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

(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

(5) Backtracking

- Flood Fill
- Target Sum Subsets
- N Queens
- Knights Tour

Principle of mathematical Induction (PMI)

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

① Trivial Case

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

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

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

③ To prove $\sum_{i=1}^{k+1} i = (1+2+\cdots k)+(k+1) = \frac{k*(k+1)}{2} + (k+1)$

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

if $n = k+1$ LHS = RHS

Recursion {High-level thinking}

n<5

1 2 3 4 (5)

(1) Recursive function :→ Expectation

void printInc(int n) :→ 1, 2, 3, - ... n

(2) Recursive fn :→ Smaller values :→ faith (call)

void printInc(n-1) :→ 1, 2, 3, - ... n-1

1 2 3 4

(3) Meeting Expectation with faith

System.out.println(n);

(4) Base Case

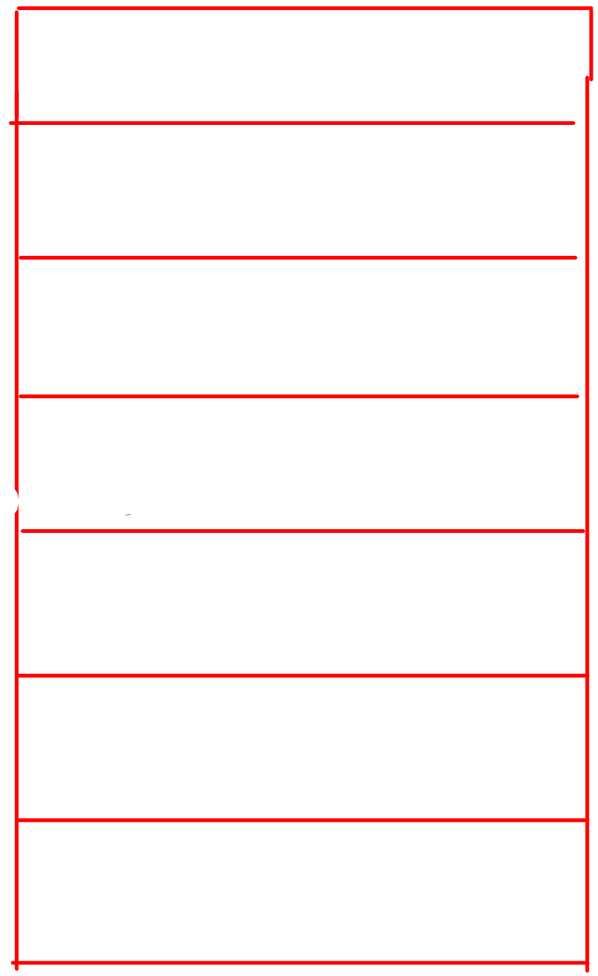
↳ To stop recursion

① → Stack overflow / memory limit exceeded

how- level thinking

function call stack

```
void printInc(int n) {
Base case → (3) if(n==0) return;
Faith → (1) printInc(n-1);
Meeting Expectation in faith → (2) System.out.println(n);
}

main() {
    printInc(5);
}
```



```java
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

} → post work

Q2) Print Decreasing

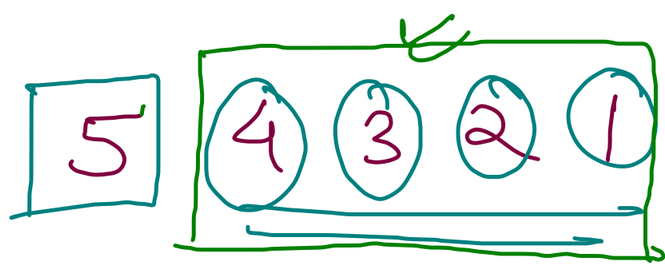
(1) printDecreasing(int n) :→ Expectation

n, n-1, n-2, ---- 1

(2) printDecreasing(n-1) :→ faith

n-1, n-2, n-3, --- 1

(3) System.out.println(n) :→ Meeting Expectation should be before the call.



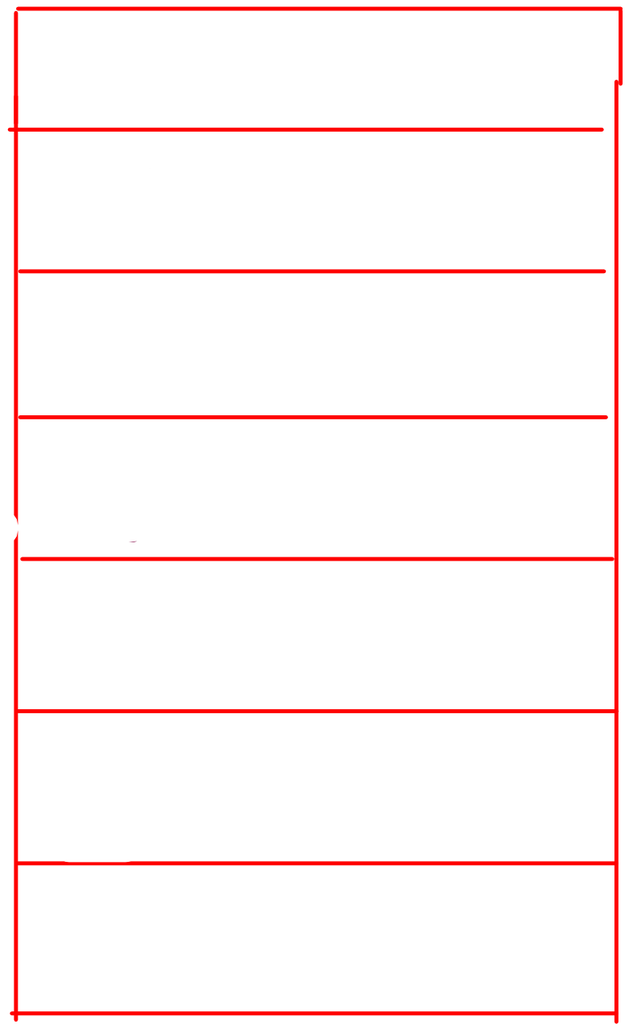
n=5 5 4 3 2 1

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

① ② ③ ④

① ⇒ Preorder

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) :→ n, n-1, n-2, -- -1, 1, 2, --- n

n=4 :→ 4 3 2 1 1 2 3 4

② Faith

PDI(n-1) :→ n-1, n-2, n-3, --- 1, 1, 2, 3, --- n-2, n-1

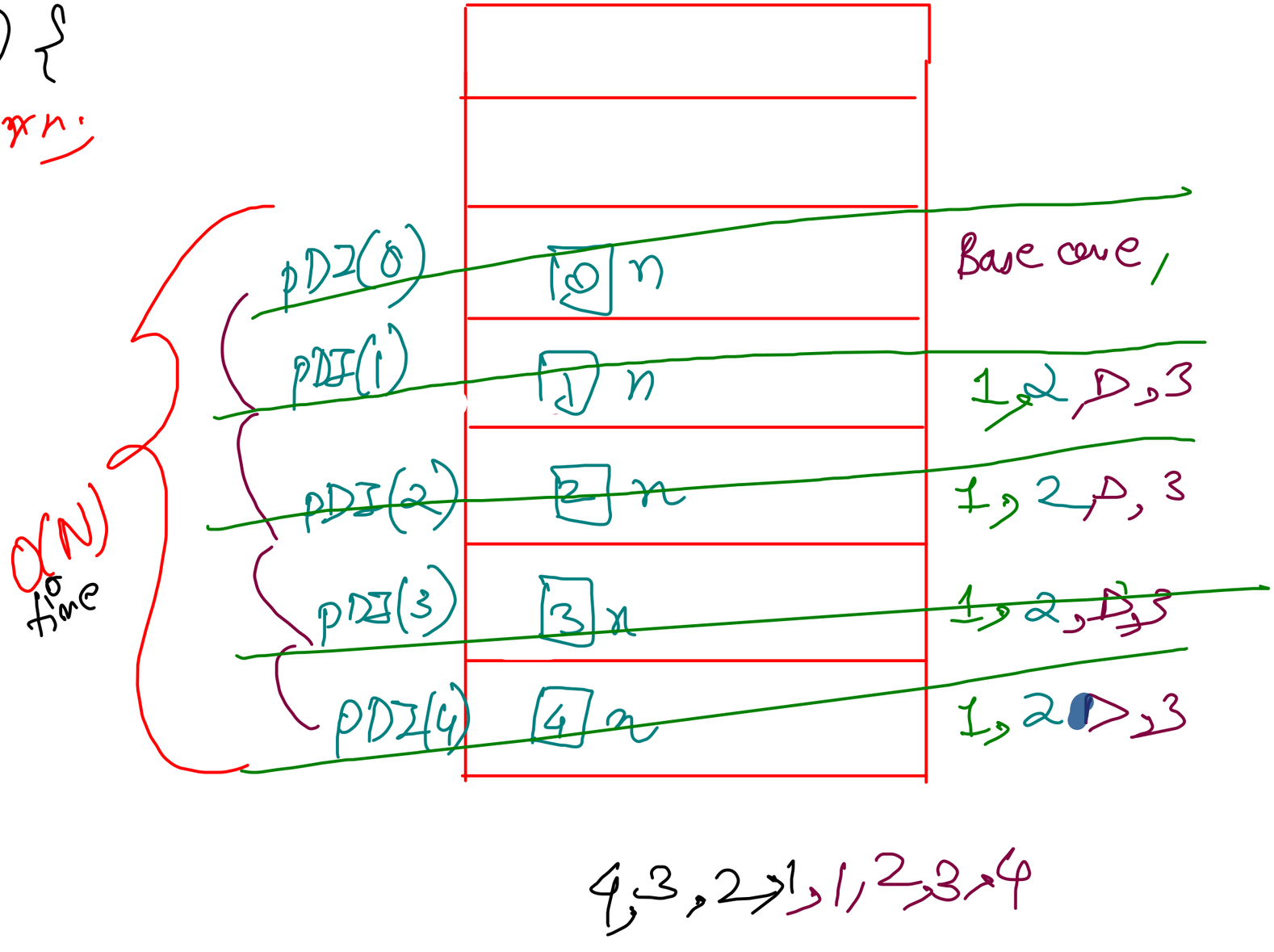
n=3 :→ 3 2 1 1 2 3

③ Meeting Expectation

Sys0(n) → Preorder
Sys0(n) → Postorder

N = 4 4,3,2,1,1,2,3,4

```
void pDI( int n) {
  ①  if(n==0) return;
Preorder{ ②  Syso(n);
failure{  ③  pDI(n-1);
Postorder{④  Syso(n);
}

main() {
    pDI(n);
}
```



4,3,2,1,1,2,3,4

Factorial

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

High level thinking

1. Expectation

int fact(int n) :→ $n * (n-1) * (n-2) * \cdots 1$

2. Faith

int fact(n-1) :→ $(n-1) * (n-2) * \cdots 1$ → factnm1

3. Meeting Expectation :→

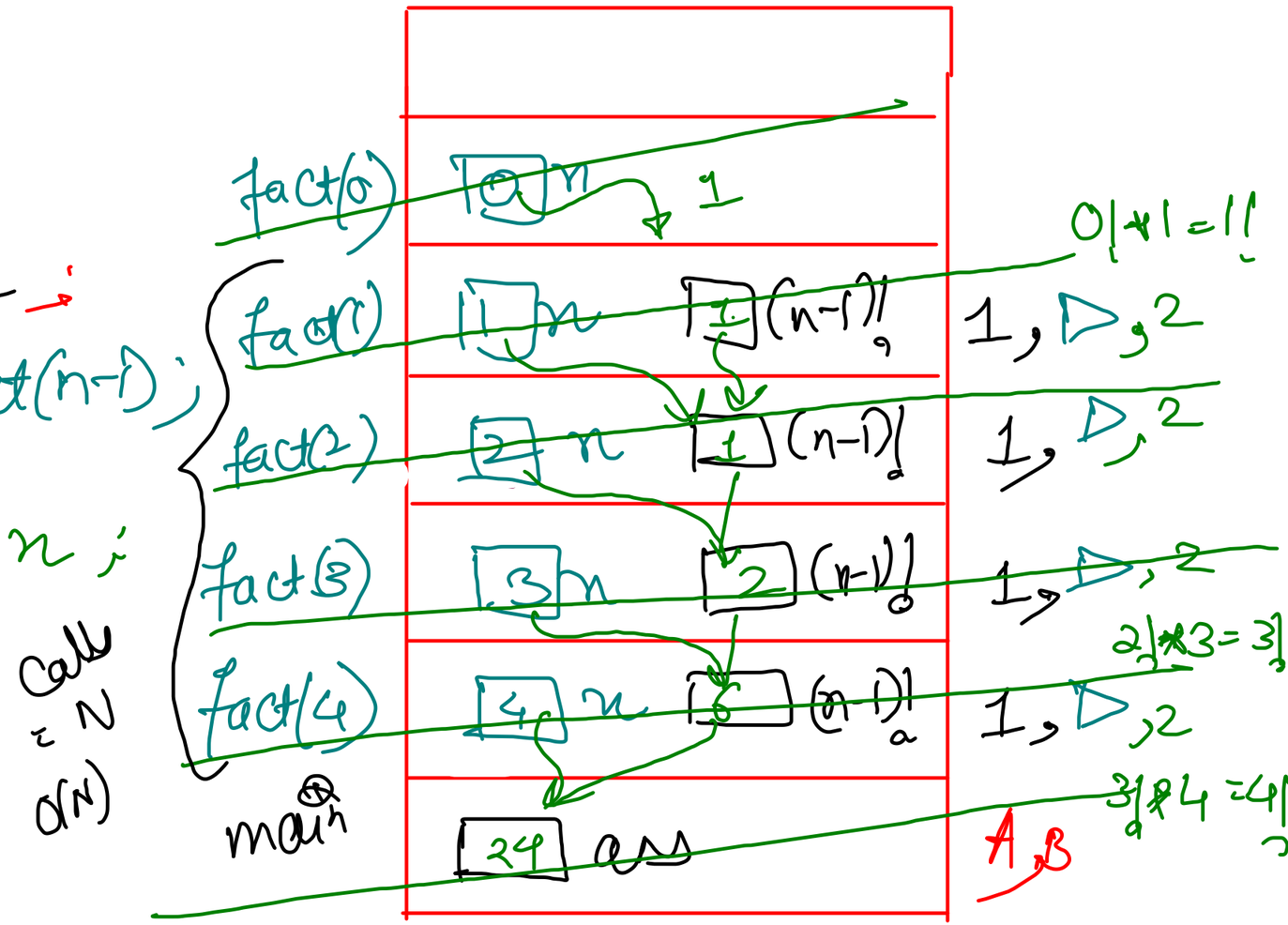
```
{ return (fact nm1) * n; }
```

How-Level thinking

N = 4 4×3×2×1

```
int fact (int n) {
Base case (0)  if (n==0) return 1;
faith {   (1)  int factnm1 = fact(n-1);
Meeting {
Expectation (2)  return factnm1 * n;
}
```

```
main() {
(A) int ans = fact(n);
(B)   Syso (ans);
}
```



24

Power - Linear $\Rightarrow O(N)$

$a^b = a * a * a * \cdots$ b times

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

(1) Expectation

constant, variable

power(x, n) :→ $\{ 3*3*3*3*3 \}$

$x * x * x * \cdots$ n times

(2) Faith

power(x, $n-1$) :→ $\{ 3*3*3*3 \}$

$x * x * \cdots$ $(n-1)$ times

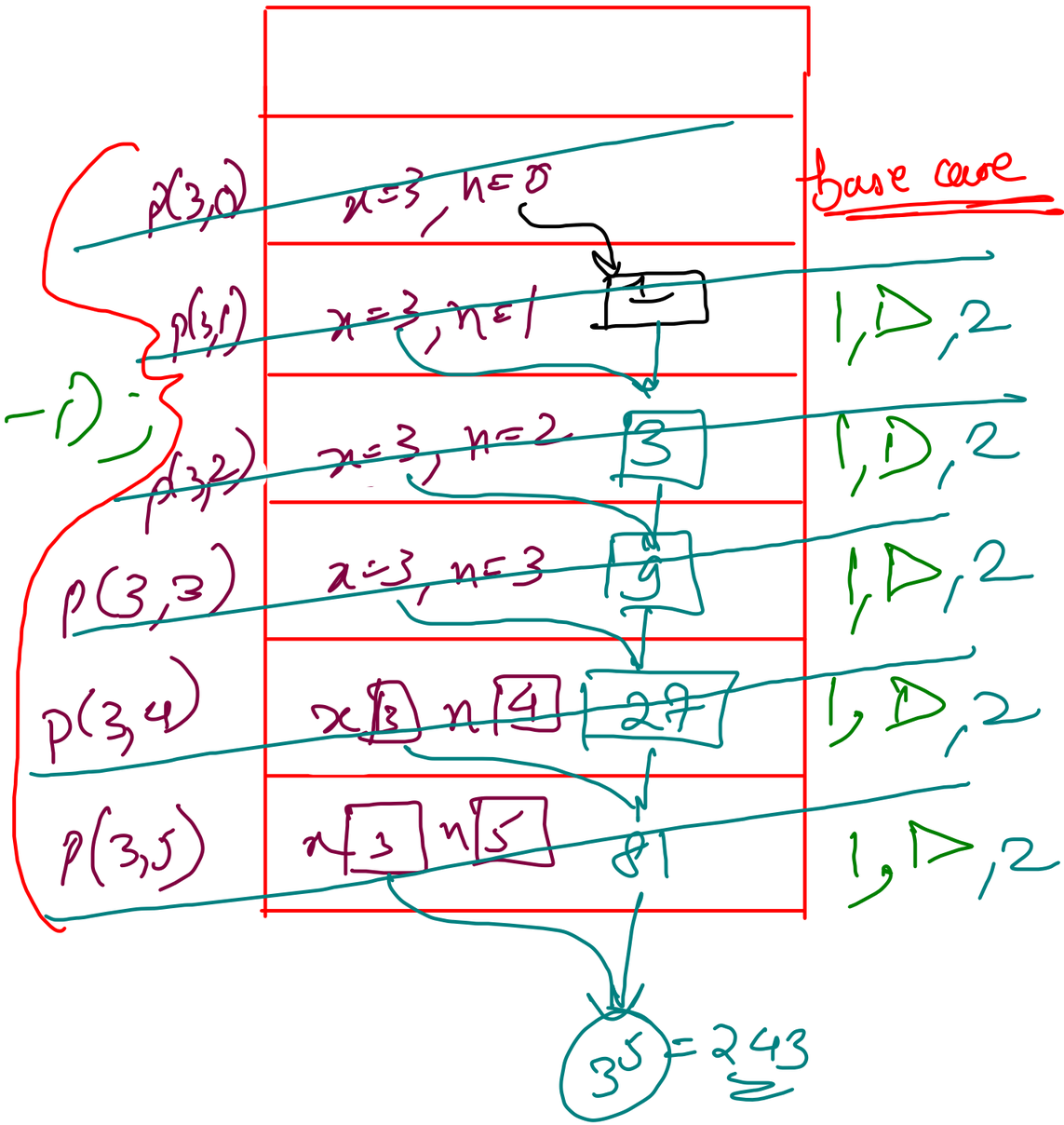
(3) Meeting Expectation :→

return pxnm1 * x;

$3^4 * 3 = 3^5$

How - Level Thinking

```
int power(int x, int n) {
    ① if(n == 0) return 1;
    ② int pxnm1 = power(x, n-1);
    ③ return pxnm1 * x;
}
```



```java
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

$a^b = a * a * a * \cdots$ b times

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

$x^n = x^{n/2} * x^{n/2}$ (n = even)

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

$x^n = x^{n/2} * x^{n/2} * x$ (n = odd)

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

① Expectation

power(x, n) :→ { 3 * 3 * 3 * 3 * 3 }

x (constant), n (variable)

$x * x * x * \cdots$ n times

② Path

power(x, n/2) :→

pxnby2

$x * x * \cdots$ n/2 times

③ Meeting Expectation :→

return pxnby2 * pxnby2 ;

*x if(n%2==1)

Low-level Thinking

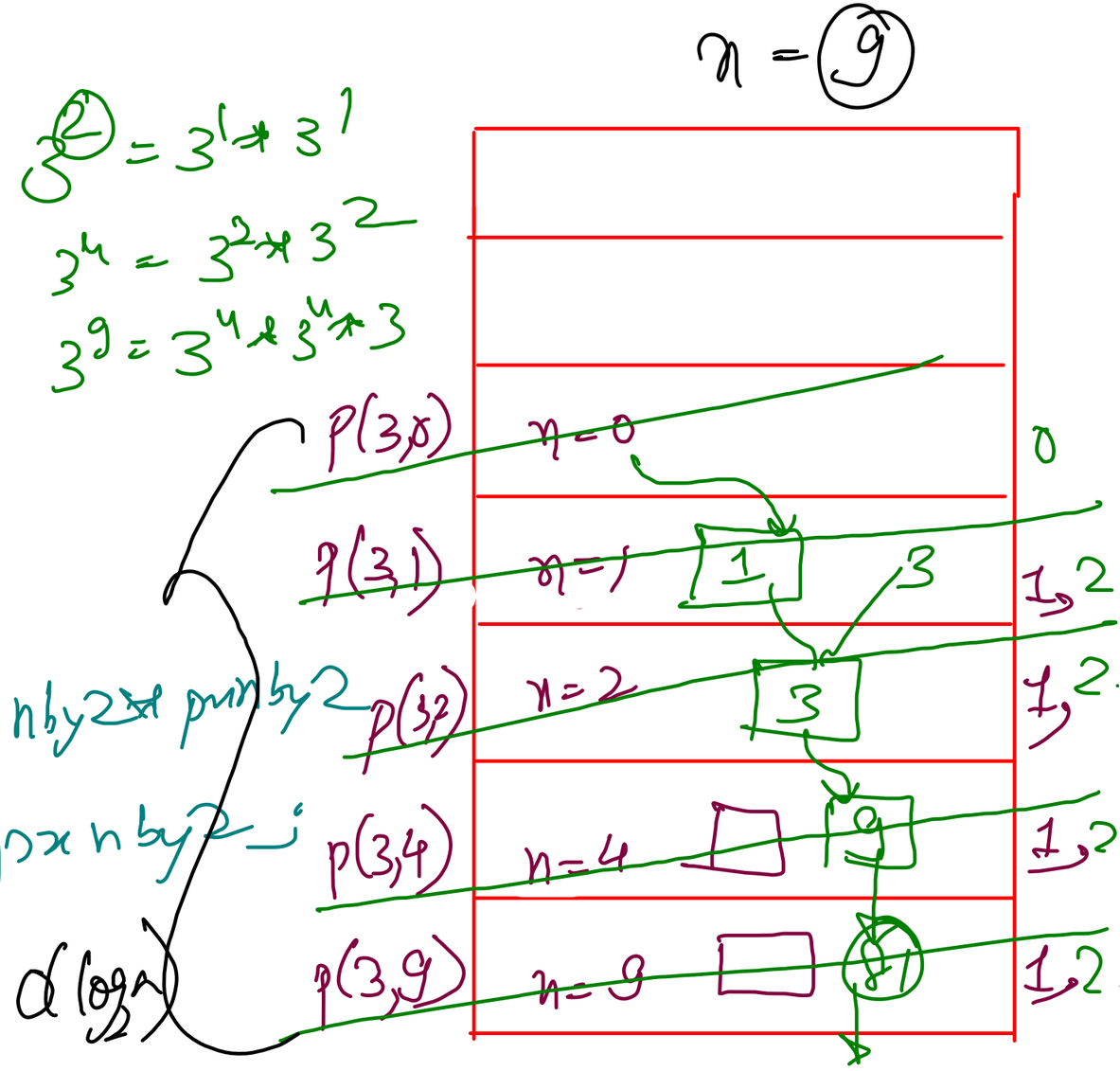
```
int power(int x, int n) {
    if(n==0) return 1;
    int pxnby2 = power(x, n/2);
    if(n%2==1) return x * pxnby2 * pxnby2;
    else return pxnby2 * pxnby2;
}
```

① ② ③ ④ ⑤ ⑥ ⑦ ⑧ ⑨ ⑩ ⑪ ⑫ ⑬ ⑭ ⑮ ⑯ ⑰ ⑱ ⑲ ⑳

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

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

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



```java
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;
}
```

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

$\log_2 n = \log_2(2^x)$

$\log_2 n = x * \log_2 2$

$\log_2 n = x \Rightarrow$ no of call $\Rightarrow O(\log_2 n)$

n

2^{20}

$O(N) \Rightarrow 1048576$

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

$= O(20)$

constant

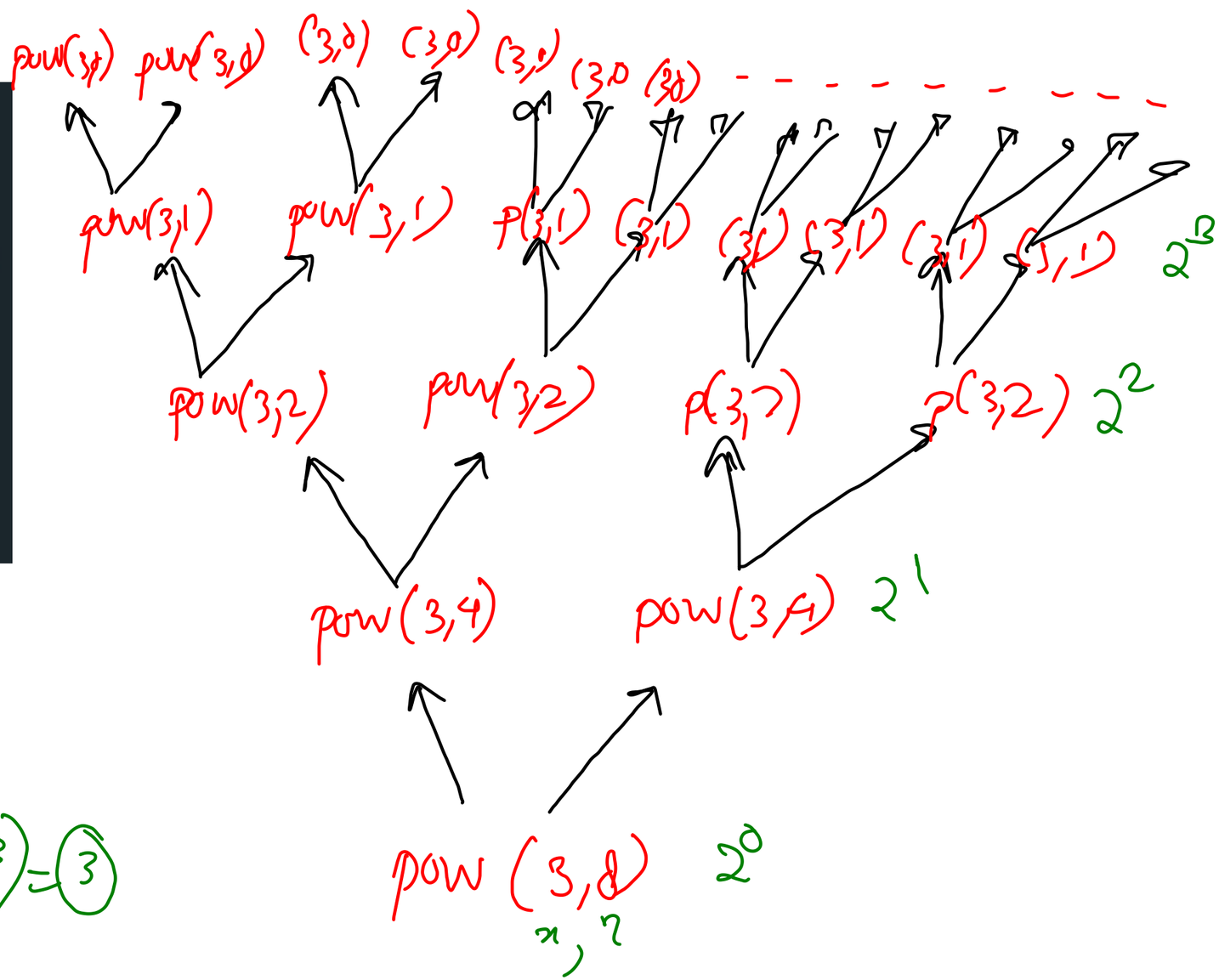
Today's Questions

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

Power → III rd method

```java
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;
}
```

L R



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

$\approx 2^4$

$\approx 2^{3+1}$

$\approx 2^{\log_2 n+1}$

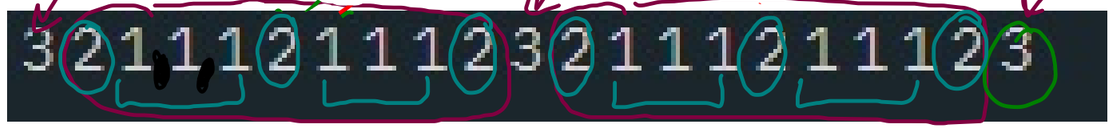
$\approx n$

$n = 8$

$O(\log_2 n) = O(\log_2 2^3) = 3$

(9) Print Zigzag

(1) Expectation :- printZigzag(int n)



3

2

(2) Faith :- printZigzag(n-1)

2 1 1 1 2 1 1 1 2

(3) Meet Expectation :-

```
Syso(n)
printZigzag(n-1)
Syso(n)
printZigzag(n-1)
Syso(n)
```

printZigzag(1)

1 1 1

printZigzag(0)

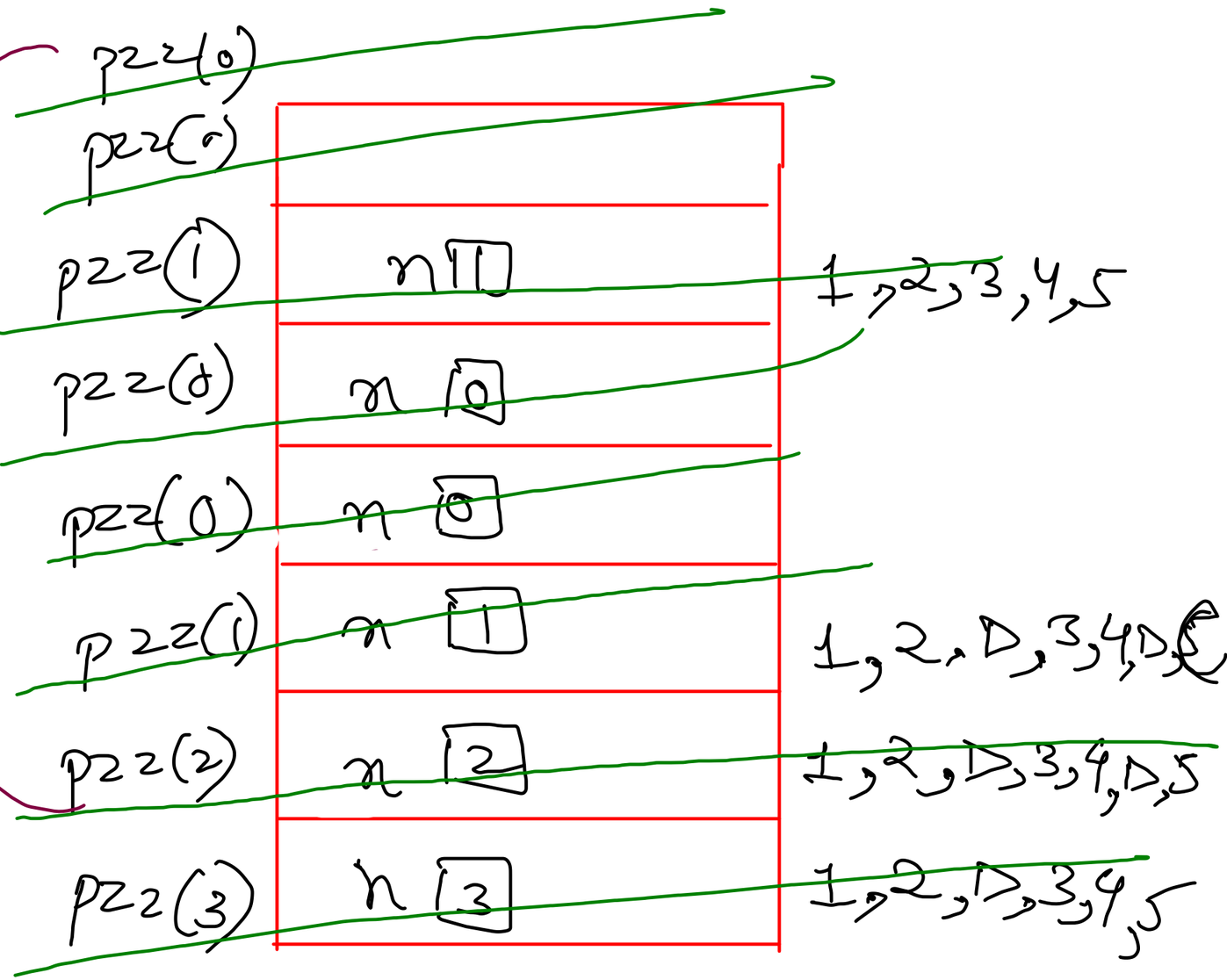
return;

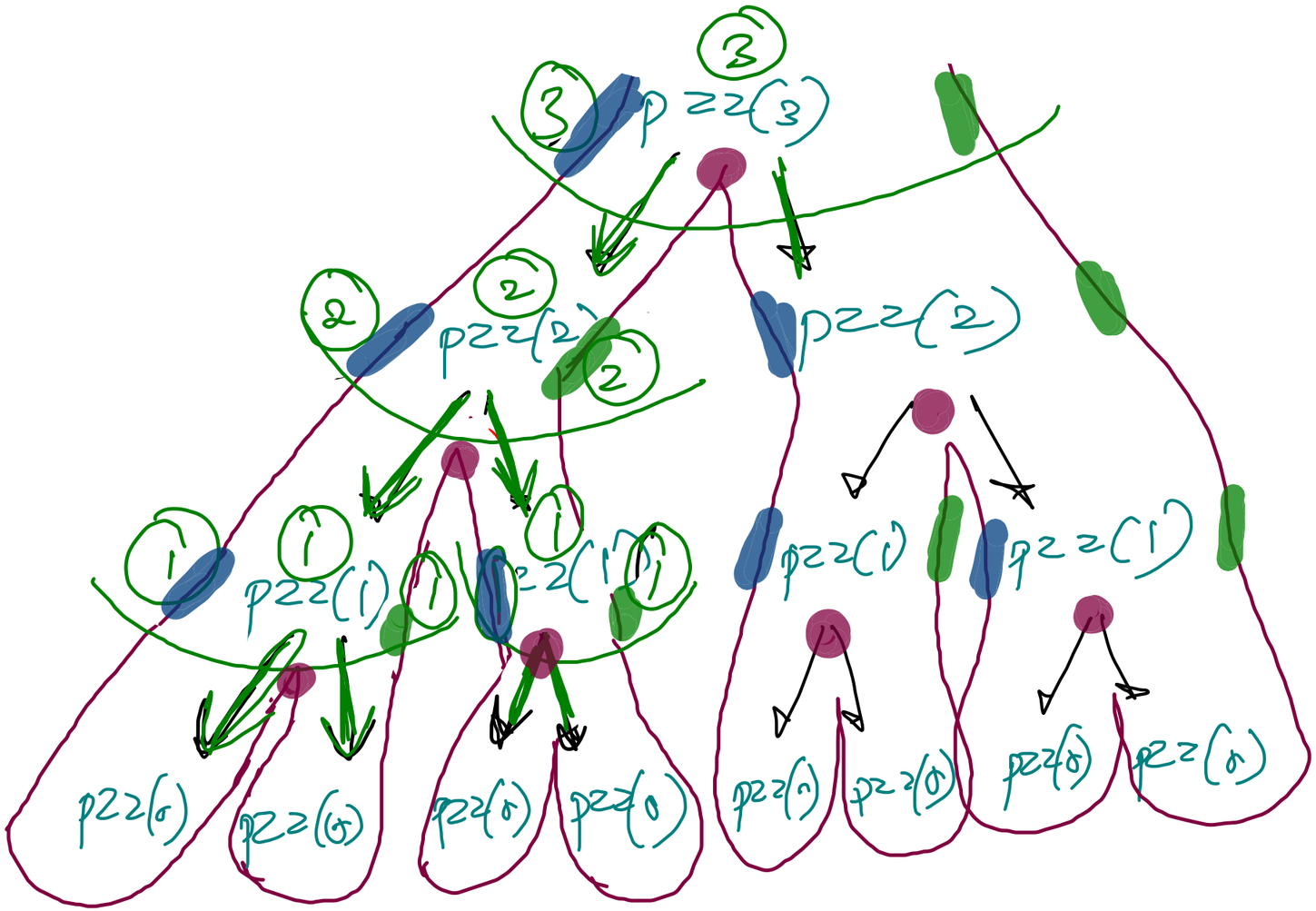
```java
public static void pzz(int n){
    if(n == 0) return;

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

x2

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

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






```java
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 + " ");
}
```

Depth first Search

⇓

Recursive tree/
callertree

Recursion

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

eg Print zigzag

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

$$O(2^n)$$

Recursion & Arrays

(Q) Display Array

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

}
```

① Expectation

{ 10, 20, 30, 40, 50 }

displayArr(arr, 0)

② Faith

display (arr, 1)

{ 20, 30, 40, 50 }

③ Meeting Expectation

↳ Preorder

syso(arr[idx])

display (arr, idx+1)

④ Base case

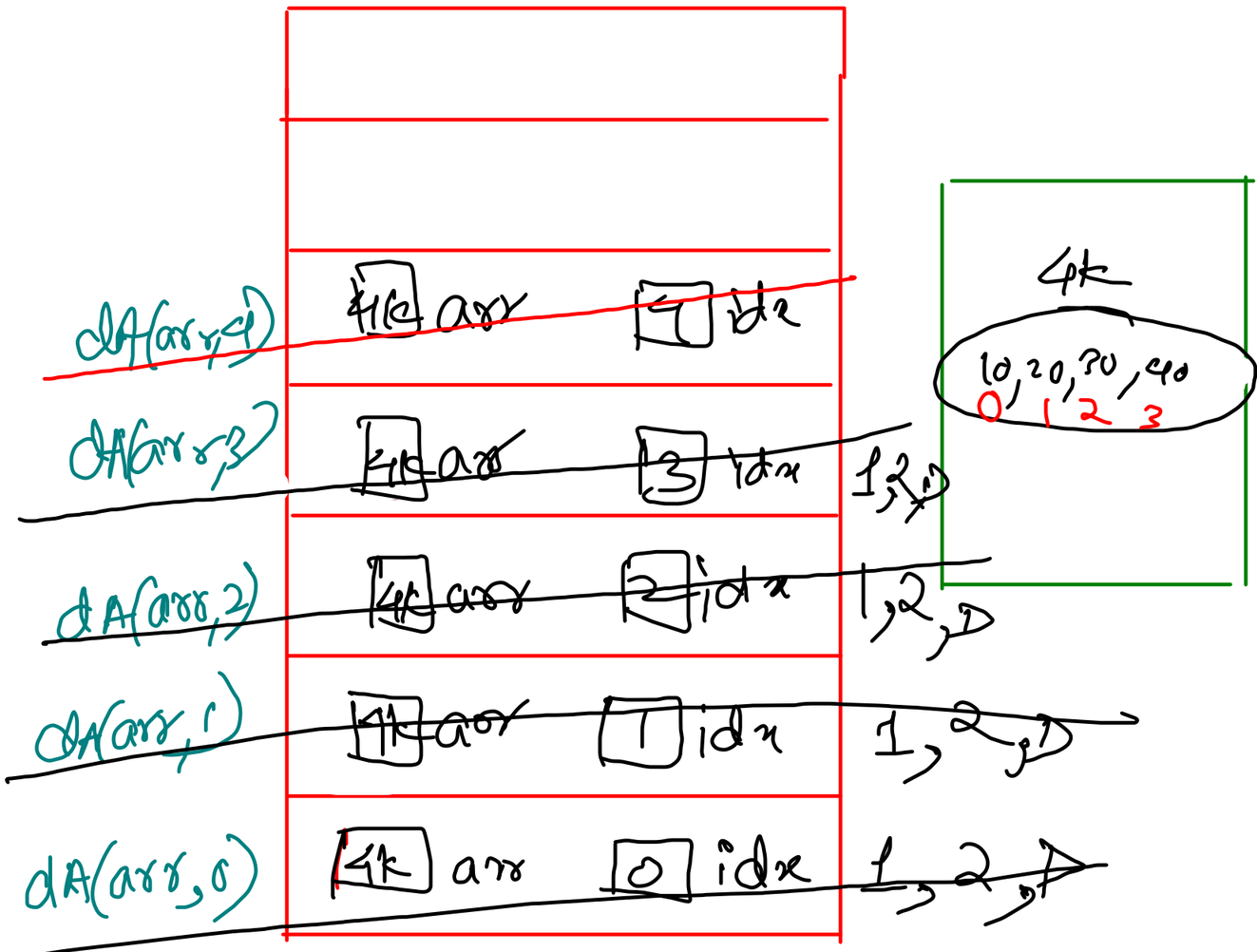
if (idx == arr.length)

return;

```java
public static void displayArr(int[] arr, int idx){
    if(idx == arr.length) return;

    // preorder -> Meeting Expectation
    System.out.println(arr[idx]);

    // faith
    displayArr(arr, idx + 1);
}
```

{preorder}

10, 20, 30, 40



⑥ Maximum In an Array

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

① Expectation

int maxOfArray(arr, 0) :→ max arr[0 : l-1]
⇒ 60

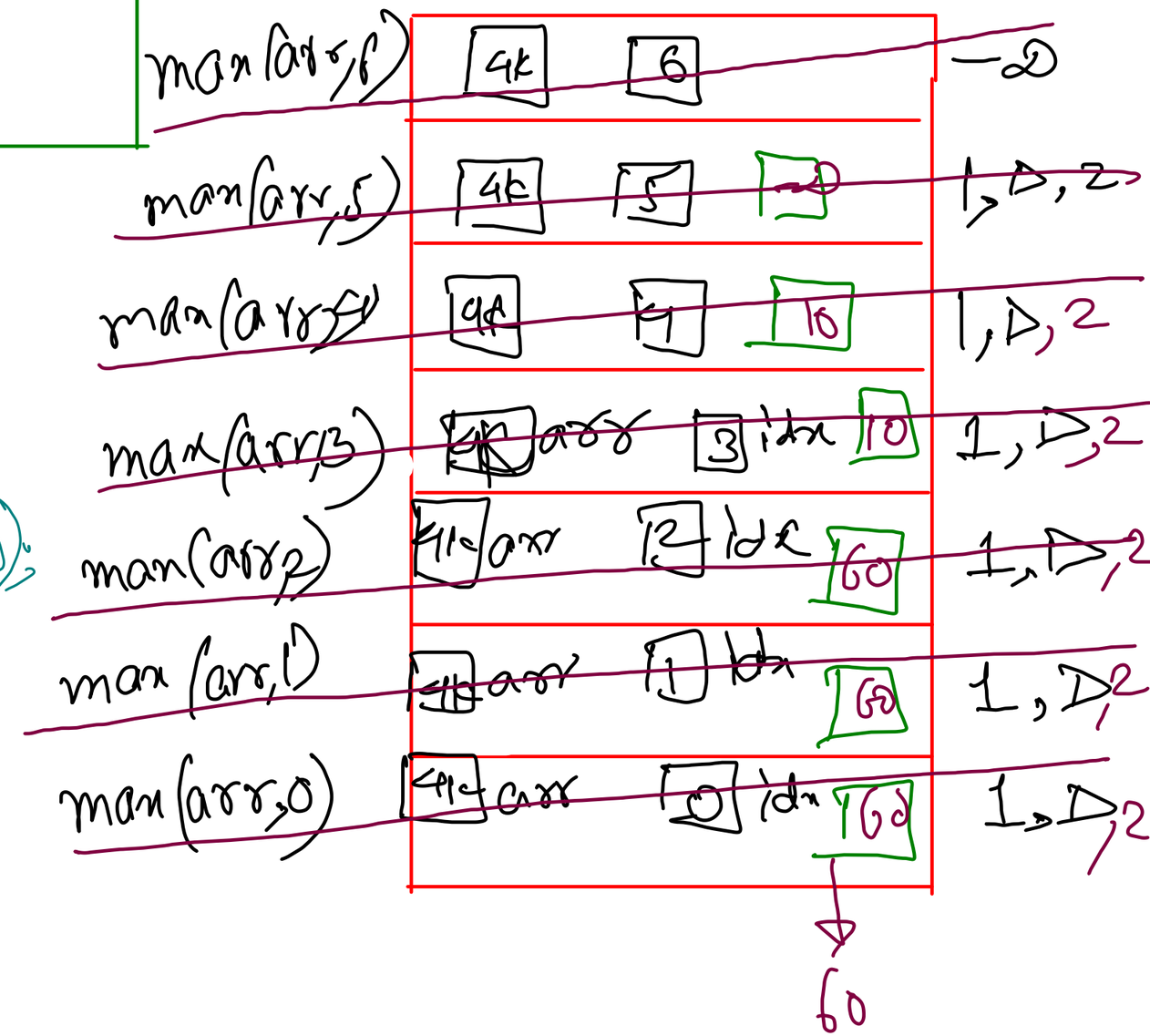
② Faith int maxOfArr(arr, 1) :→ max arr[1 : l-1]

③ Meeting Expectation :→ return max(arr[idx], maxCall)

heap

4k { 20, -30, 50, 60, 0, 10 }
0 1 2 3 4 5

```
int maxOfArr(int[] arr, int idx){
    ① if(idx == arr.length) return -∞;
    ② int maxTemp = max(arr, idx+1);
    ③ return Math.max(maxTemp, arr[idx]);
}
```



Addition

$x + ? = x$ (? = 0)

Subtraction

$x - ? = x$ (? = 0)

Multiplication

$x * ? = x$ (? = 1)

Minimum

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

$+\infty$ → Integer. MAX_VALUE

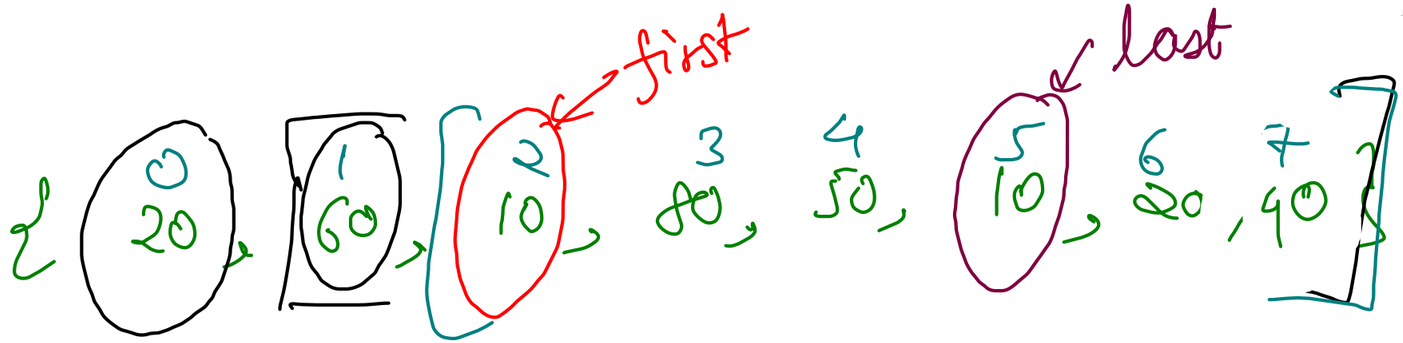
$+2^{31} - 1$

Maximum

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

$-\infty$ → Integer. MIN_VALUE

-2^{31}



First Index

fI (arr, 0) :→ 2

faith

temp = fI(arr, idx+1)

Meeting Exp

if(arr[idx] == target) T → return idx;
F → return temp;

L→R

lastIndex

lI (arr, a.l-1)

faith

temp = lI(arr, idx-1)

R→L

Meeting Exp

if(arr[idx] == target)
T → return idx;
return temp;

```java
public static int firstIndex(int[] arr, int idx, int x){
    if(idx == arr.length) return -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 }

```java
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);
    }
}
```