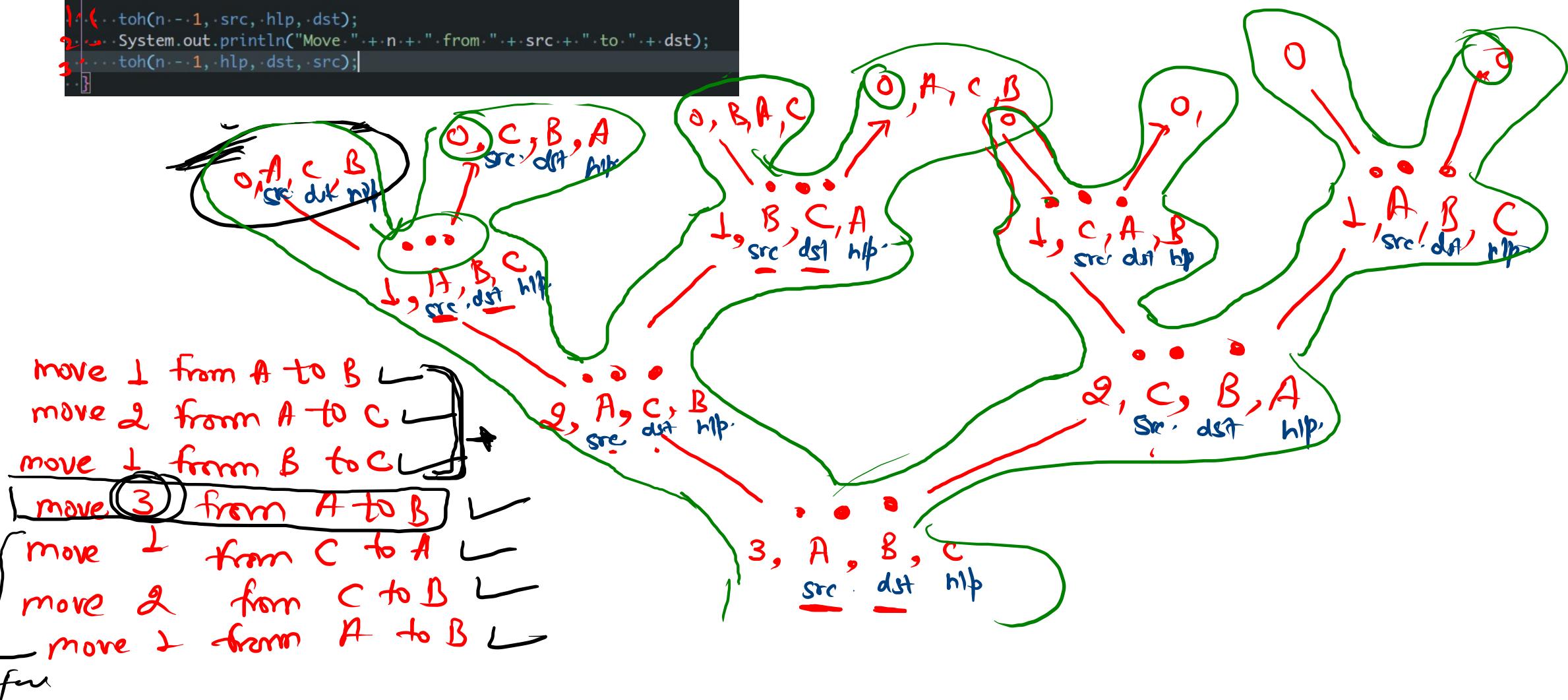
faith

$\text{toh}(n-1)$ move helper destination using source

```

public static void toh(int n, char src, char dst, char hlp) {
    if (n == 1) {
        System.out.println("Move " + n + " from " + src + " to " + dst);
    } else {
        toh(n - 1, src, hlp, dst);
        System.out.println("Move " + n + " from " + src + " to " + dst);
        toh(n - 1, hlp, dst, src);
    }
}

```



Recursion with array →

Traversing ↴ ↵

Important → index ↴ print Array

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

Expected ↗

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

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

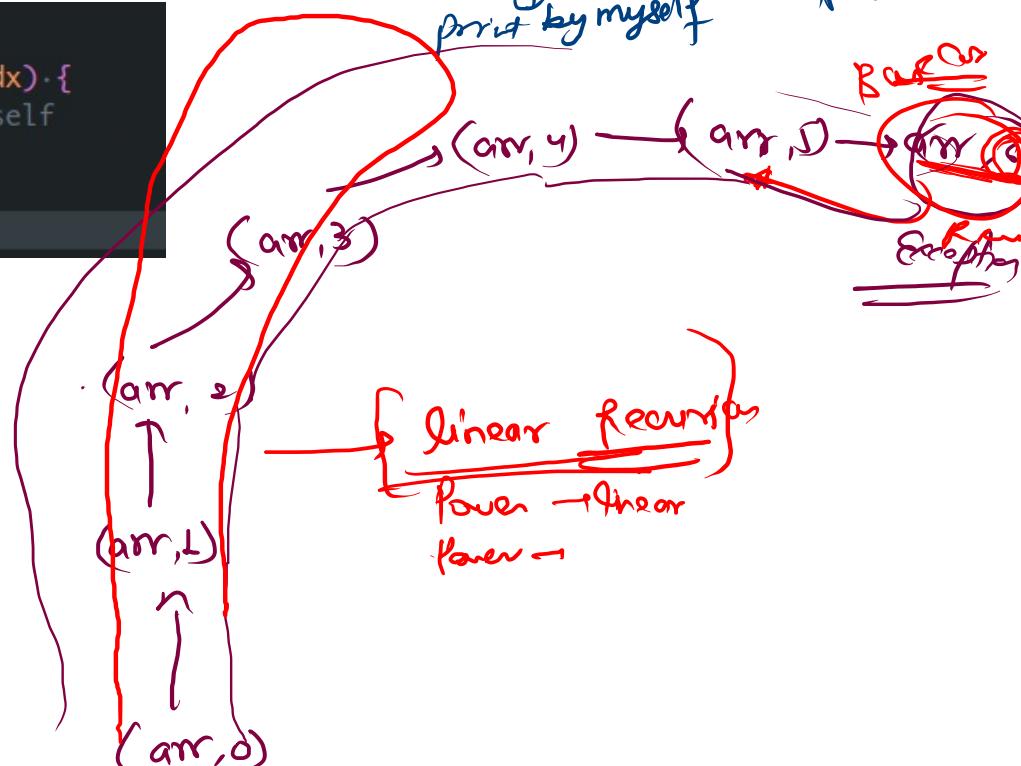
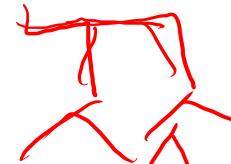
Merging → print(arr, 0) = arr[0] printArr(arr, 1)

→ print(arr, i) = (arr[i]), print(arr, i+1),
print by myself faith

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

idx = 0 1 2 3 4 5
arr = 10, 20, 30, 40, 50, 60

10
20
30
40
50
60



display Reverse -

arr = { 10, 20, 30, 40, 50 }

Expectation

, dispRev(arr, 0) → 50 40 30 20 10

fact ,

, dispRev(arr, 10) → 50 40 30 20,

Mager

dispRev(arr, 0) →

dispRev(arr, f) sys0(arr[0])
fact

done
by myself .

Base case arr.length

Generic

Base case =

idx = arr.length

dispRev(arr, idx) = dispRev(arr, idx+1) → sys0(arr[idx])

max of arr array →

$$\text{arr} = \begin{array}{c} \textcircled{10} \\ \text{---} \\ 20, 30, 17, 16, 7 \end{array}$$

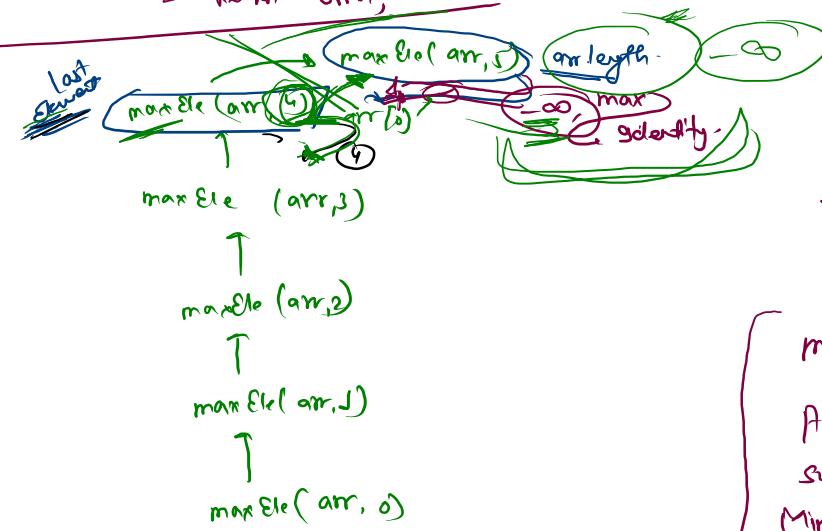
Expectation, $\rightarrow \max(\text{arr}, 0) \rightarrow \max \text{ element return. (0 to arr.length)}$
faith $\rightarrow \max(\text{arr}, 0) \rightarrow \max(\text{from } 1 \text{ to arr.length})$

Merging →

$$\max(\text{arr}, 0) \Rightarrow \text{rec. max} = \max(\text{arr}, 1); \\ = \text{omax} = \max(\text{Recmax}, \text{arr}[0]); \\ \Rightarrow \text{return omax};$$

```
...-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;
}
```

Index → 0 1 2 3 4 5 → last
 $\text{arr} = \begin{array}{cccccc} 10 & 20 & 30 & 70 & \textcircled{6} & 10 \end{array}$
 Return arr[5].



operator
 $|a * e| = a$
Identity

{ multiplication
 Addition
 subtraction
Min
Max

The diagram illustrates properties of multiplication and addition. On the left, it shows $a * e = a \Rightarrow e=1$ and $a + e = a \Rightarrow e=0$. On the right, it shows $a - e = a \Rightarrow e=0$, $\min(a, e) = a \Rightarrow e \geq a$, and $\max(a, e) = a \Rightarrow e \leq a$. A green oval labeled "e=0" is shown near the subtraction equation, and another labeled "e \geq a" is shown near the min equation. A red arrow labeled "Identity" points from the multiplication equation to the addition equation.