

Recursion & Backtracking

① Introduction

```
</> Print Increasing  
</> Print Decreasing  
</> Print Increasing Decreasing  
</> Power-linear  
</> Power-logarithmic  
</> Print Zigzag  
</> Tower Of Hanoi
```

② Recursion & Arrays

```
</> Display Array  
</> Display Array In Reverse  
</> Max Of An Array  
</> First Index  
</> Last Index  
</> All Indices Of Array
```

③ Recursion & ArrayList {Get}

```
</> Get Subsequence  
</> Get Kpc  
</> Get Stair Paths  
</> Get Maze Paths  
</> Get Maze Path With Jumps
```

⑤ Backtracking

```
</> Flood Fill  
</> Target Sum Subsets  
</> N Queens  
</> Knights Tour
```

④ Recursion on the way up (Print)

```
</> Print Subsequence  
</> Print Kpc  
</> Print Stair Paths  
</> Print Maze Paths  
</> Print Maze Paths With Jumps  
</> Print Encodings  
</> Print Permutations
```

Principle of Mathematical Induction (PMI)

① Trivial Case

$$\sum_{i=1}^0 i = 0$$

$$\sum_{i=1}^1 i = 1$$

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

② Assume

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

③ To prove

$$\begin{aligned}\sum_{i=1}^{k+1} i &= (\underbrace{1+2+\dots+k}_{\text{LHS}}) + (k+1) = \frac{k*(k+1)}{2} + (k+1) \\ &= \frac{(k+1)*(k+2)}{2} \end{aligned}$$

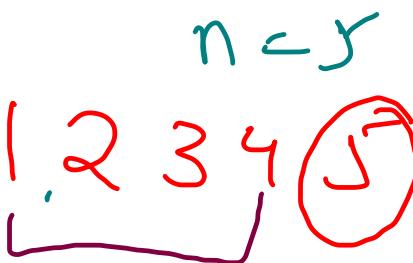
if $n=k+1$ LHS = RHS

Recursion

{ High-level thinking }

- ① Recursive function \rightarrow expectation

void printInc(int n) \rightarrow 1, 2, 3, - ... n



- ② Recursive fn \rightarrow smaller values \rightarrow faith (call)

void printInc(n-1) \rightarrow 1, 2, 3, - ... n-1 | 2 3 4

- ③ Meeting Expectation with faith

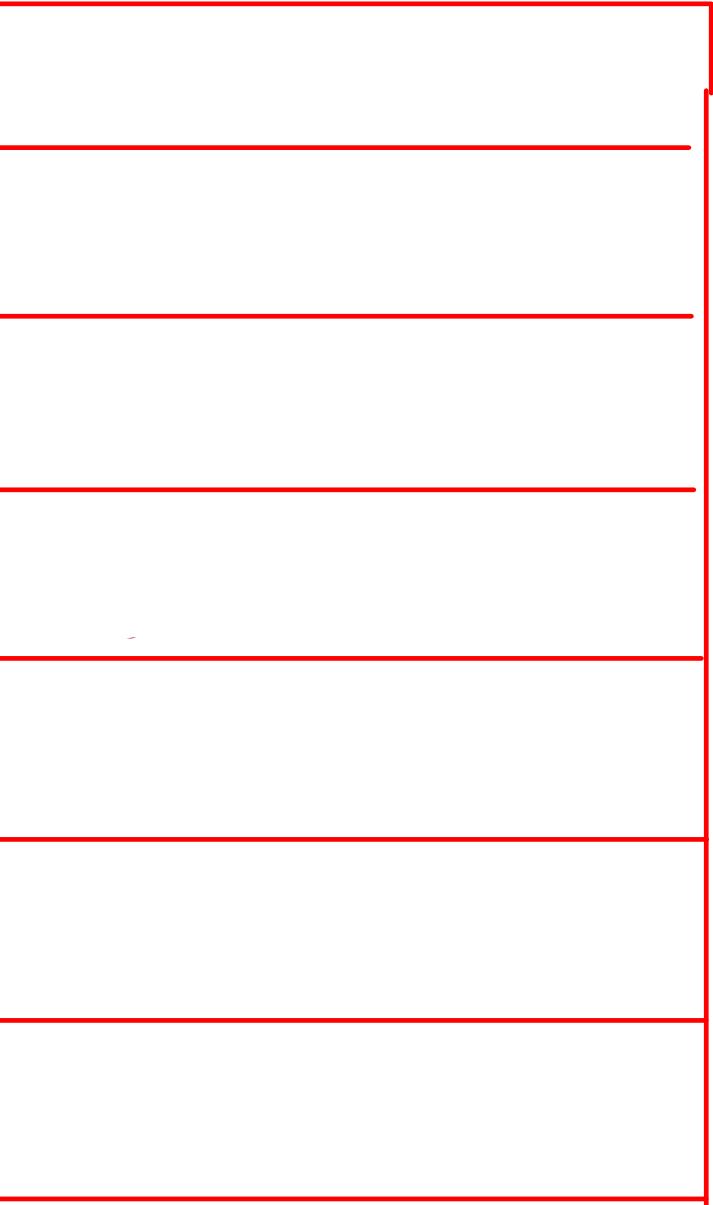
System.out.println(n);

- ④ Base Case
- ↳ To stop recursion
 - ↳ Stack overflow / memory limit exceeded

low-level thinking

function call stack

```
void printInc(int n) {  
    Base case → ① if(n == 0) return;  
    Fact → ① printInc(n-1);  
    Meeting Expectation → ② System.out.println(n);  
  
    main() {  
        printInc(5);  
    }
```



```
public static void main(String[] args) throws Exception {
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();
    printIncreasing(n);
}

public static void printIncreasing(int n){
    if(n == 0){
        // Base Case
        return;
    }

    // 1. Faith : 1, 2, .. n-1
    printIncreasing(n - 1);

    // 2. Meet Expectation with Faith
    System.out.println(n);
}
```

Tail Recursion

work is being done while returning

Q2) Point Decreasing

① pointDecreasing(int n) :->

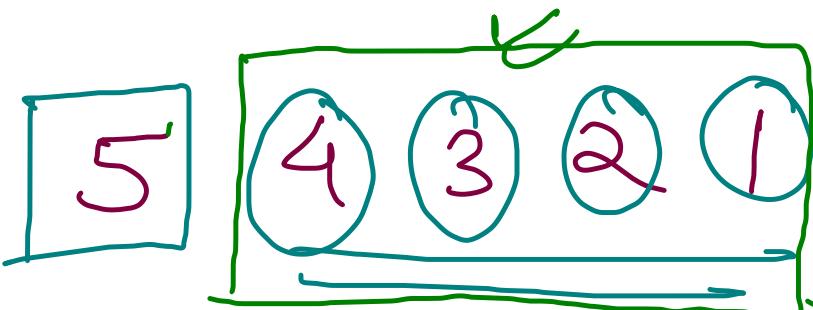
Expectation

$n, n-1, n-2, \dots, 1$

② pointDecreasing(n-1) :->

$n-1, n-2, n-3, \dots, 1$

③ System.out.println(b) :-> Meeting Expectation
Should be
before the
call.



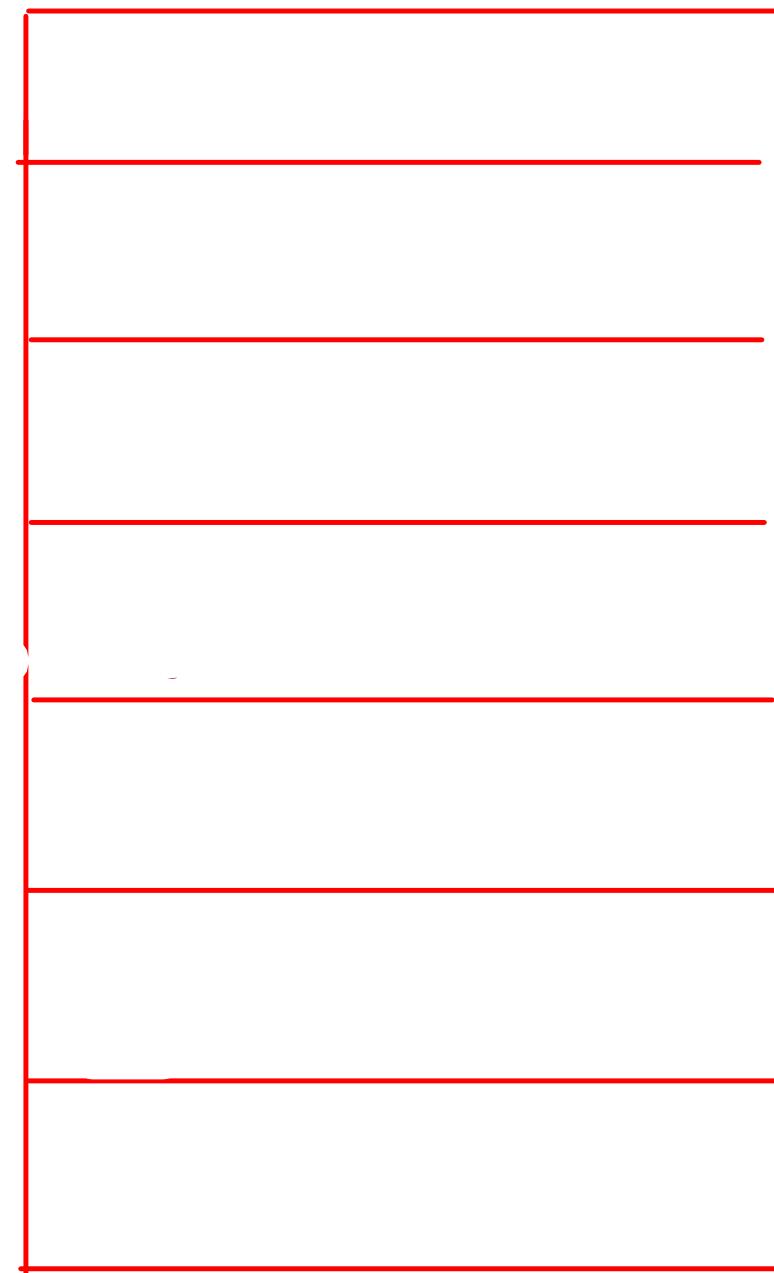
$n=5$ 5 4 3 2 1

void printDec (int n) {
 ① if ($n == 0$) return;
 System.out.println (n); } → Border

② printDec (n-1);

3

~~head recursion~~
work is done before
calling functions



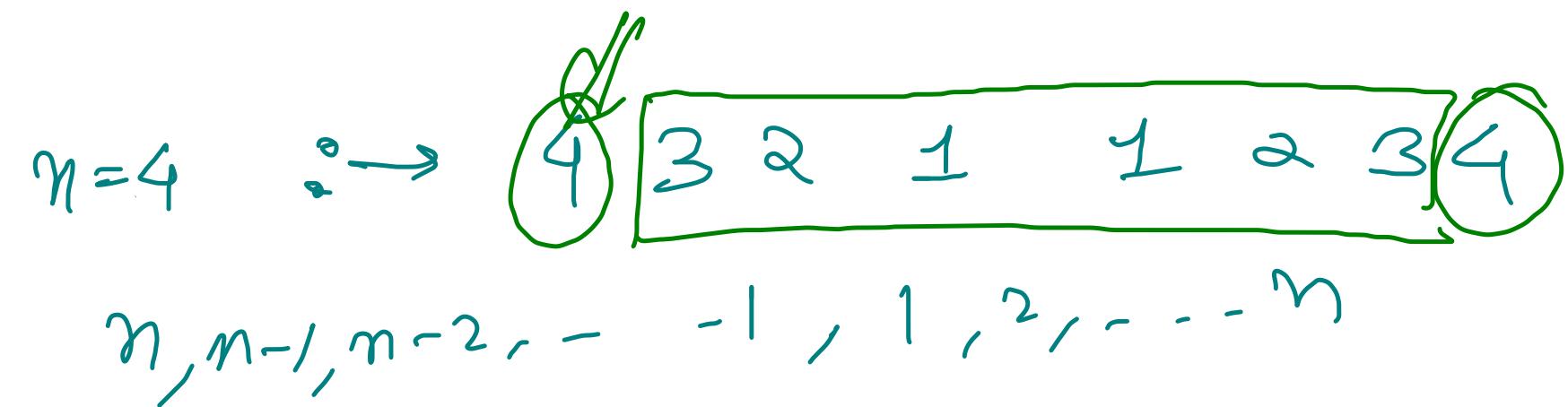
5, 4, 3, 2, 1

Q) Print Increasing Decreasing

High-Level Thinking

① Expectation

PDI(int n) : →



② Fault

PDI(n-1) : →

$n=3 \rightarrow$ 3 2 1 1 2 3

$n-1, n-2, n-3, \dots, 1, 1, 2, 3, \dots, n-2, n-1$

③ Meeting Expectation

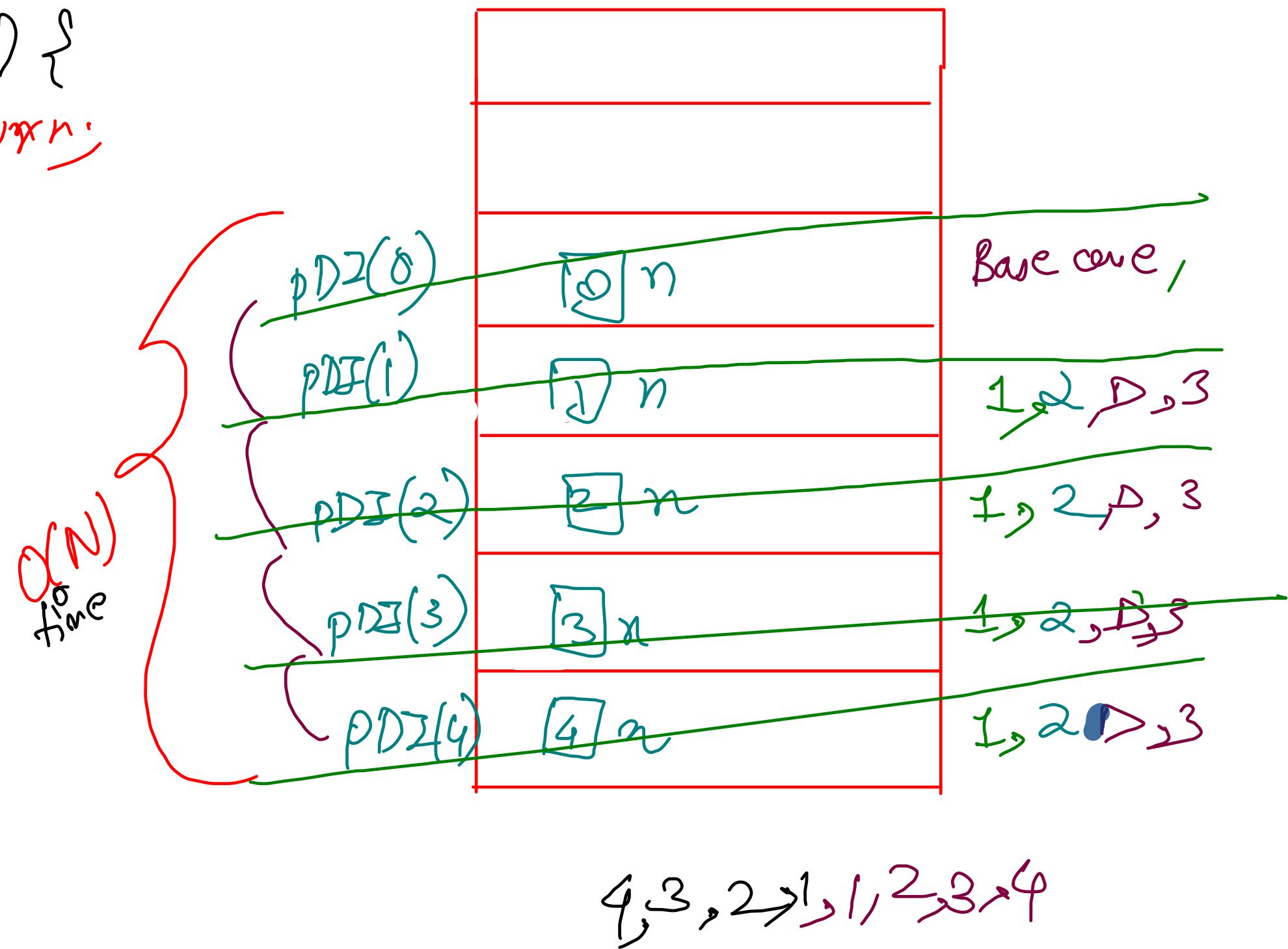
SPO(n) → Preorder
Postorder

$$N = 4 \quad 4, 3, 2, 1, 1, 2, 3, 4$$

```

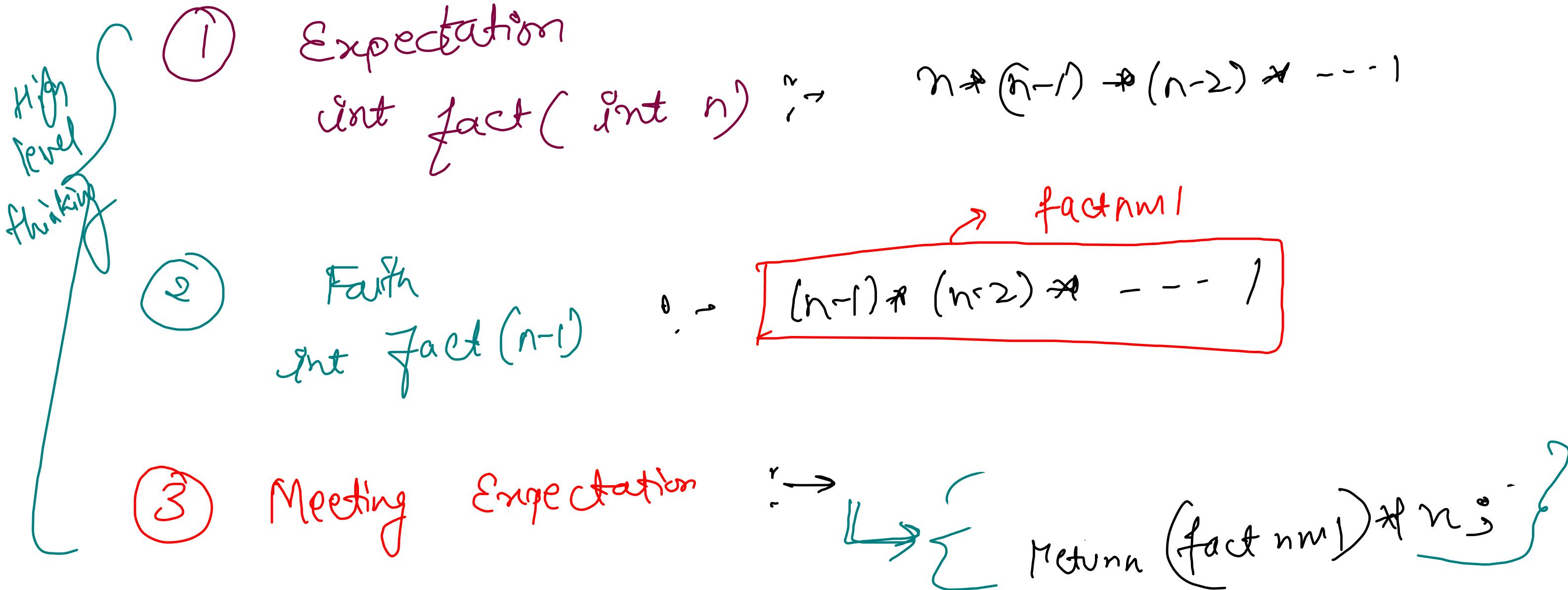
void PDI( int n ) {
    if ( n == 0 ) return;
    Preorder{ ① } Sys0(n);
    father{ ② } PDI( n-1 );
    Postorder{ ③ } Sys0(n);
}
main() {
    PDI( n );
}

```



Factorial

$$5! = 5 * 4 * 3 * 2 * 1$$



low-level thinking

$$N=4 \quad 4 \times 3 \times 2 \times 1$$

```
int fact (int n) {
```

Base case ① if ($n == 0$) return 1;

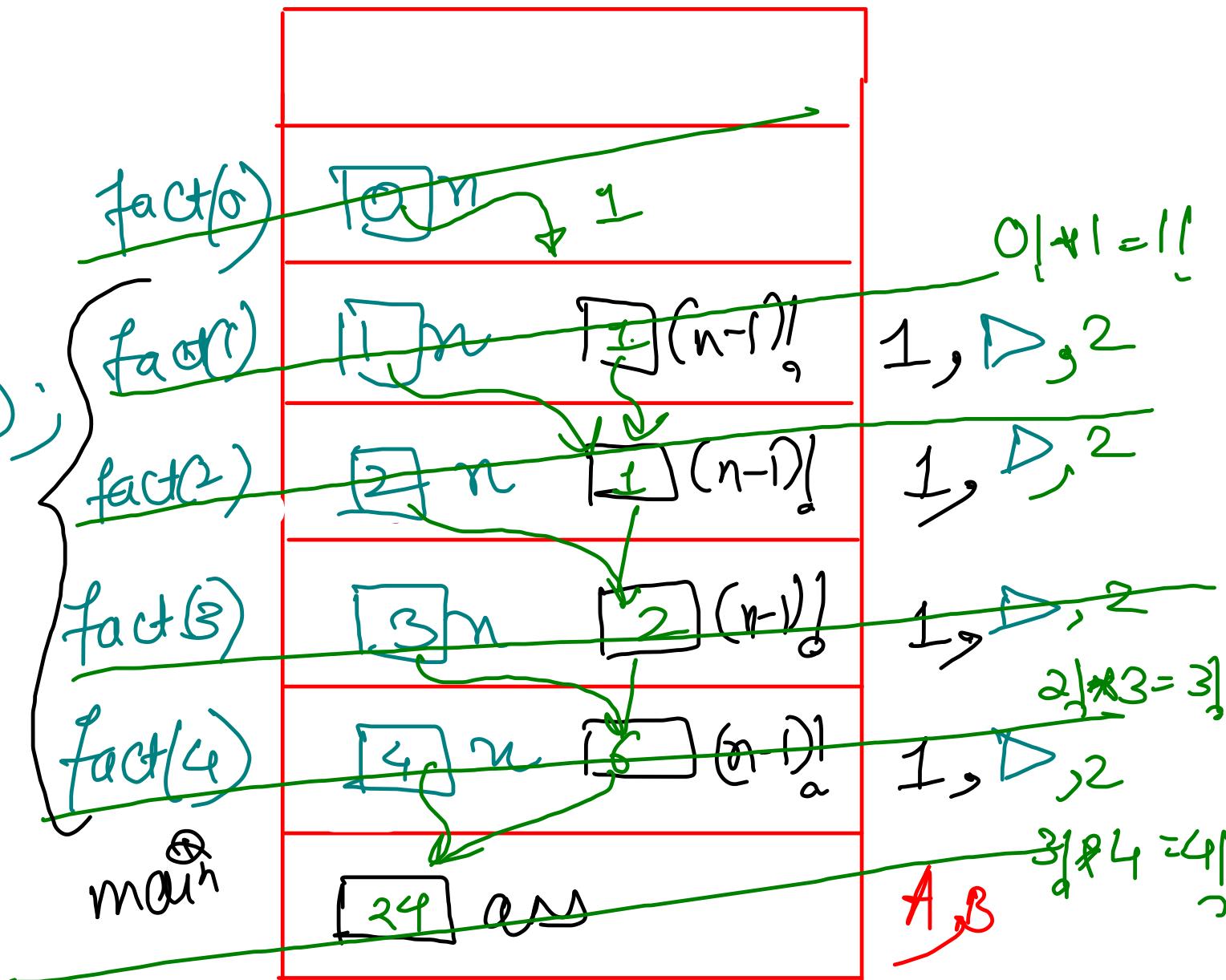
faith { ② int factm1 = fact(n-1);

Meeting
Expectation
3 }
return factm1 * n;

```
main() {
```

A) int ans = fact(4);
B) System.out.println(ans);

Call
 $n = N$
 $O(N)$



24

Power - Linear $\Rightarrow O(N)$

$$a^b = a * a * a * \dots b \text{ times}$$

$$3^5 = 3 * 3 * 3 * 3 * 3$$

① Expectation
power(x, n) $\xrightarrow{\text{constant variable}}$

$$\{ 3 * 3 * 3 * 3 * 3 \} \\ x * x * x * \dots n \text{ times}$$

② Faith
power($x, n-1$) $\xrightarrow{\text{}}$

$$\{ 3 * 3 * 3 * 3 \} \\ x * x * \dots (n-1) \text{ times}$$

③ Meeting Expectation $\xrightarrow{\text{return}}$

return

pxnml * x^j
 $3^4 * 3 = 3^5$

Low-level Thinking

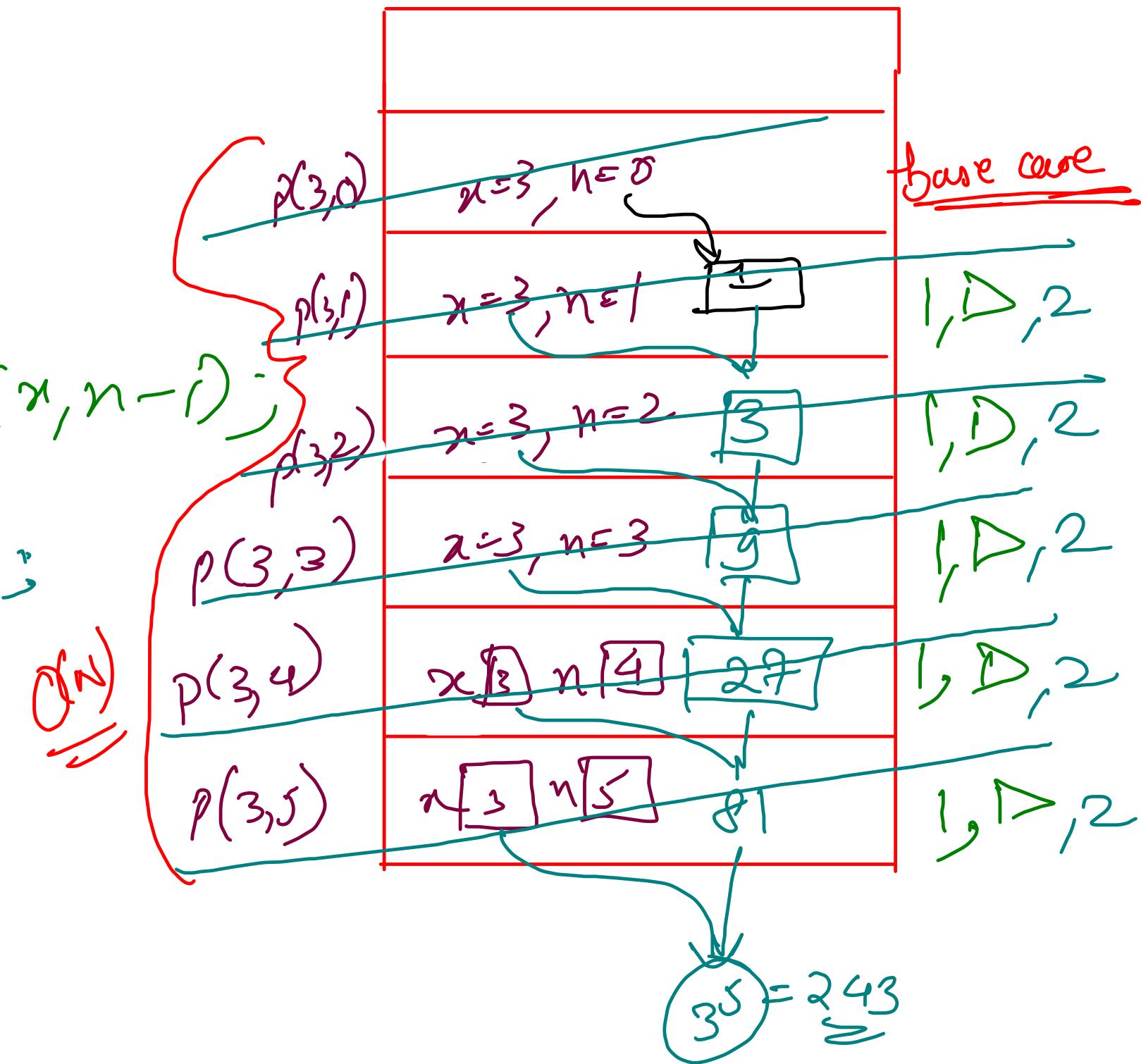
```
int power(int x, int n) {
```

① if ($n == 0$) return 1;

② int pxnml = power(x, n-1);

③ return pxnml * x;

3



```
public static void main(String[] args) throws Exception {
    // write your code here
    Scanner scn = new Scanner(System.in);
    int x = scn.nextInt();
    int n = scn.nextInt();
    int ans = power(x, n);
    System.out.println(ans);
}

public static int power(int x, int n){
    if(n == 0) return 1; //  $x^0 = 1$ 
    int pxnm1 = power(x, n - 1); // faith
    return pxnm1 * x; // meeting expectation
}
```

Power - logarithmic

$$x = \cancel{x}^{\frac{n}{2}} + \cancel{x}^{\frac{n}{2}}$$

~~n = even~~

$$a^n = a * a * a * \dots \text{ b times}$$

$$3^5 = 3 * 3 * 3 * 3 * 3$$

$$3^8 = 3^4 * 3^4$$

$$\cancel{n \text{ odd}} \quad 3^9 = 3^4 * 3^4 * 3$$

$$a^{\frac{n}{2}} * a^{\frac{n}{2}}$$

① Expectation
 power(x, n) $\stackrel{\substack{\text{constant} \\ \text{variance}}}{\rightarrow}$

$$\{ 3 * 3 * 3 * 3 * 3 \}$$

$$x * x * x * \dots \text{ n times}$$

② fair
 power(x, n_k) $\stackrel{\substack{\text{?} \\ \vdash}}{\rightarrow}$

$$\overbrace{x * x * \dots}^{\text{n/2 times}} \xrightarrow{\text{pxnbyz}}$$

③ Meeting Expectation $\stackrel{\substack{\text{?} \\ \vdash}}{\rightarrow}$

return $\boxed{\begin{array}{l} \text{pxnbyz} \Rightarrow \text{pxnbyz} \\ \text{b} * x \text{ if } (n/2 = 1) \end{array}}$

Low-level Thinking

```
int power(int x, int n) {
```

① if ($n == 0$) return 1;

② int pxnby2 = power(x, $n/2$);

if ($n \% 2 == 1$) return $x * pxnby2 * pxnby2$;
else return $pxnby2 * pxnby2$;

3

$$3^2 = 3^1 * 3^1$$

$$3^4 = 3^2 * 3^2$$

$$3^9 = 3^4 * 3^4 * 3$$

$$p(3, 0)$$

$$q(3, 1)$$

$$p(3, 2)$$

$$p(3, 4)$$

$$p(3, 9)$$

$$n = 9$$

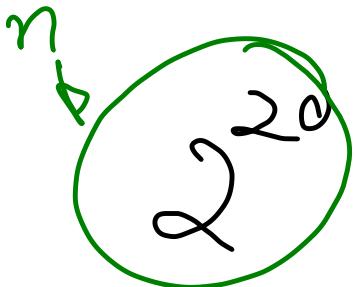
d(log)

$$3^9 = 81 * 81 * 3$$

```
public static void main(String[] args) throws Exception {  
    Scanner scn = new Scanner(System.in);  
    int x = scn.nextInt();  
    int n = scn.nextInt();  
    int ans = power(x, n);  
    System.out.println(ans);  
}
```

```
public static int power(int x, int n){  
    if(n == 0) return 1;  
  
    int pxnby2 = power(x, n/2);  
  
    if(n % 2 == 1) return pxnby2 * pxnby2 * x;  
    else return pxnby2 * pxnby2;  
}
```

$$n = 2^x \Rightarrow \text{no of calls} = x$$



$$\log n = \log(2^x)$$

$$O(n) \Rightarrow 1048576$$

$$\log n = x \Rightarrow \log_2 2$$

$$O(\log_2 n) \Rightarrow O(\log_2 2^{20}) \\ = O(20)$$

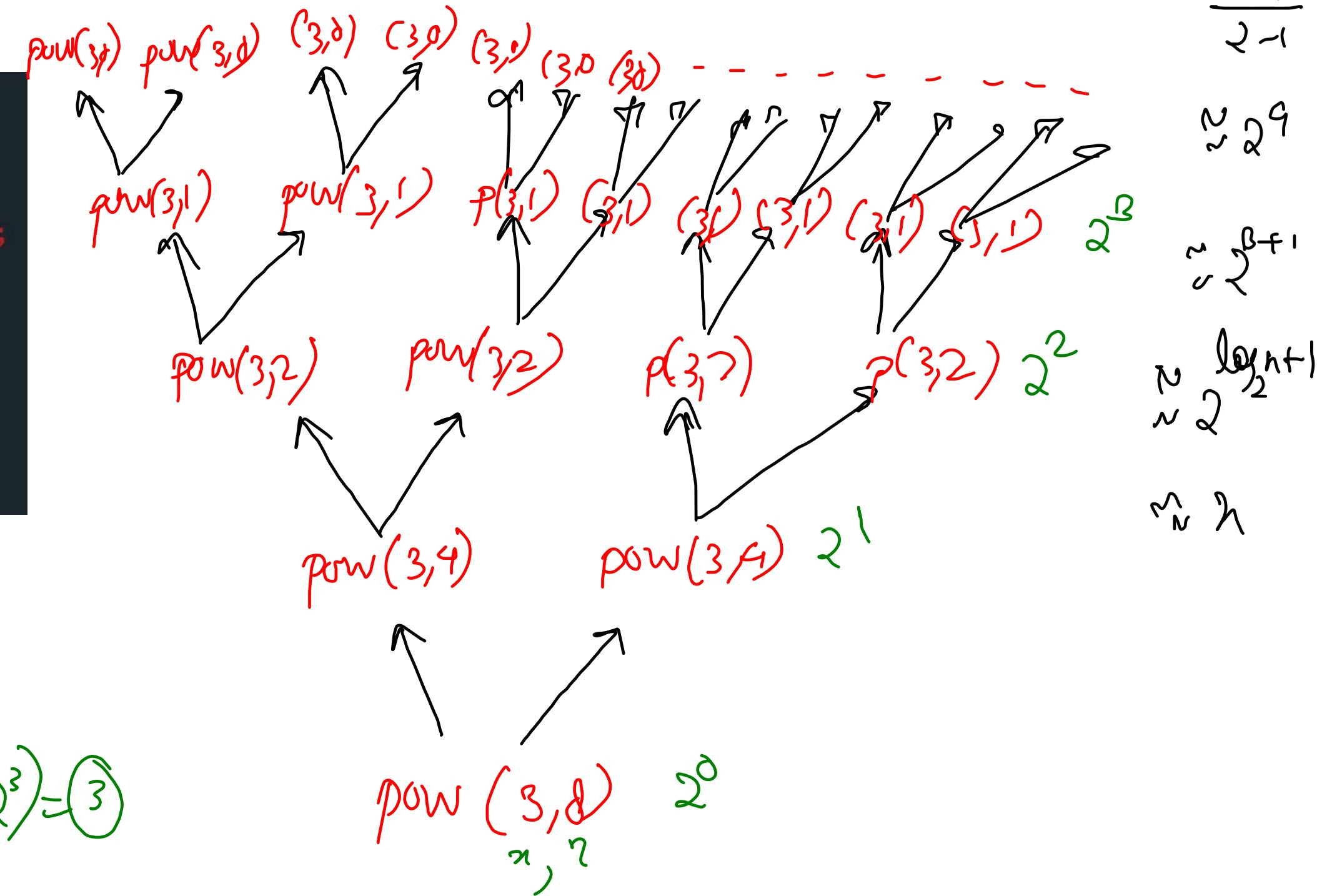
$$\log n = x \Rightarrow \text{no of call} \Rightarrow O(\log_2 n) \text{ constant}$$

Today's Questions

- ① Power - 3rd method
- ② Print zigzag
- ③ Tower of Hanoi
- ④ Display Array
- ⑤ Display Array - Reverse
- ⑥ Max of Array

Power \rightarrow IIIrd method

```
public static int power(int x, int n) {
    if(n == 0){
        return 1;
    }
    int xpn = power(x, n/2) * power(x, n/2);
    if(n % 2 == 1){
        xpn = xpn * x;
    }
    return xpn;
}
```



$$n = 8$$

$$\mathcal{O}(\log_2 n) = \mathcal{O}(\log_2 2^3) = 3$$

(Q) Print ZigZag

① Expectation :-

printZigzag(int n)

3 2 1 1 1 2 1 1 1 2 3 2 1 1 1 0 2 1 1 1 0 2 1 1 1 2 3

②

Faith :- printZigzag($n-1$)

2 1 1 2 1 1 2

③ Meet Expectation :-

sys(n)

printZigzag($n-1$)

sys(n)

printZigzag($n-1$)

sys(n)

printZigzag(1)
1 1 1

printZigzag(0)
return ;

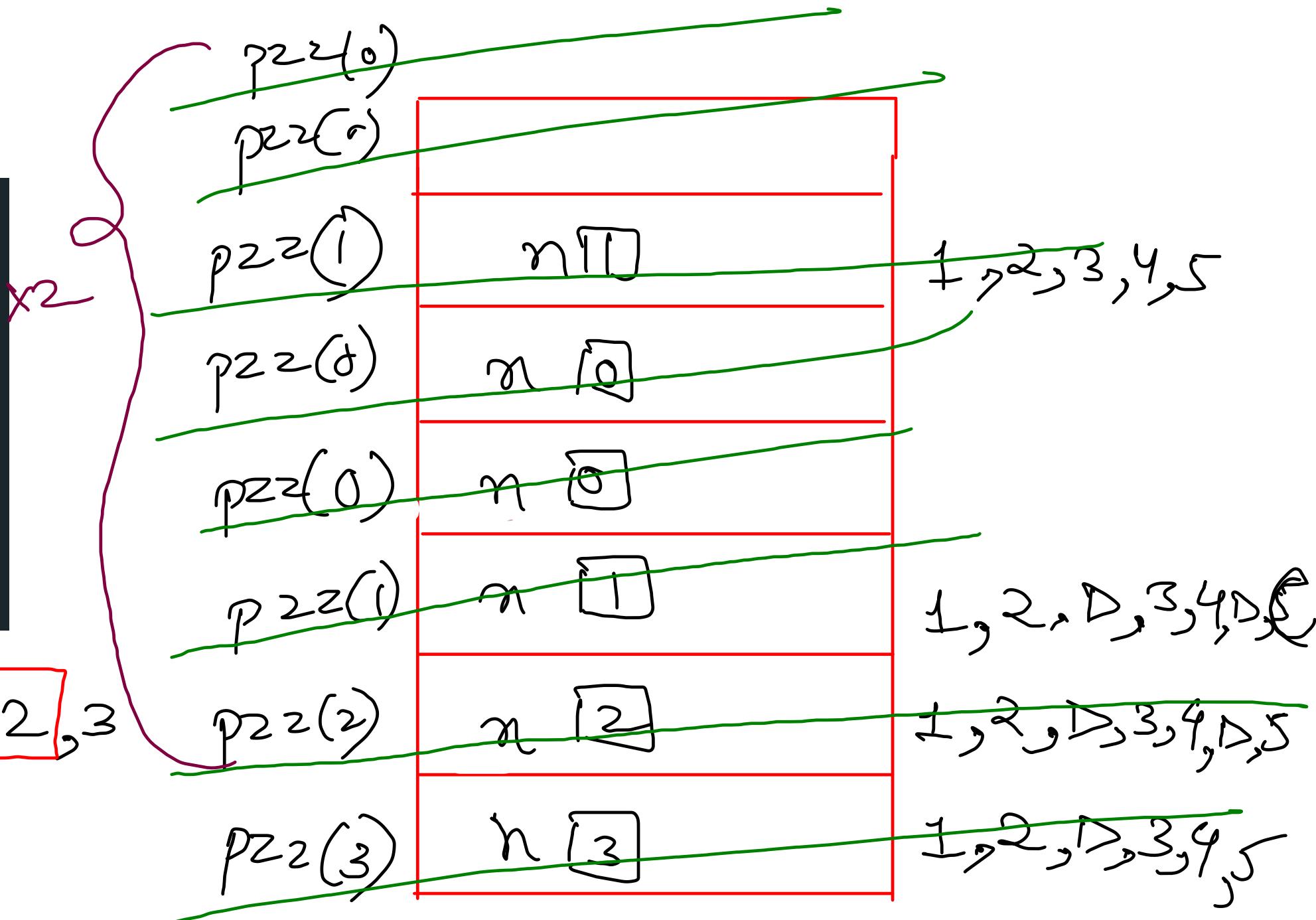
```

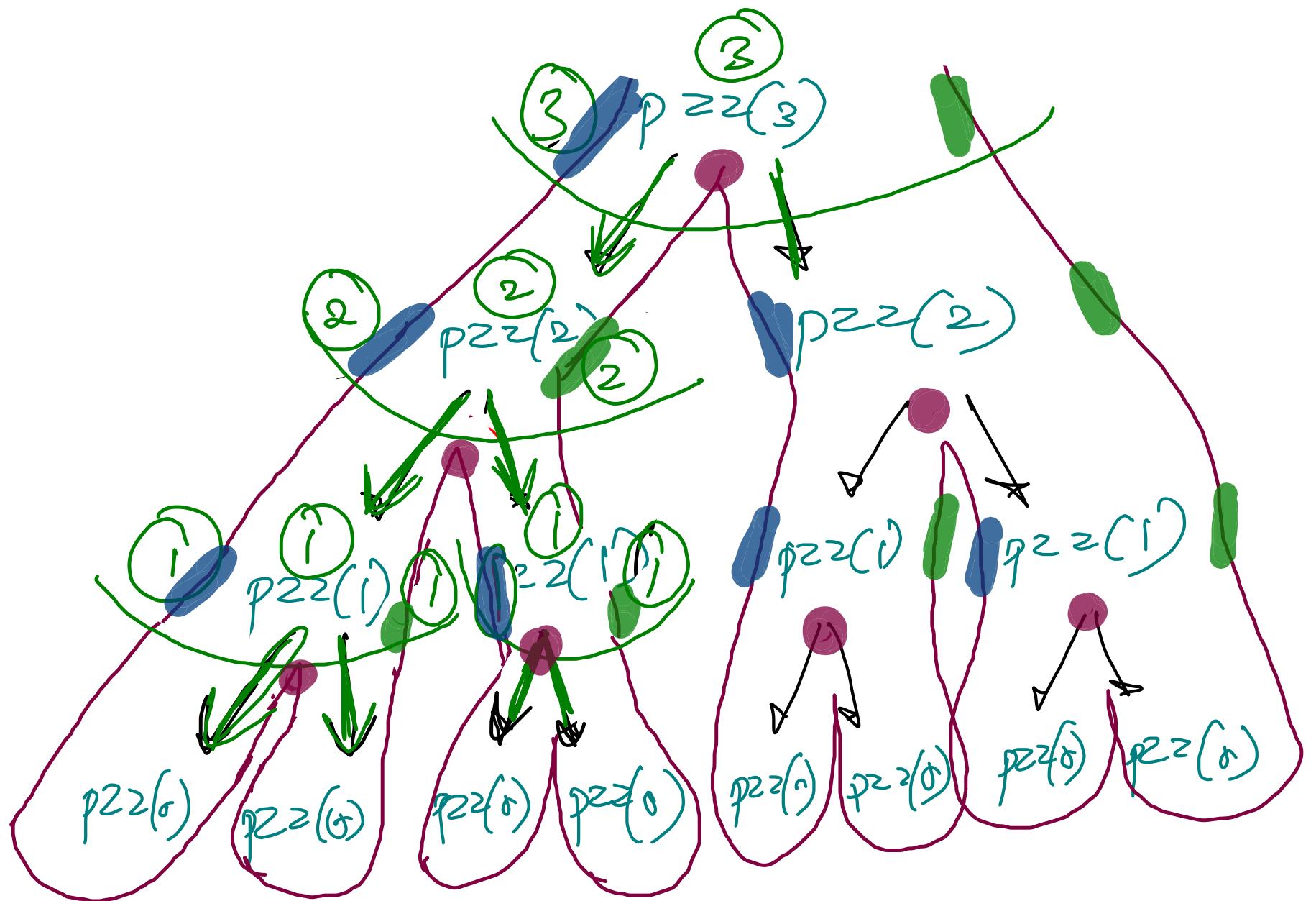
public static void pzz(int n){
    if(n == 0) return;
    System.out.print(n + " ");
    pzz(n - 1);
    System.out.print(n + " ");
    pzz(n - 1);
    System.out.print(n + " ");
}

```

3, 2, 1, 1, 1, 2, 1, 1, 1, 2, 3

2, 1, 1, 1, 2, 1, 1, 1, 2, 3





```

public static void pzz(int n){
    if(n == 0) return;
    Precord → System.out.print(n + " ");
    Left → pzz(n - 1);
    Inorder → System.out.print(n + " ");
    Right → pzz(n - 1);
    Postrecord ← System.out.print(n + " ");
}
  
```

Depth first Search

↓
Recursive free/
euler free

Recursion

c ↗
(calls)^{height} + {preorder + inorder + postorder} * height

eg Point zigzag

$$2^n + \{k+k+k\} * n = 2^n + 3kn$$

$O(2^n)$

Recursion & Arrays

① Display Array

```
public static void displayArr(int[] arr, int idx){  
    ...  
}
```

③ Meeting Expectation

→ Preorder

sys(arr[idx])

display(arr, idx + 1)

① Expectation
{ 10, 20, 30, 40, 50 }

displayArr(arr, 0)

② Faith

display(arr, 1)

{ 20, 30, 40, 50 }

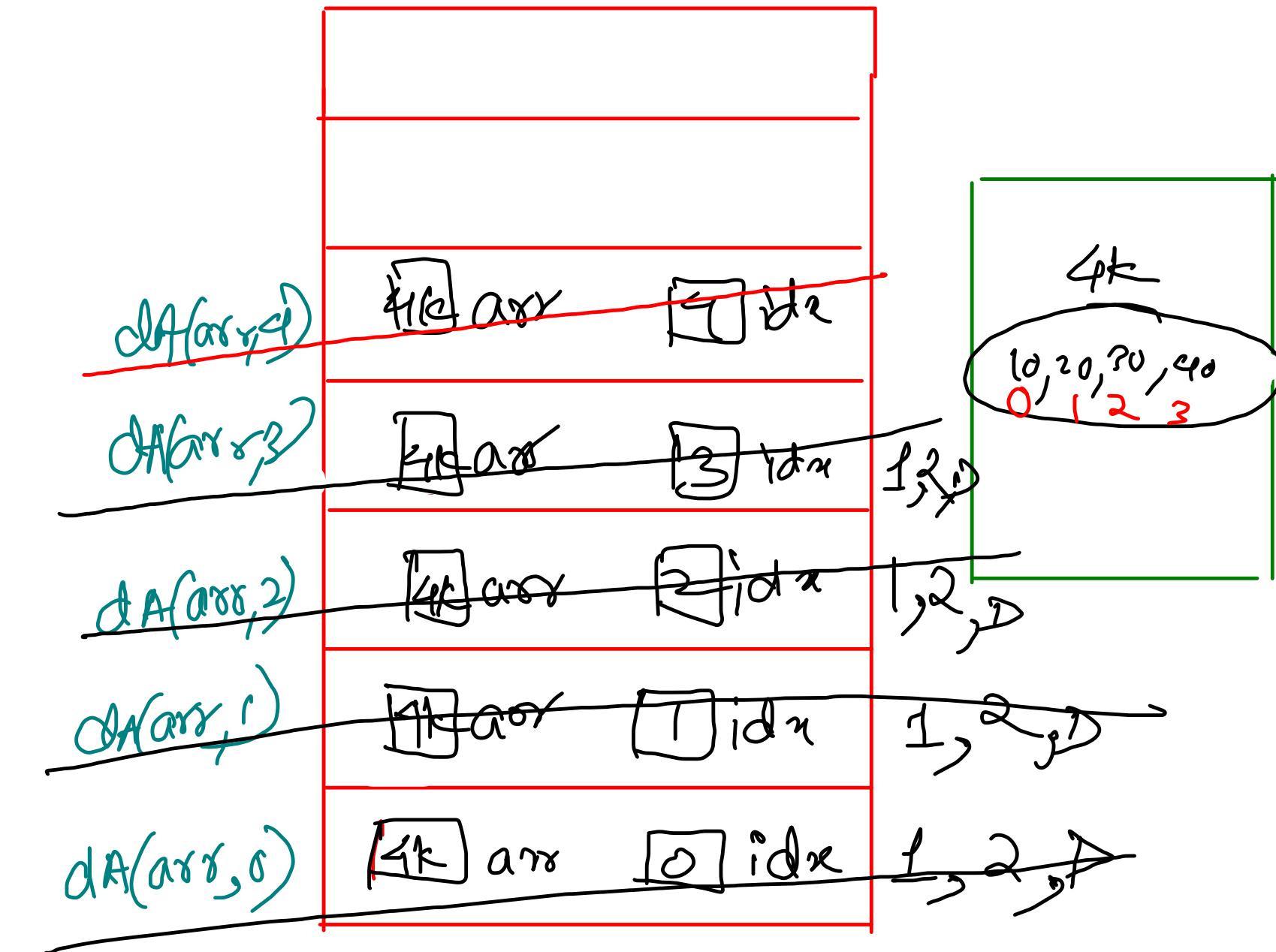
④ Base case
if(idx == arr.length)
 return

```

public static void displayArr(int[] arr, int idx){
    if(idx == arr.length) return;
    // preorder -> Meeting Expectation
    System.out.println(arr[idx]); {bordered}
    // faith
    displayArr(arr, idx + 1);
}

```

10, 20, 30, 40



Q) Maximum In an Array

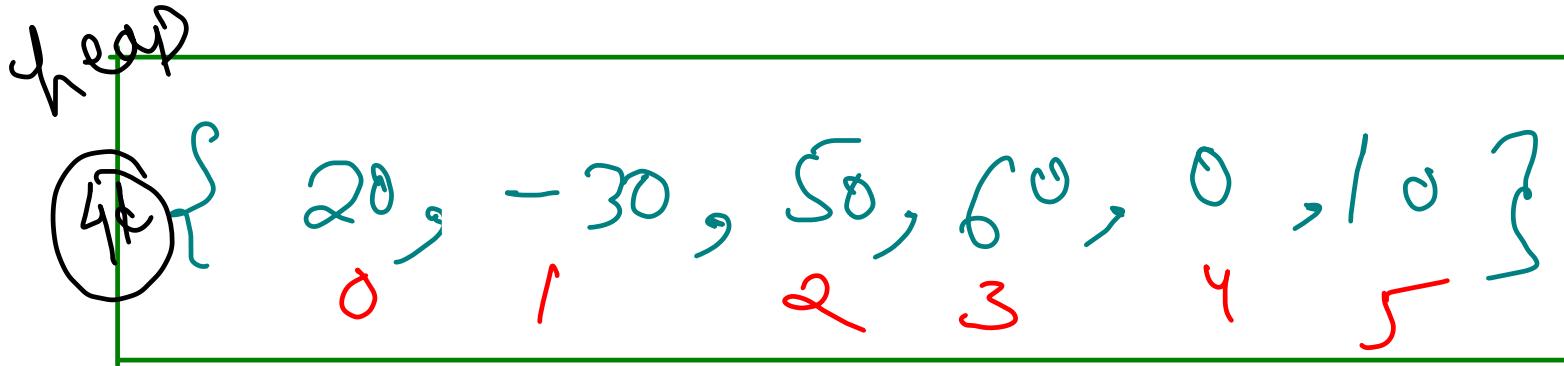
{ 20, -30, 50, 60, 0, 10, 40 }

① Expectation

int max of Array(arr, l) \Rightarrow max arr[0 : l-1] \Rightarrow 60

② Faith int max of Arr(arr, l) \Rightarrow max arr[1 : l-1]

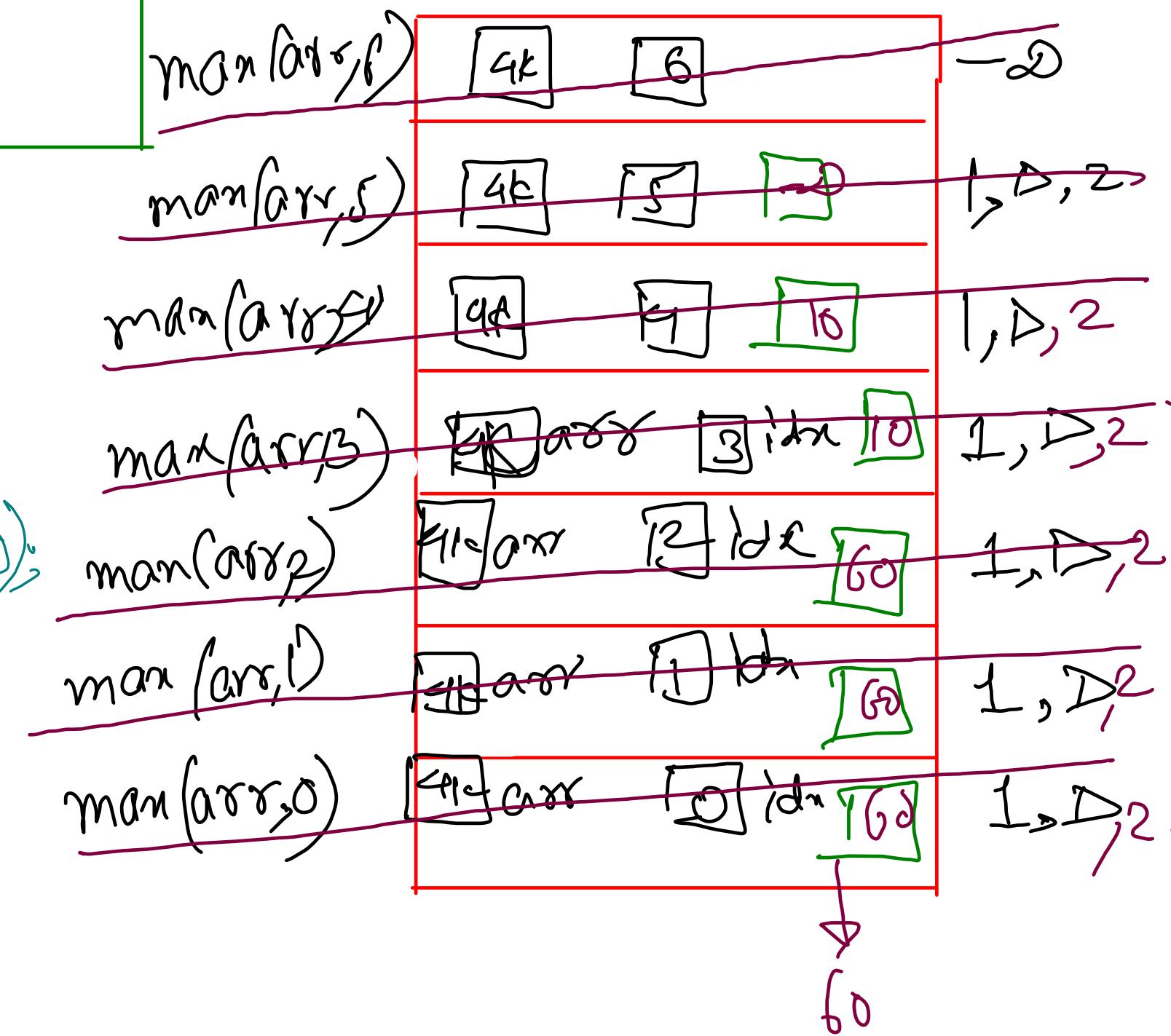
③ Meeting Expectation \Rightarrow return max(arr[i:l], maxall)



```

int maxOfArr(int [] arr, int idn) {
    ⑥ if( idn == arr.length ) return -∞;
    ① int maxTemp = max(arr, idn+1);
    ② return max(maxTemp, arr[idn]);
}

```



Addition

$$x + ? = x$$

Subtraction

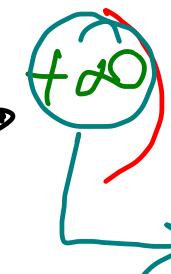
$$x - ? = x$$

Multiplication

$$x * ? = x$$

Minimum

$$\min(x, +\infty) = x$$



Integer. MAX VALUE

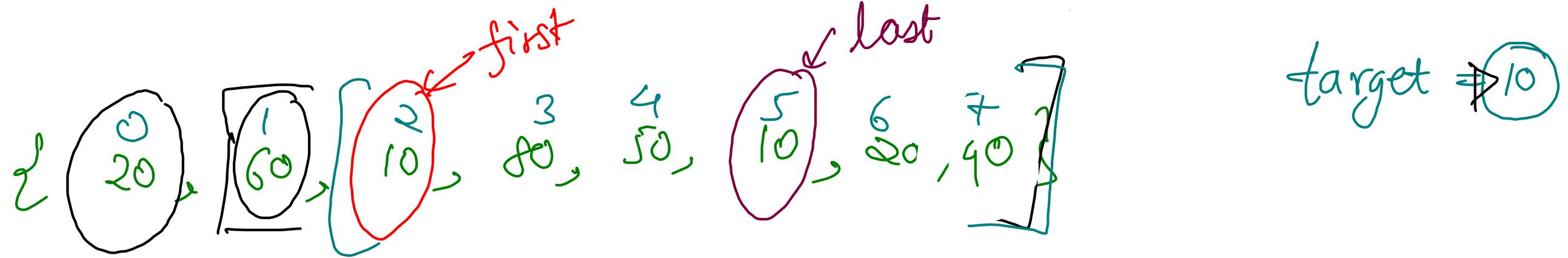
$$+2^{31} - 1$$

Maximum

$$\min(x, -\infty) = x$$

Integer. MIN VALUE

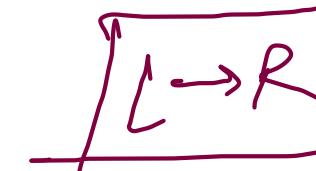
$$-2^{31}$$



First Index

$fI(\text{arr}, c) \rightarrow 2$

faith



temp = $fI(\text{arr}, \text{idn}+1)$

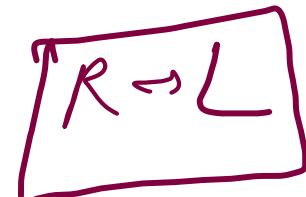
Meeting End

if ($\text{arr}[\text{idn}] == \text{target}$)
 T → return $\text{idn};$
 F → return temp;

Last Index

$fI(\text{arr}, \text{a.l}-1)$

faith



temp = $fI(\text{arr}, \text{idn}-1)$

Meeting End

if ($\text{arr}[\text{idn}] == \text{target}$)
 T → return $\text{idn};$
 F → return temp;

```

public static int firstIndex(int[] arr, int idx, int x){
    if(idx == arr.length) return -1;

    ① if(arr[idx] == x){
        // meeting expectation
        return idx;
    } else {
        // faith
        return firstIndex(arr, idx + 1, x);
    }
}

```

{ 20 50 10 80 50, 10, 20, 40 }

```

public static int lastIndex(int[] arr, int idx, int x){
    if(idx == -1) return -1; // search unsuccessful

    if(arr[idx] == x){
        return idx;
    } else {
        return lastIndex(arr, idx - 1, x);
    }
}

```

