

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

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

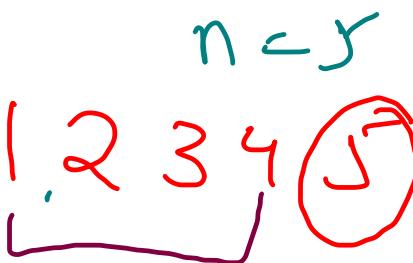
if $n=k+1$ LHS = RHS

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

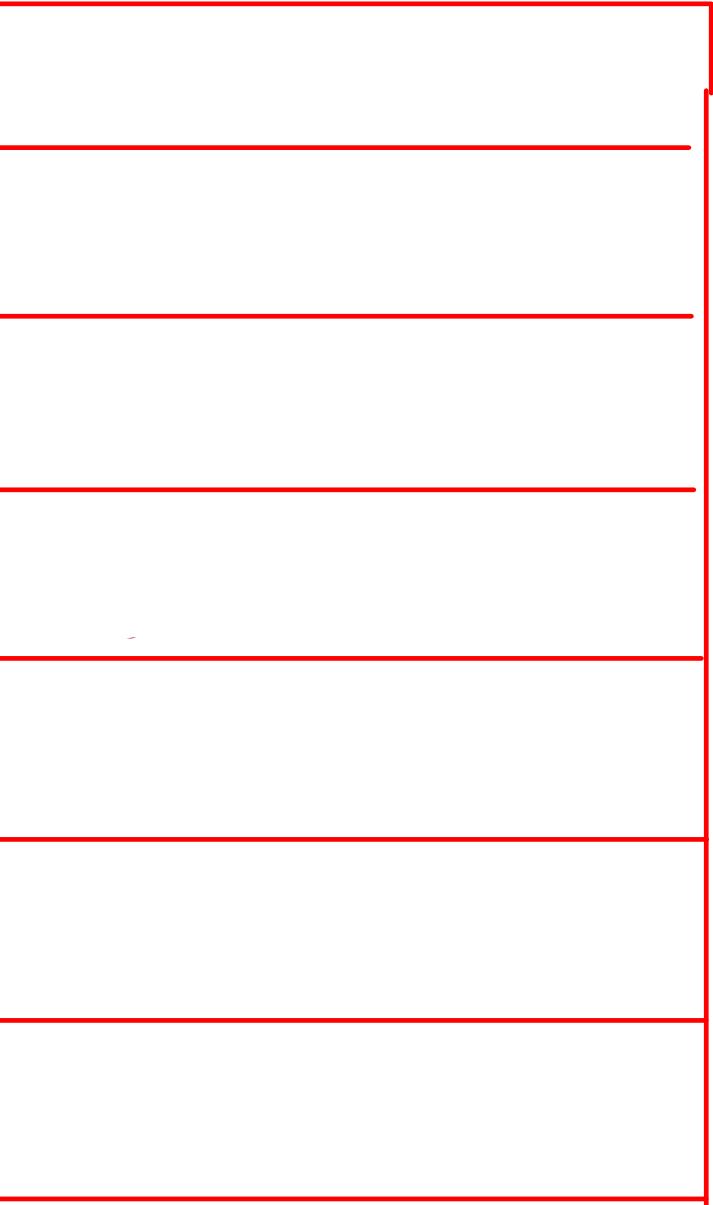
System.out.println(n);

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

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);  
    }  
}
```



Tail Recursion

↓

Work is being done while returning

$$(1)^n + (0+k)*n \\ = 1 + kn \Rightarrow$$

Q2) Point Decreasing

① pointDecreasing(int n) :->

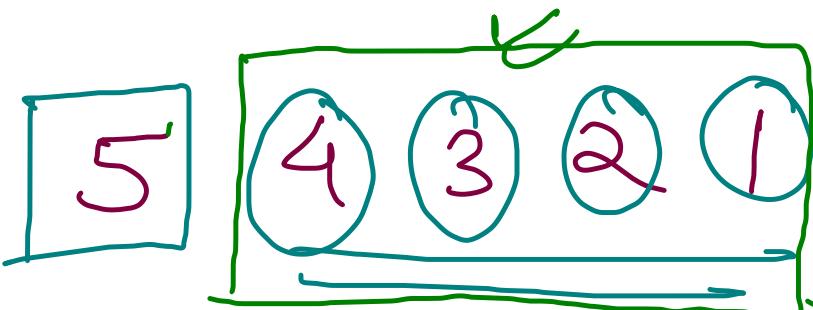
Expectation

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

② pointDecreasing(n-1) :->

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

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



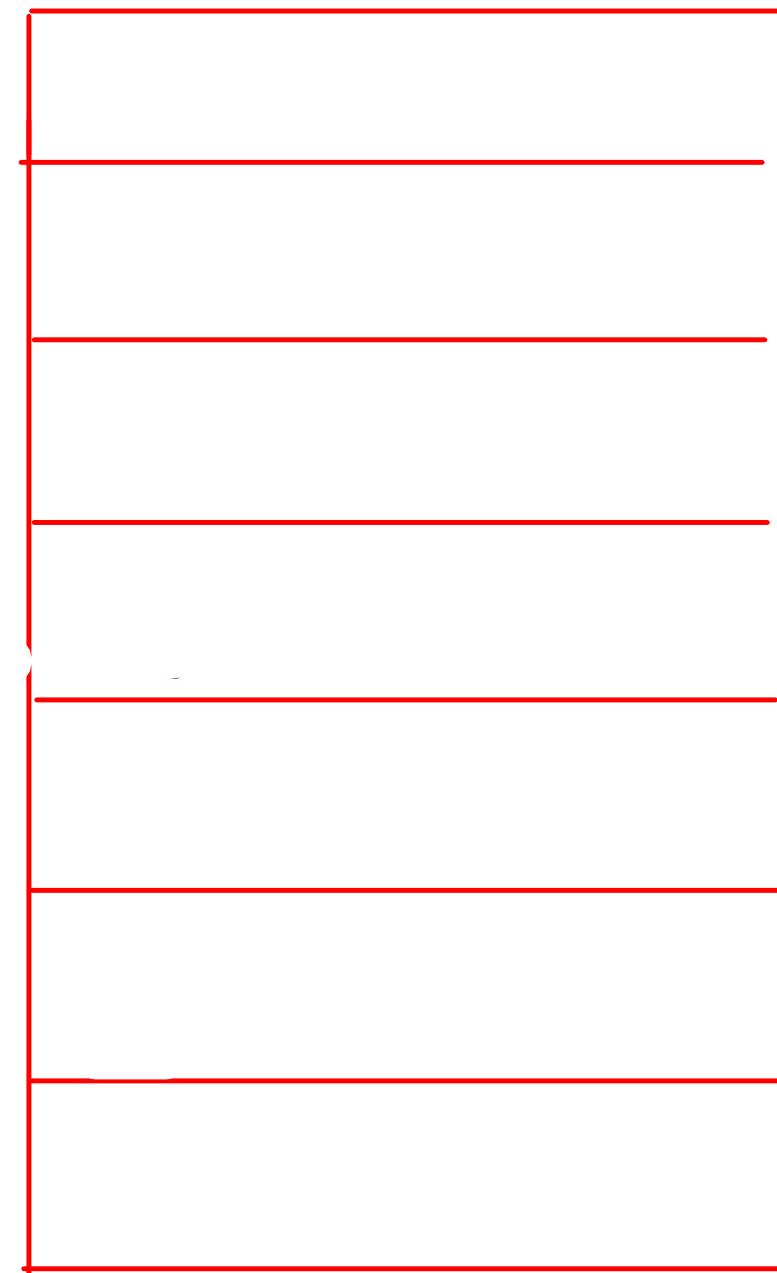
$n=5$ 5 4 3 2 1

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



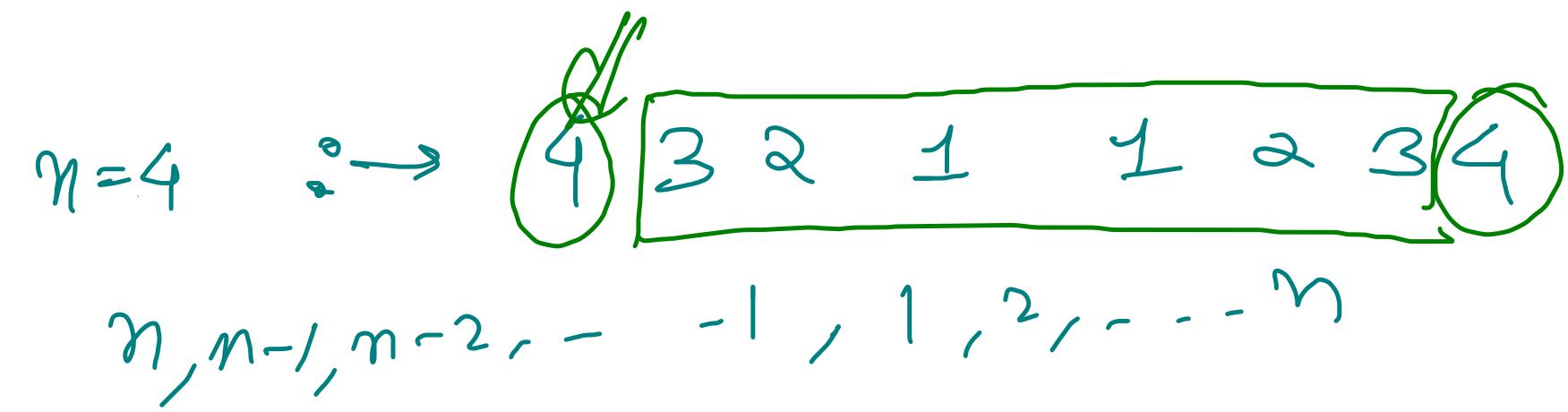
5, 4, 3, 2, 1

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

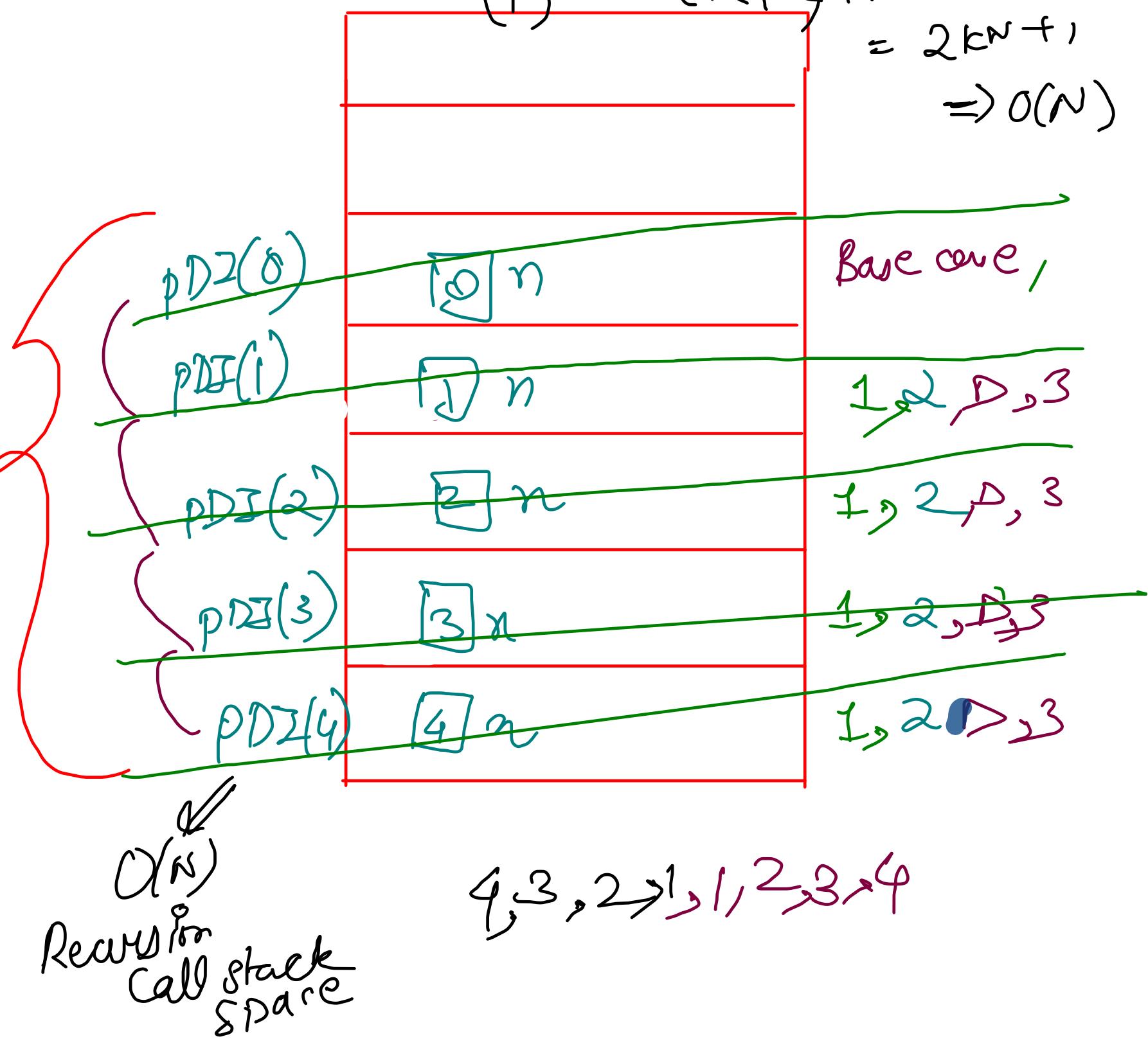
$$N = 4 \quad 4, 3, 2, 1, 1, 2, 3, 4$$

```

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

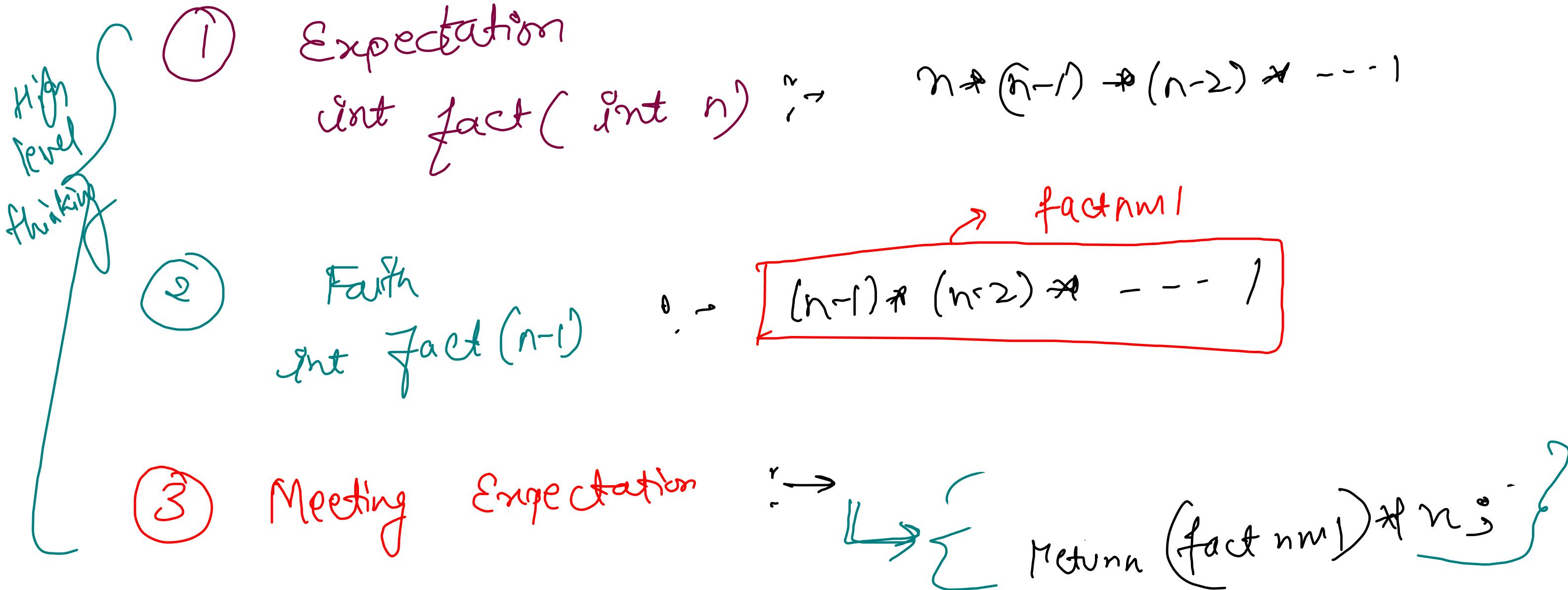
```

$$(1)^N + (k+k)*N = 2^{kn} + 1 \Rightarrow O(N)$$



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;

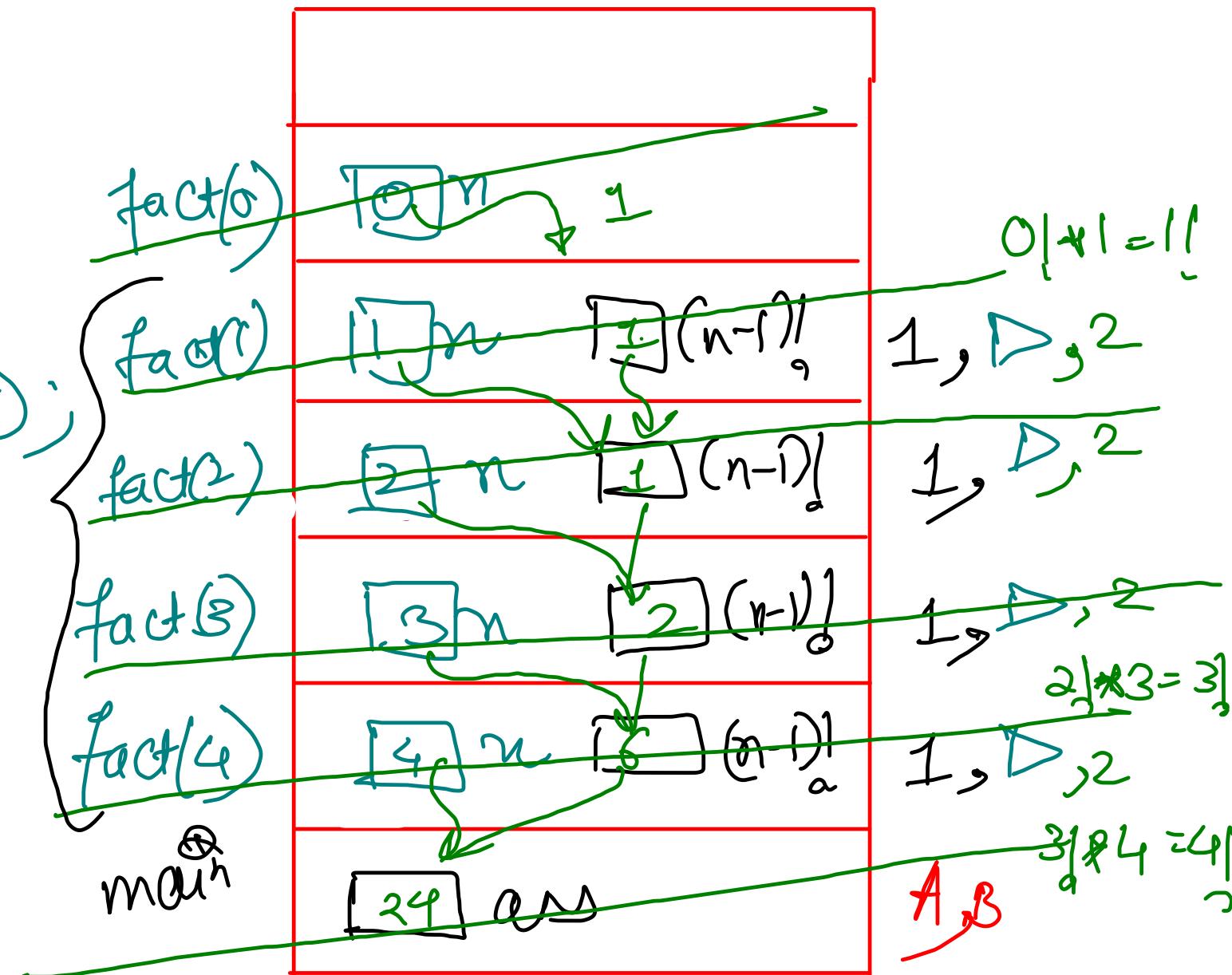
faith { ② int factm1 = fact(n-1);

Meeting
Expectation
3 }
return factm1 * n;

```
main() {
```

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

Call
 $O(N)$



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

$p_{XNML} * x^j$
 $3^4 * 3 = 3^5$

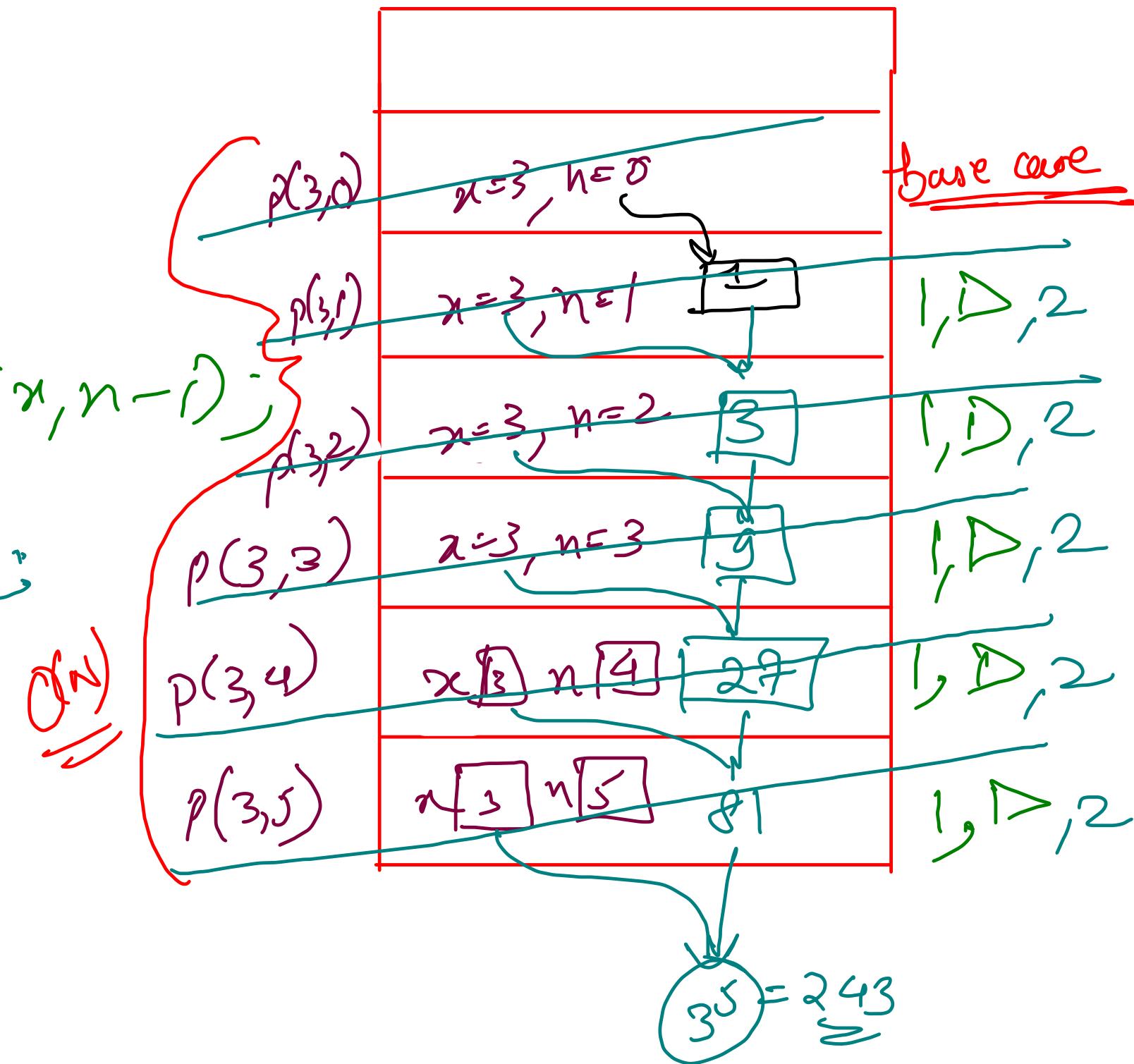
Low-level Thinking

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

① if ($n == 0$) return 1;

② int pxnml = power(x, n-1);
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
}

```

(Calls) ^{height} + (pre + post) ^{height}

$$(1)^N + (k+k)*N$$

$$1 + 2 \cdot N$$

$O(N)$

$$\begin{aligned}
x^N &\sim 2^6 = 2 * 2^5 = 2 * 2 * 2^4 \\
&= 2 * 2 * 2 * 2^3 \\
&= 2 * 2 * 2 * 2 * 2 * 2
\end{aligned}$$

Power - logarithmic

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

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

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

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

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

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

~~if odd~~

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

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

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

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

$$\overbrace{x * x * \dots}^{\text{n/2 times}} \xrightarrow{\text{pxnby2}}$$

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

return $\boxed{\begin{array}{l} \text{pxnby2} \Rightarrow \text{pxnby2} \\ 0 * x \text{ if } (n/2 \text{ is } 1) \end{array}}$

Low-level Thinking

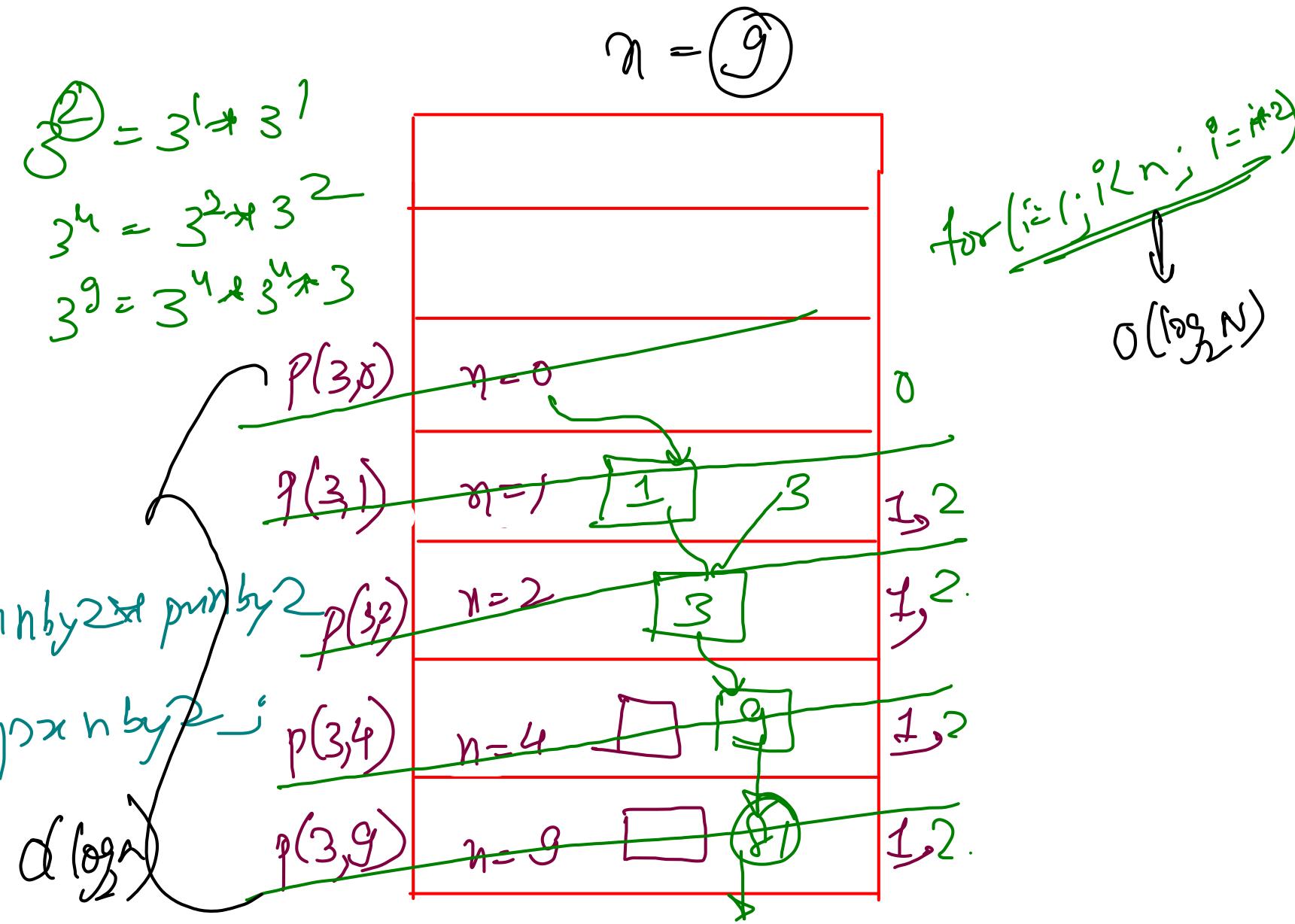
```
int power(int x, int n) {
```

① if ($n == 0$) return 1;

② int p_{xnby2} = power(x, n/2);

③ if ($n \% 2 == 1$) return $x * p_{xnby2} * p_{xnby2}$;
 else return $p_{xnby2} * p_{xnby2}$;

}



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

$$16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1 \rightarrow 0$$

$$2^n = N \Rightarrow n = \log_2 N$$

```

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); variable
    if(n % 2 == 1) return pxnby2 * pxnby2 * x;
    else return pxnby2 * pxnby2;
}

```

$$(\text{Calls})^{\text{height}} + (\text{pre+post}) * \text{height}$$

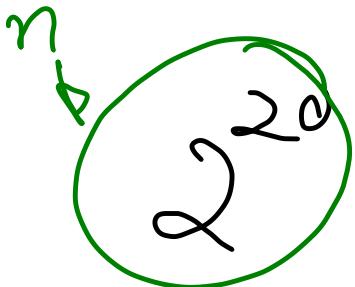
$$(1)^h + (k+k)*h$$

$$(1)^{\log_2 N} + k * \log_2 N$$

$$1 + \log_2 N$$

$$\boxed{O(\log_2 N)}$$

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



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

$$O(n) \Rightarrow 1048576$$

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

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

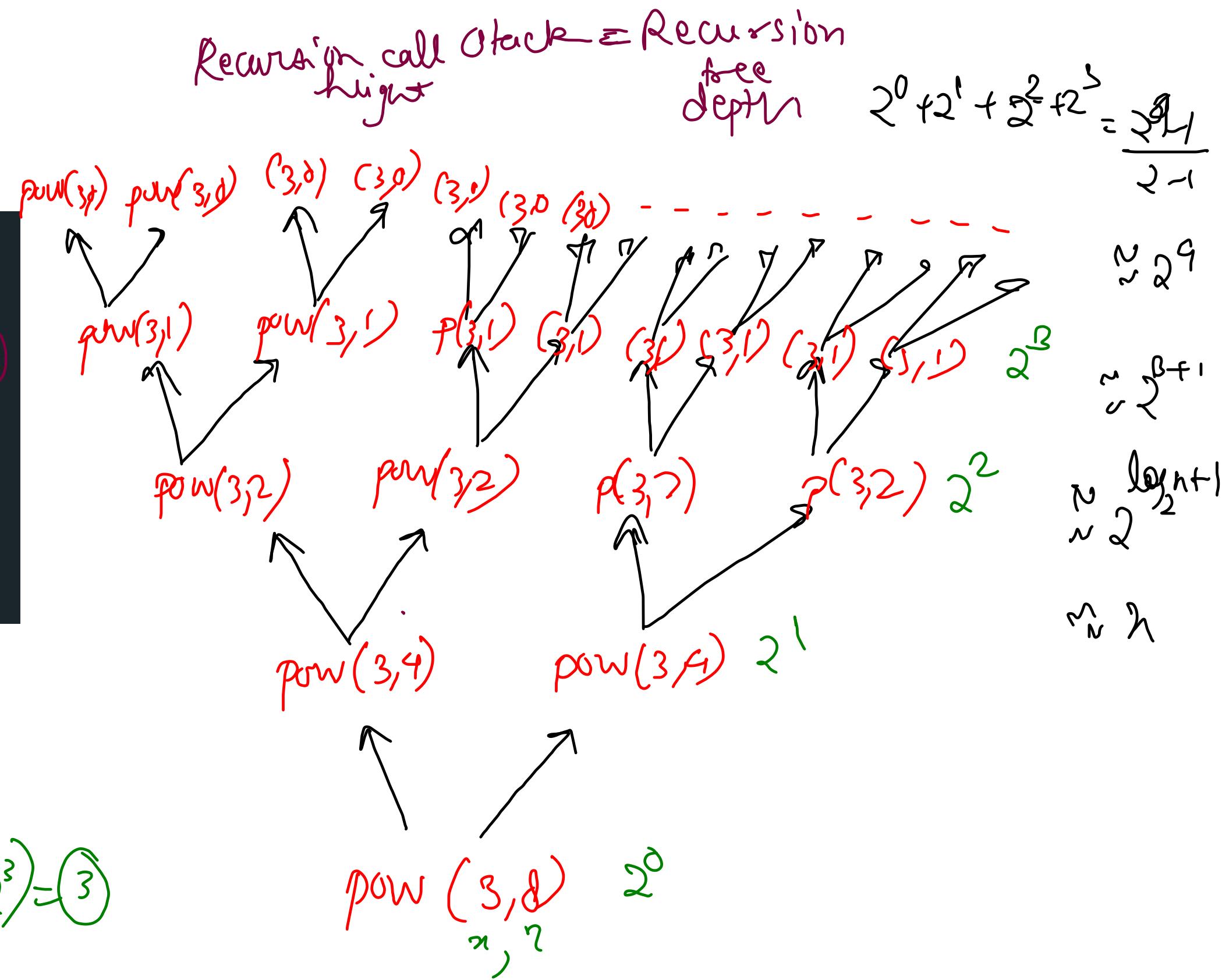
$$\log n = x \Rightarrow \text{no of call} \Rightarrow O(\log_2 n) \text{ constant}$$

Today's Questions

- ① Power - 3rd method
- ② Print zigzag
- ③ Display Array
- ④ Display Array - Reverse of Home work }
- ⑤ Max of Array
- ⑥ First Index of Array
- ⑦ Last Index 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;  
}
```



Recursion \Rightarrow (Calls)^{height} + {pre + post}* height

$$(2)^h \quad \{k_1 + k_2\} * h$$

$$(2)^{\log_2 N} + k * \log_2 N$$

$$N + \cancel{k \log N}$$

$O(N)$

(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 ;

```

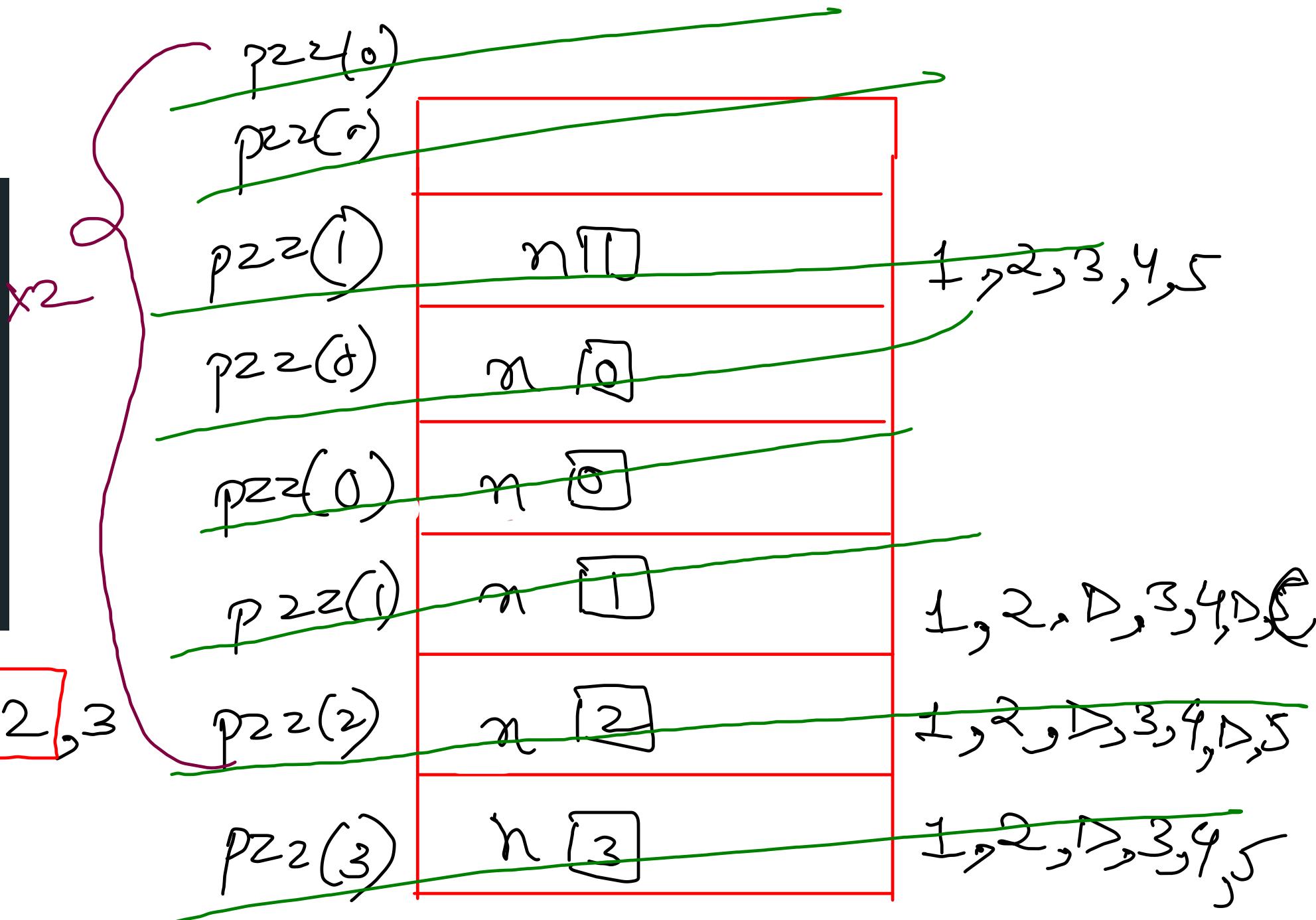
public static void pzz(int n){
    if(n == 0) return;

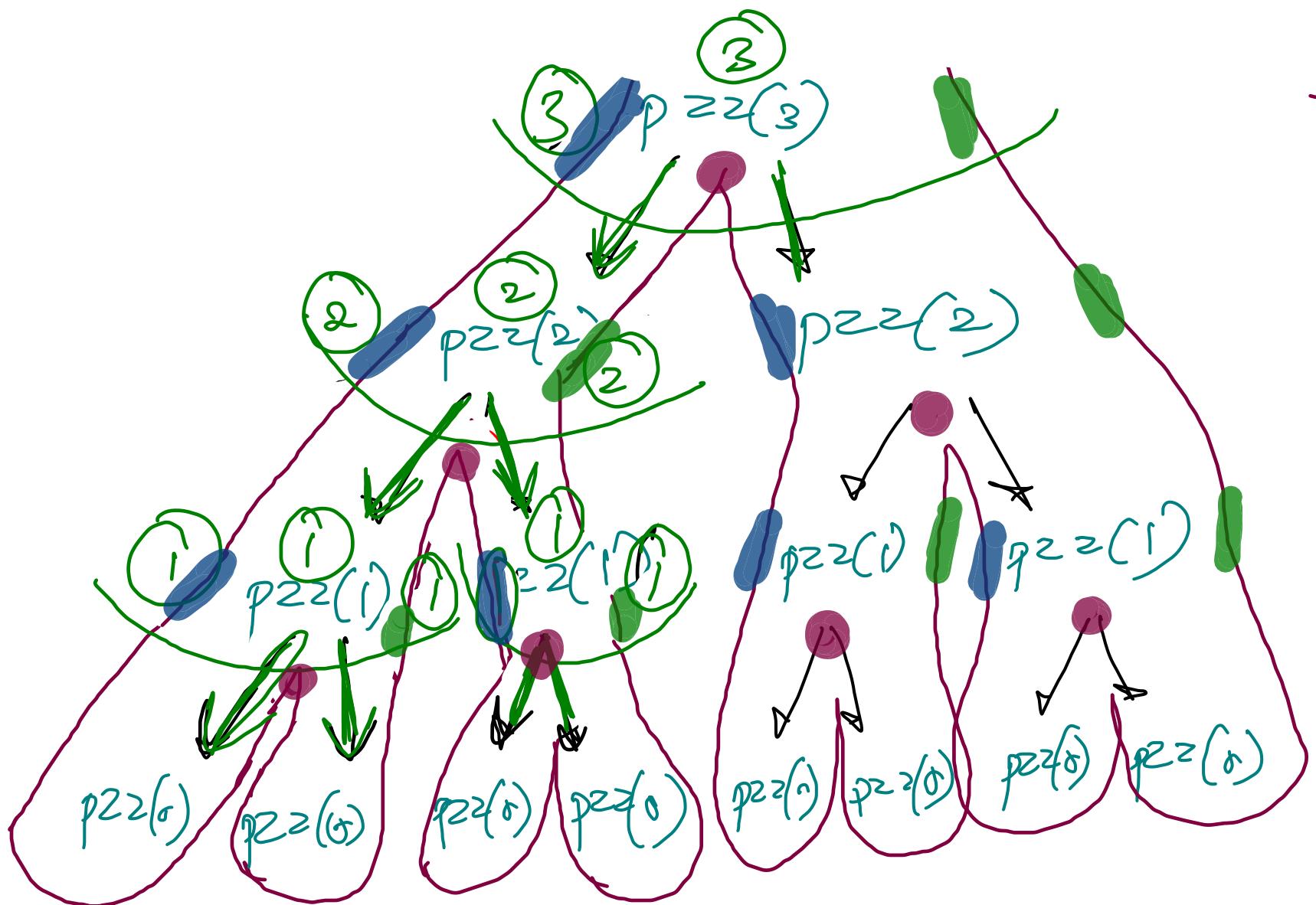
    System.out.print(n + " ");
    pzz(n - 1);
    System.out.print(n + " ");
    pzz(n - 1);
    System.out.print(n + " ");
}

```

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

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





$O(n)$ height Decreasing ($n \rightarrow n-1$), Dividing ($n \rightarrow n/2$) $O(\log_2 n)$

```
public static void pzz(int n){
    if(n == 0) return;
    Preorder System.out.print(n + " ");
    Left pzz(n - 1);
    Middle System.out.print(n + " ");
    Right pzz(n - 1);
    Postorder System.out.print(n + " ");
}
```

Depth first search

↓
Recursive tree/
euler tree

$$(2)^N + \left\{ k_1 + k_2 \right\} \propto N$$

$$= 2^N + kn \Rightarrow O(2^N)$$

Recursion

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

\Rightarrow Point zigzag

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

$\boxed{O(2^n)}$

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 }

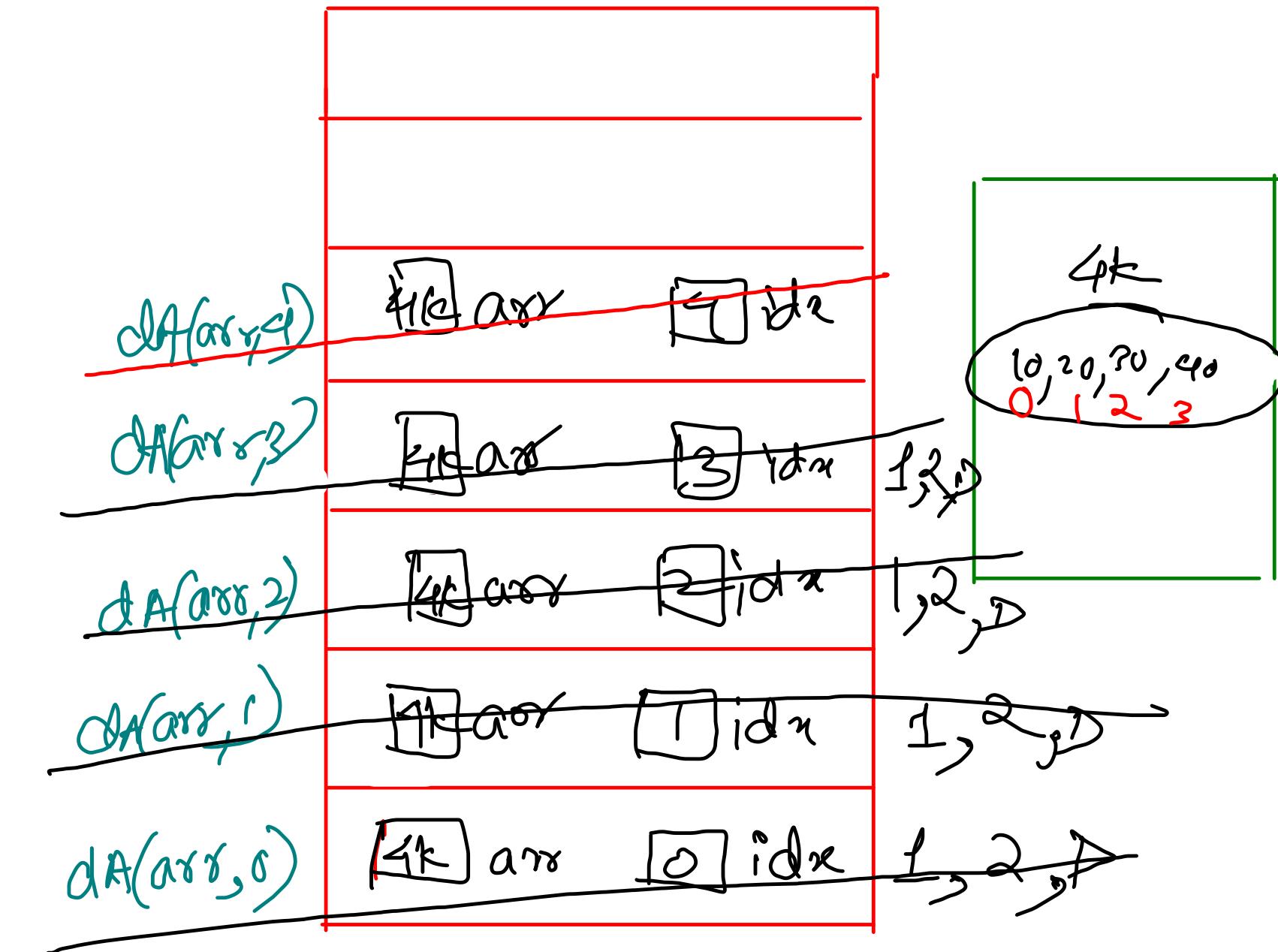
④ Base case
if(idx == arr.length)
 return

```

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

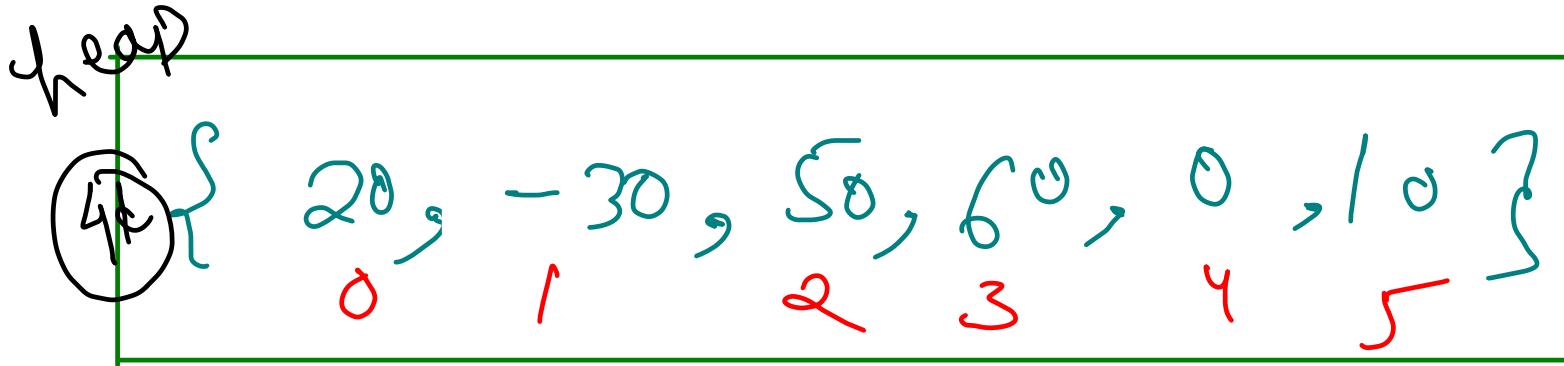
{ 20, -30, 50, 60, 0, 10, 40 }

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)



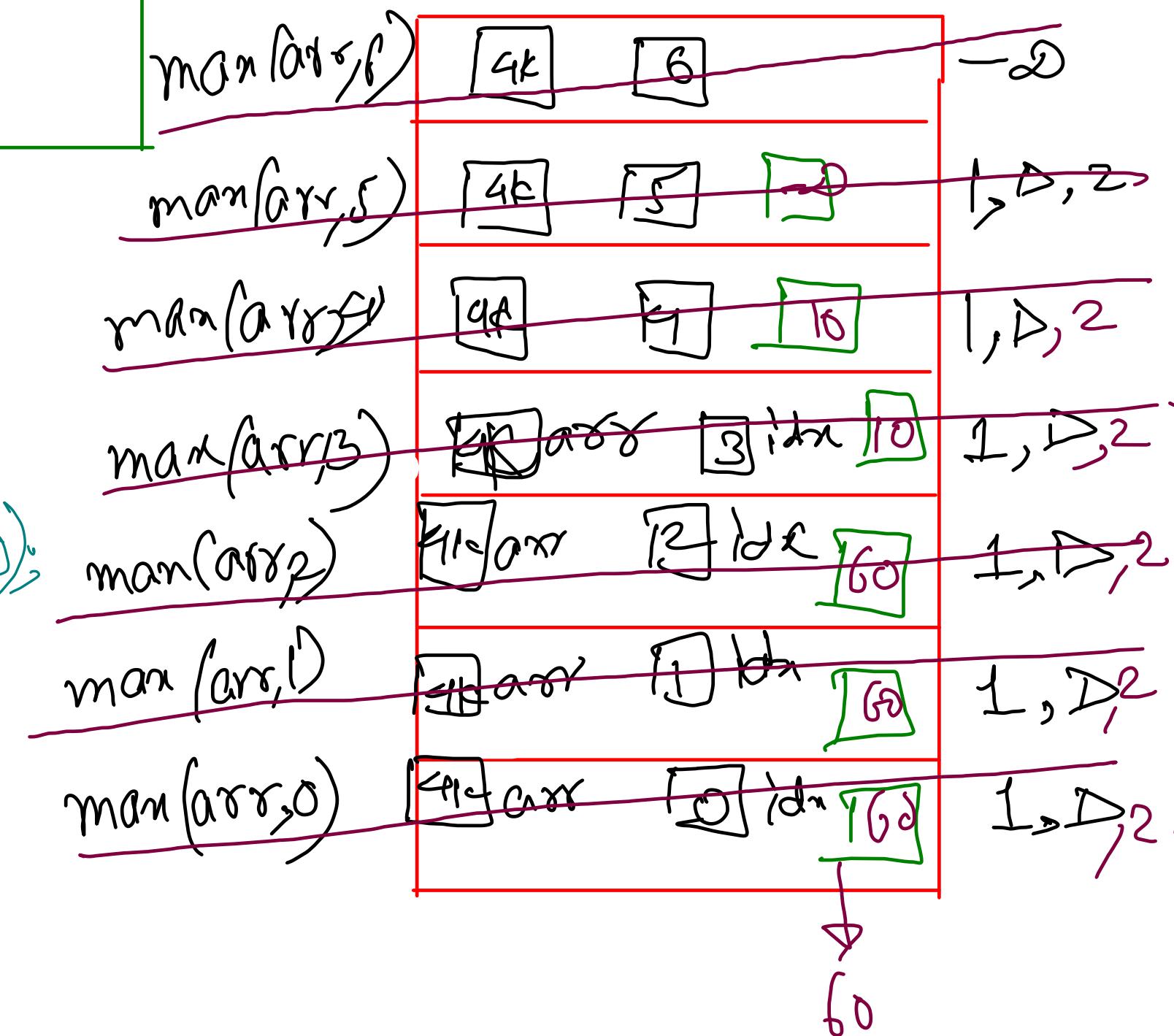
```

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]);
}

```

$$(1)^{K^1} + (K+k)^{\star N}$$

$O(N)$



Addition

$$x + ? = x$$

Subtraction

$$x - ? = x$$

Multiplication

$$x * ? = x$$

Minimum

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



Integer. MAX VALUE

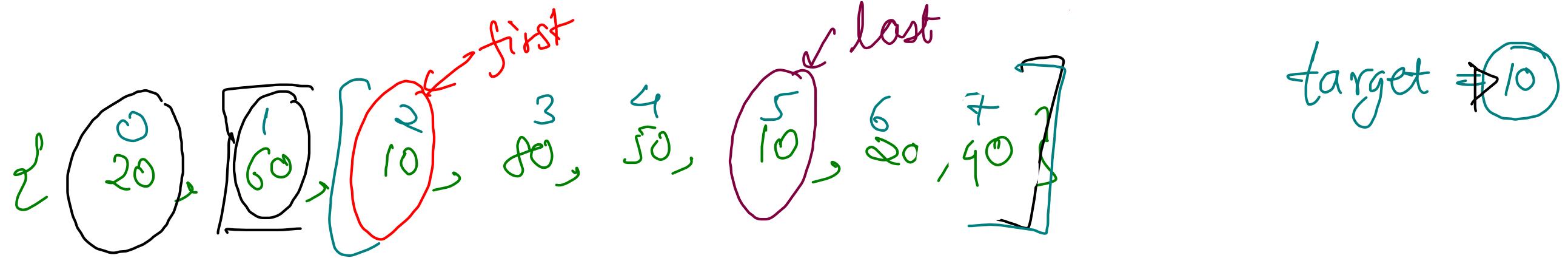
$$+2^{31} - 1$$

Maximum

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

Integer. MIN VALUE

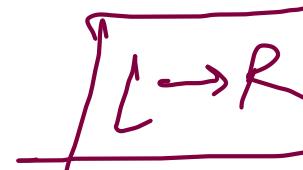
$$-2^{31}$$



First Index

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

faith



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

Meeting End

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

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

faith



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

Meeting End

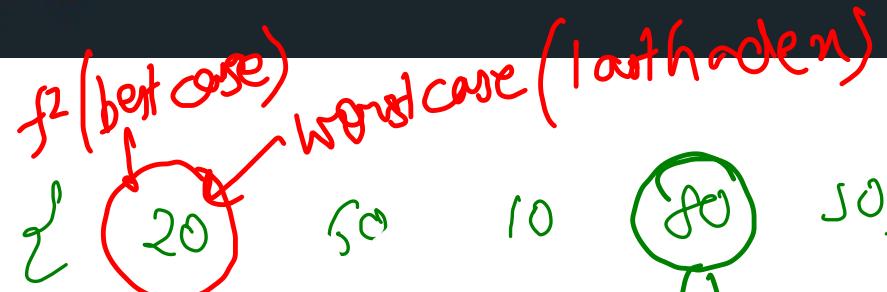
if($\text{arr}[\text{idn}] == \text{target}$)
 T → return $\text{idn};$
 F → return $\text{temp};$

```

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);
    }
}

```



```

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);
    }
}

```

Best case $\rightarrow \Theta(1)$ Worst case $\rightarrow O(n)$

$$\text{avg case} = \Theta(n/2) = \Theta(n)$$

firstIndex(worst case)
lastIndex(best case)

$f_2(\text{arr}, 2)$

$f_2(\text{arr}, 1)$

$f_2(\text{arr}, 0)$

$\text{id}_n = 2$

$\text{id}_n = 1$

$\text{id}_n = 0$

$\hookrightarrow 1$

$\hookrightarrow 2$

$\hookrightarrow 2$

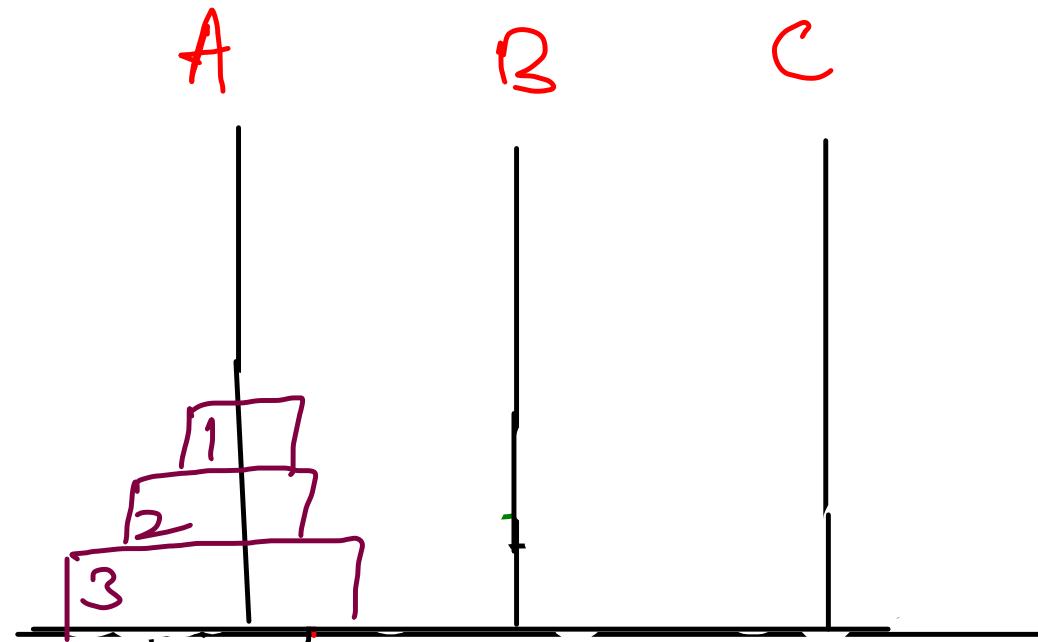
Today's Questions

- ① Tower of Hanoi^o
- ② All indices of Array

+ ~~Discussion~~ on DSA based projects

Tower of Hanoi

1. There are 3 towers. Tower 1 has n disks, where n is a positive number. Tower 2 and 3 are empty.
2. The disks are increasingly placed in terms of size such that the smallest disk is on top and largest disk is at bottom.
3. You 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 disk.
 - 3.3.3 you can only move a disk at the top.



Constraints:

- ① only one disk
can be moved
at a time
- ② Larger disk cannot be
placed on top of
smaller disk

① Expectation

tower of hanoi(n , A, B, C)

(source) form
to
using



Instructions
to move 3 disk
from A to B
using C

② Faith

(2.1) tower of hanoi ($n-1$, A, C, B) → Move $(n-1)$ disks
from A to C
using B

Meeting Expectation

(2.2) Move n^{th} disk from A to B

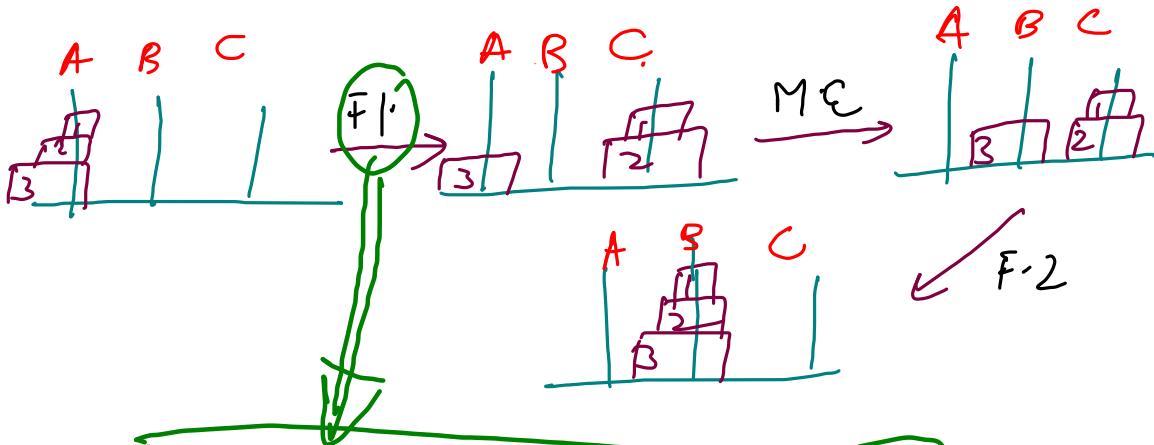
(2.3) tower of hanoi $n-1$, C, B, A) →

move $(n-1)$ disk
from C to B
using A.

```

public static void toh(int n, int srcTower, int destTower, int auxTower){
    if(n == 0) return;
    ① toh(n - 1, srcTower, auxTower, destTower);
    // Move N - 1 disks from source to auxiliary
    ② System.out.println(n + "[" + srcTower + "-">+ destTower + "]");
    // Move nth disk from source to destination
    ③ toh(n - 1, auxTower, destTower, srcTower);
    // Move N - 1 disks from auxiliary to destination
}

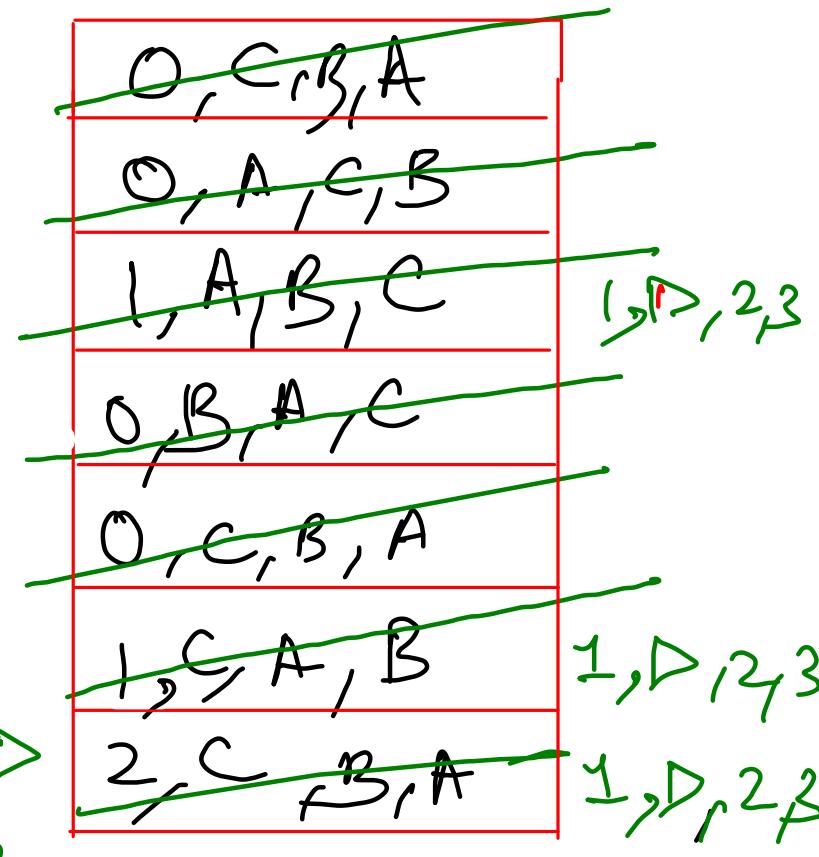
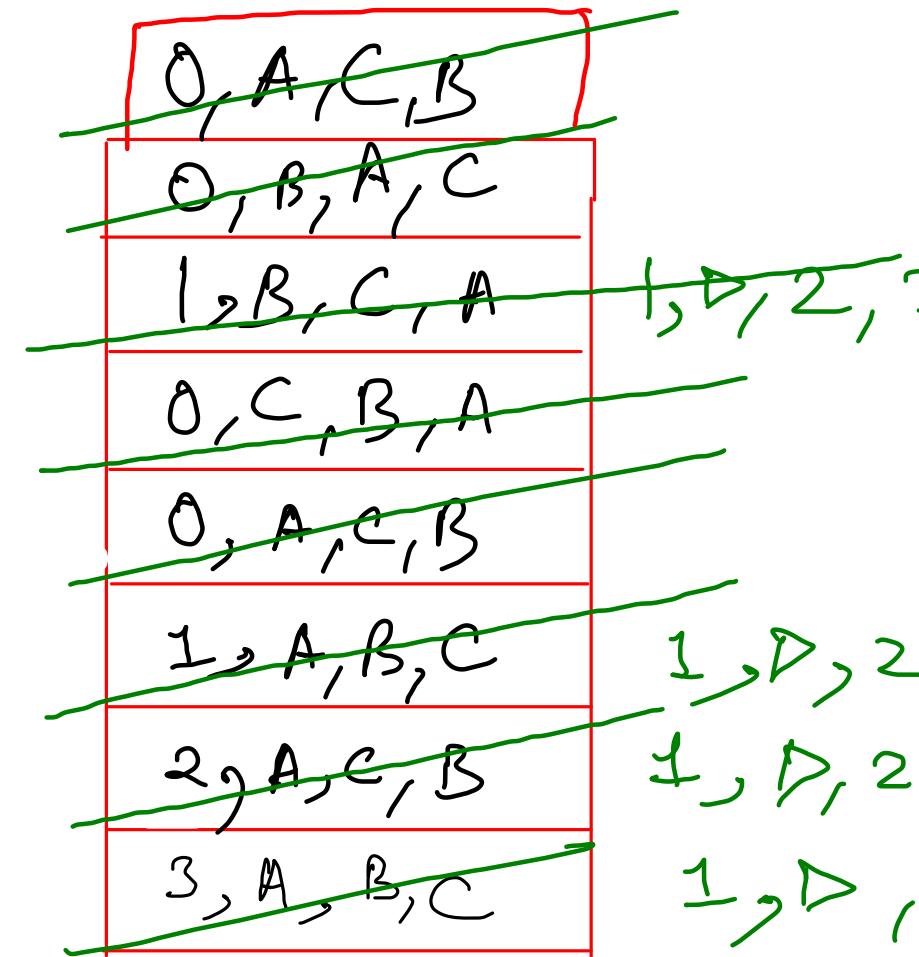
```



- ① 1st from A to B
- ② 2nd from A to C
- ③ 1st from B to C

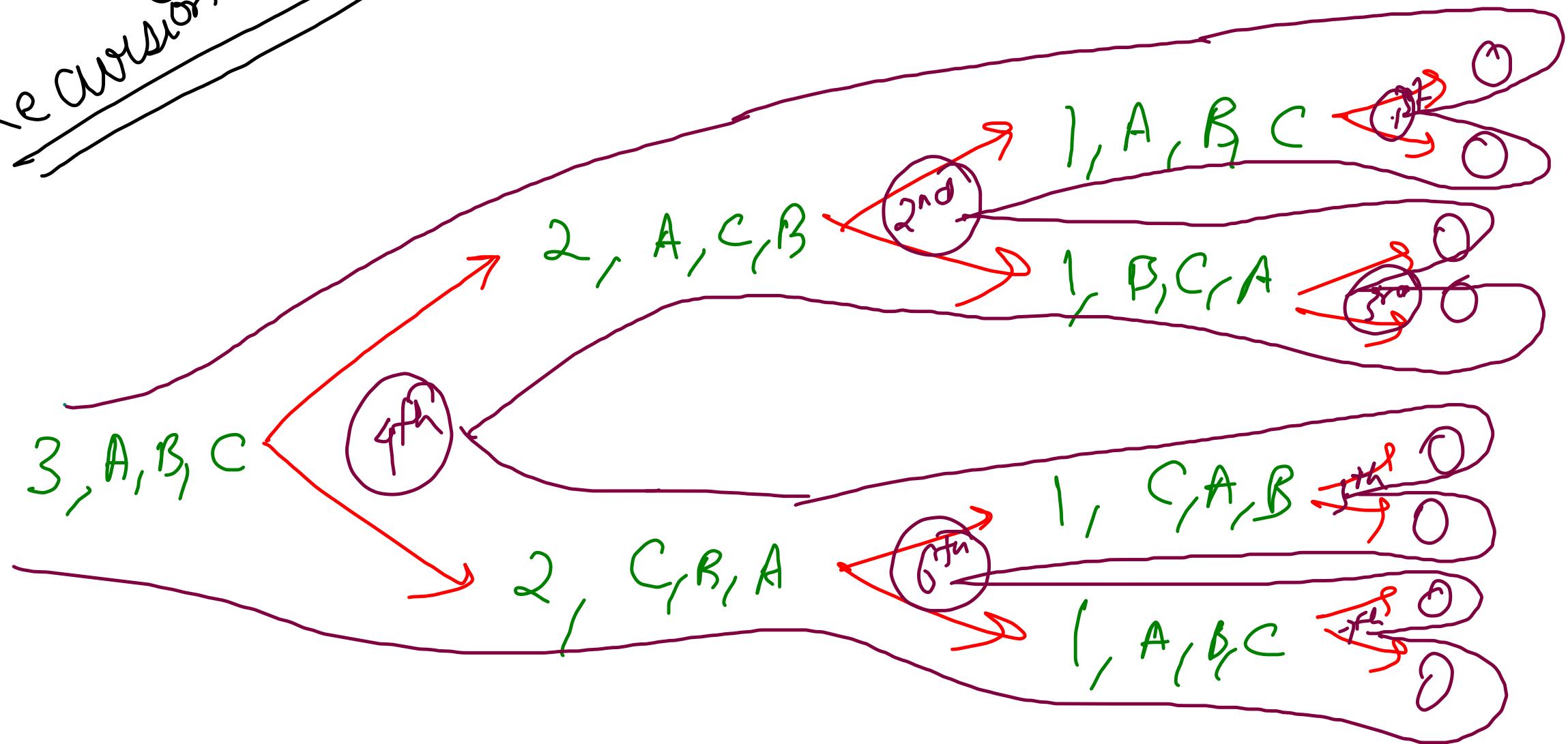
- ④ 3rd from A to B

- ⑤ 1st from C to A
- ⑥ 2nd from C to B
- ⑦ 1st from A to B



$$TC = Q^m + \{k\} + N = 2^n + k$$

Recursion Tree

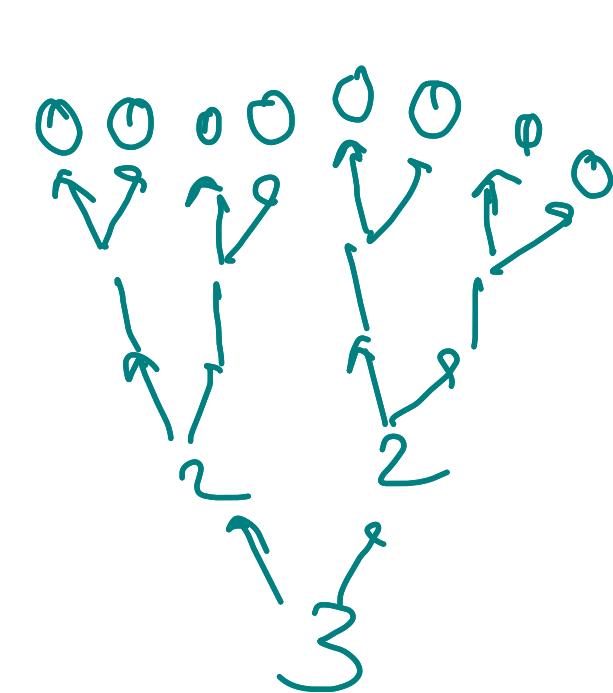
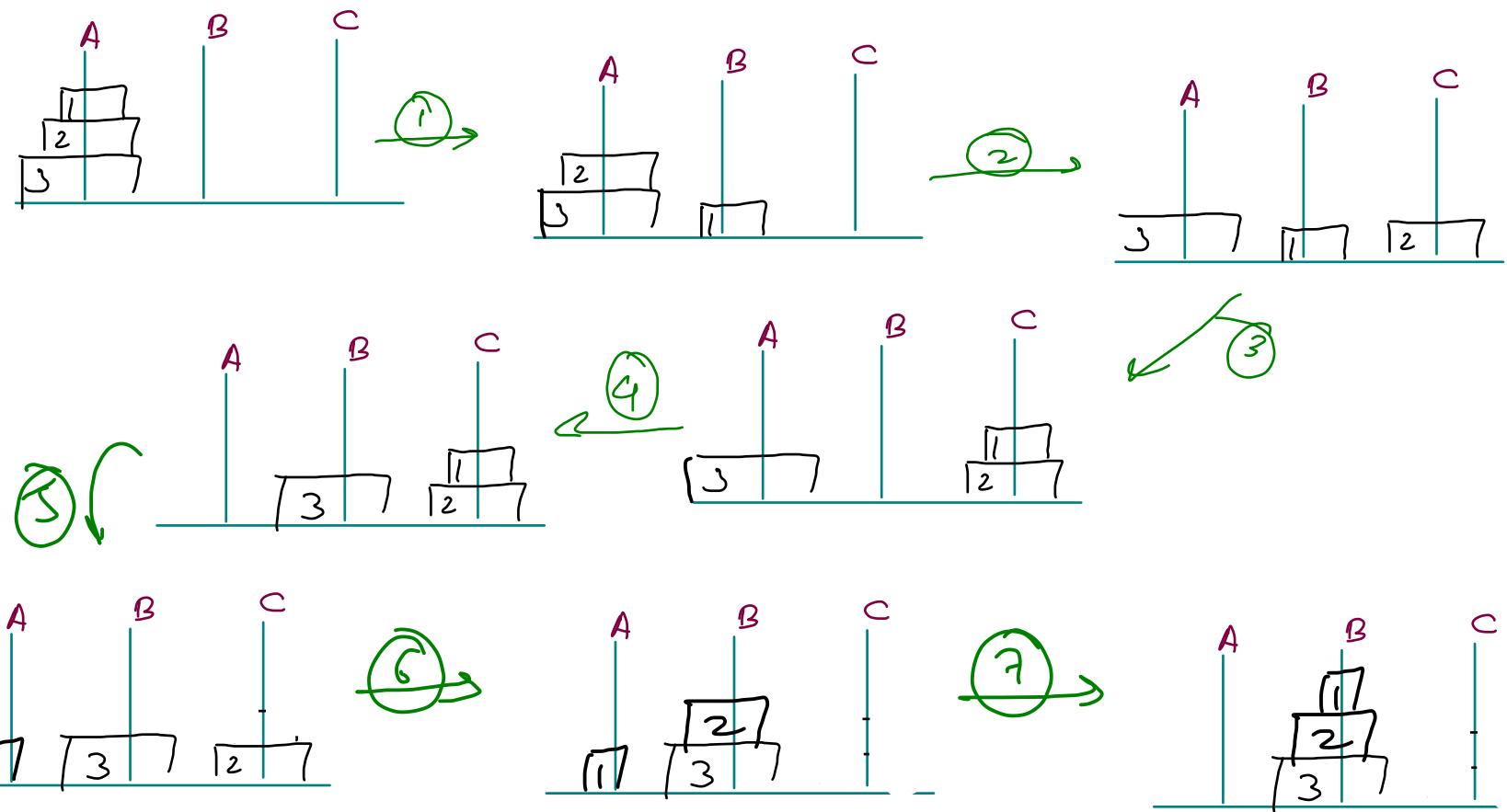


$\partial(n)$
Recursion
cell
stack
 \downarrow

Recursion
depth/
Recursion
tree
Height

$O(2^n)$

- 2 discs from A to C using B
- ① 1st from A to B
 ② 2nd from A to C
 ③ 1st from B to C
- Meeting expectation
- ④ 3rd from A to B
- 2 discs from C to B
- ⑤ 1st from C to A
 ⑥ 2nd from C to B
 ⑦ 1st from A to B



DSA-based Projects {Unique Project Ideas}

MERN

- ① Writing own library/header file for
 - B Tree, B+ Tree & SQL + DSA}
 - Segment Tree, Fenwick Tree, Number Theory Algorithms,
DSU, Hashmaps, Priority Queue/Heaps, Graph Algorithms
- {Competitive (Advanced DSA) + Generic Programming
Templates + DSFS}

②

Games

~~Webdev~~
→ CSS
→ JS

→ {+ 2 players}

- ~~Canvas~~ → Snake & ladder → Graphs Algo {DFS}
- Sudoku → Backtracking {+ levels/themes}
- Crossword Puzzle → Backtracking {+ levels/themes}
- Chess → 2D matrix {+ AI → Bot}
- Jump Game → Geriody + Dynamic Programming
{Frogs/Cartoon} {Climb Stairs}
- Mario Game

③ WhatsApp/Telegram Clone

- LRU Cache Algo \Rightarrow Ranking/ ordering of chats
 - Trie Data Structure \Rightarrow Searching Contacts/ Messages
prefix tree
 - CRUD operations { Firebase database }
 - Chatbots { APIs }
 - Audios/ Video calls
 - 1 to 1
 - To many \oplus
- } Networking

④ Splitwise App Clone

→ GFG

nosql
↑
restore
↑
↑

↳ Minimizing Cash flow Algorithm

↳ CRUD applications { persistent data (database),
+ authentication }

↳ System Design

↳ Priority Queue, Hashmap → Multiset Data
Structures

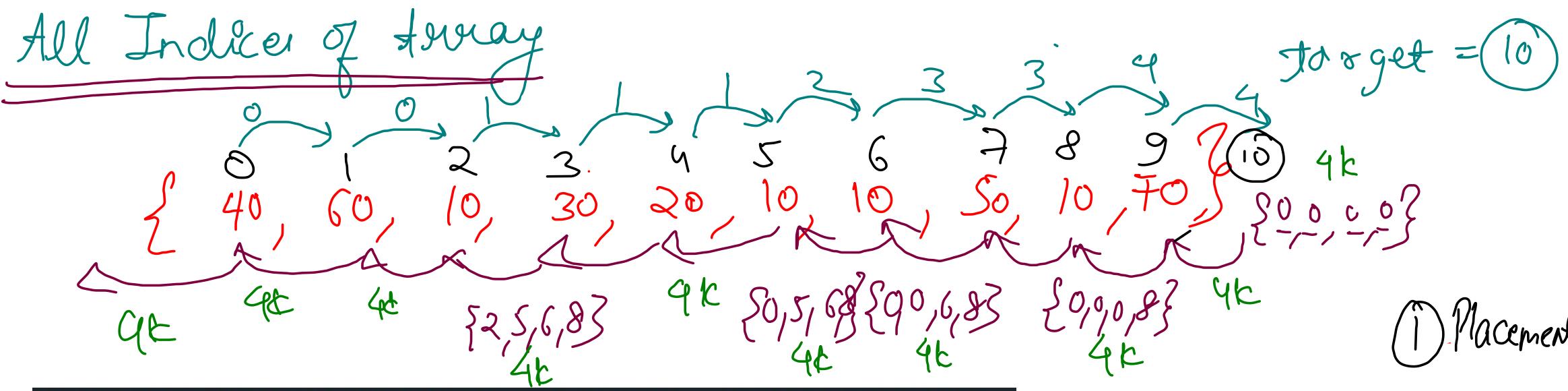
My Project :→ Flutter ⇒ LEN DEN

More Projects

{  } → Sorting visualizers
Path finding Algo visualizers

{  } → Text Editors : { Stack Data Structure }

{  } → File Zipping / Huffman Encoding
(TinyURL) URL Shortener



```

public static int[] allIndices(int[] arr, int target, int idx, int count) {
    if(idx == arr.length){
        int[] base = new int[count];
        return base;
    }
    if(arr[idx] == target){
        int[] res = allIndices(arr, target, idx + 1, count + 1);
        res[count] = idx;
        return res;
    } else {
        int[] res = allIndices(arr, target, idx + 1, count);
        return res;
    }
}

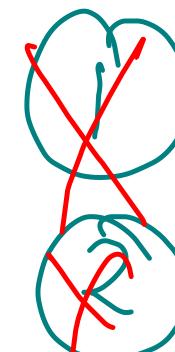
```

- ① Total occ
size array
return
- ② Occurrences
from left
part

$$(1)^N + \{k\} * N$$

$O(N)$

Codechef & codeforces



Binary Tree



BST



Trie



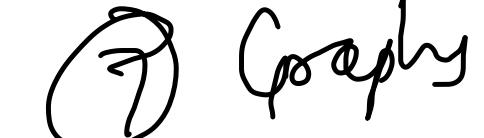
Number Theory



Bit



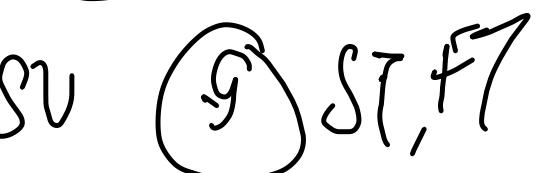
DP



Graph



DSU



ST, f7

Leetcode Virtual Contest



BT



SEO



A&S



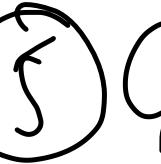
BST



SES



DP



Graph



R&B

R&B - Lecture ④

Recursion & ArrayList {Get}

- ① </> Get Subsequence
- ② </> Get Kpc
- ③ </> Get Stair Paths
- ④ </> Get Maze Paths
- ⑤ </> Get Maze Path With Jumps

Previous Classes

R&B - lecture ①
introduction

R&B - Lecture ②
Arrays

R&B - lecture ③
Tower of Hanoi, DSA based
projects.

Binary Search - lecture ① & ②

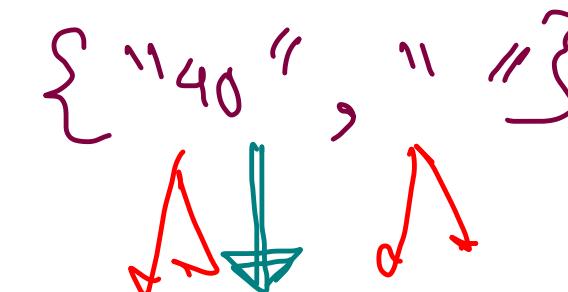
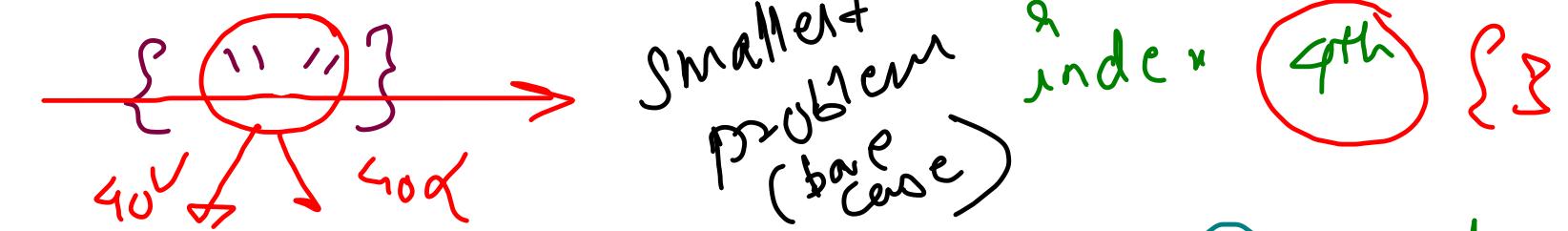
Resume Building & LinkedIn
Profile Building

Get Subsequence

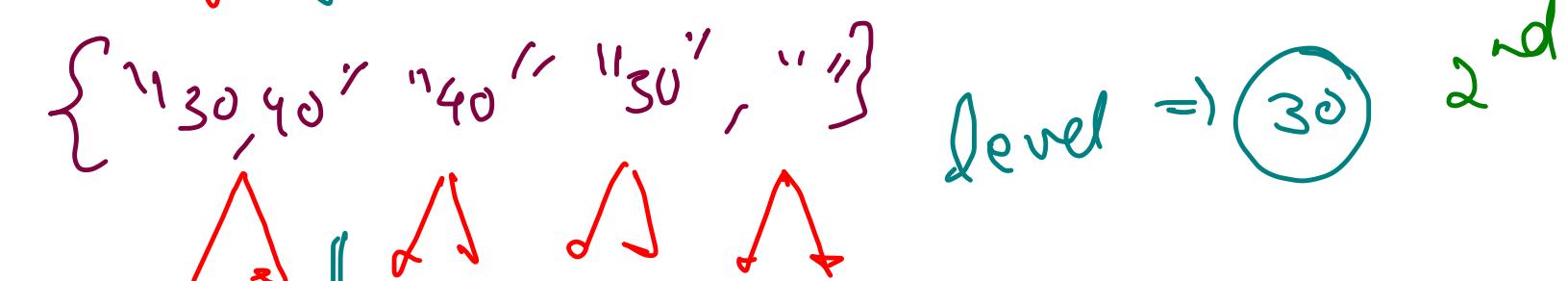
$$\left. \begin{array}{c} \downarrow \\ 10, 20, \\ 0 \end{array} \right\} , \left. \begin{array}{c} \uparrow \\ 30, 40 \\ 1 \end{array} \right\} ;$$

$\text{A}[S] \text{ getSS}(\underset{\uparrow}{\text{id}}, \alpha^0, \alpha^{4k})$ expected

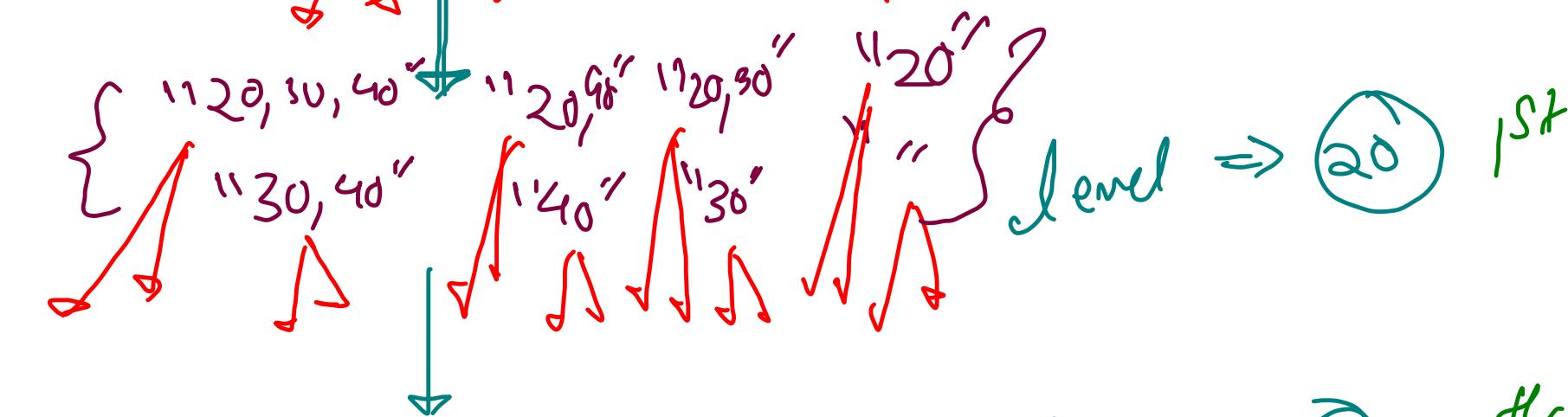
A \hookrightarrow get $\overset{?}{\leftarrow}$ (1, 4(c)) Fair



level = 40 3rd



Level \Rightarrow 30 2nd



level \Rightarrow 20 1st

largest problem { "10, 20, 30, 40" } "10, 30, 40" "10, 20, 40" "10, 40" level > 10 0
"20, 30, 40" "30, 40" "20, 40" "40"
"10, 20, 30" "20, 30" "10, 30" "30" "10 20" "20" "10" "" }

```

public static ArrayList<String> gss(int idx, string str)
{
    if(idx == str.length()){
        ArrayList<String> base = new ArrayList<>();
        base.add("");
        return base;
    }
    // Faith
    ArrayList<String> smallAns = gss(idx + 1, str);
    ArrayList<String> ans = new ArrayList<>();

    // No
    for(String smallSubset: smallAns){
        ans.add(smallSubset);
    }

    // Yes
    for(String smallSubset: smallAns){
        ans.add(str.charAt(idx) + smallSubset);
    }
    return ans;
}

```

4K { (0, 20), 30 }

gss(3, 4K)
 ↑ ↓ 6K
 gss(2, 4K)
 ↑ ↓ 8K
 gss(1, 4K)
 ↑ ↓ 10K
 gss(0, 4K)
 ↓ 12K

6K { " " }

8K { "", " 30" }

10K { "", " 30", " 20", " 2030" } 1st 20

12K { "", " 30", " 20", " 20, 30",
 " 10", " 11030", " 1020",
 " 102030" } 10

3rd (base case)

2nd 30

10

Get Keypad Combinations

"649"

```

0 -> .
1 -> abc
2 -> def
3 -> ghi
4 -> jkl
5 -> mno
6 -> pqrs
7 -> tu
8 -> vwx
9 -> yz
    
```

"649"
 ↗ ↗
 ↘ ↘
 ↗ ↗
 ↘ ↘
 ↗ ↗
 ↘ ↘

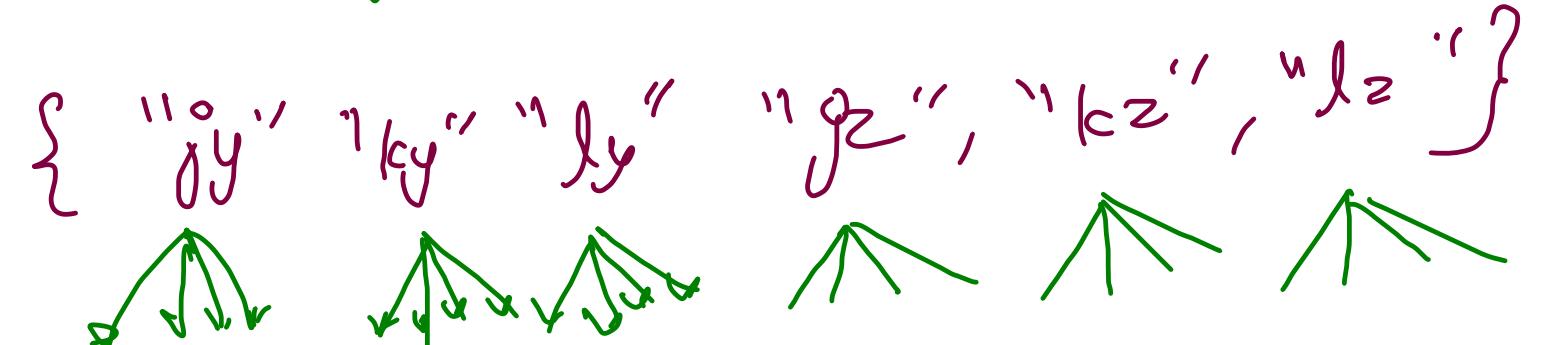
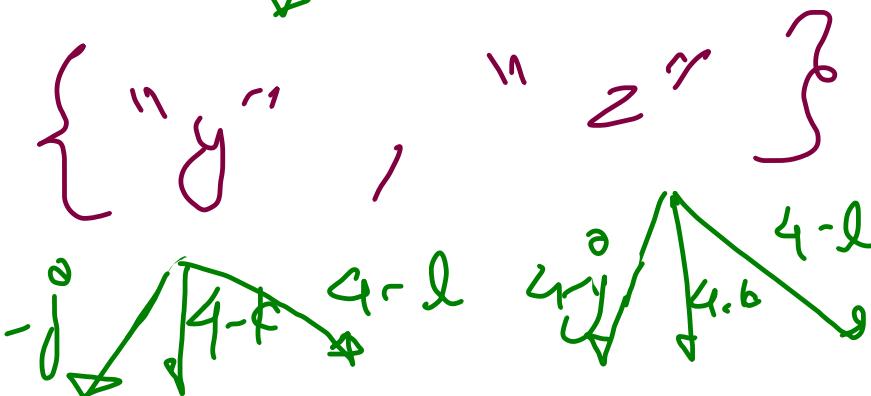
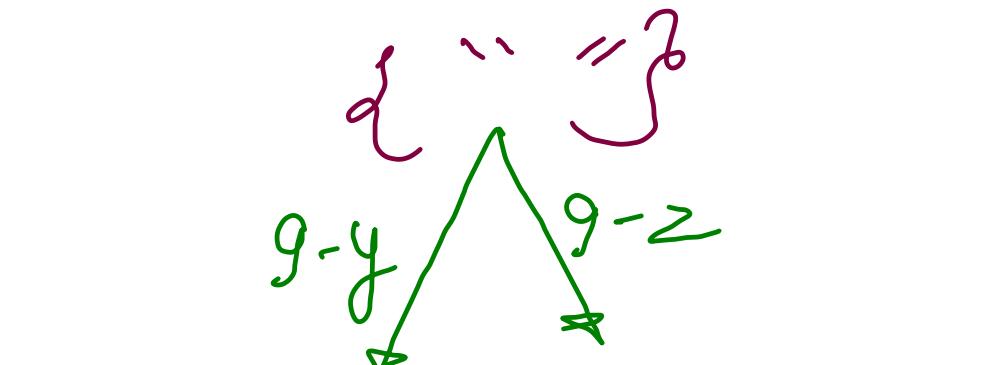
24
 ↗ ↗
 ↘ ↘

①

②

⑥

②4



{ 24 configurations . }

3rd (base case)

⑨ 2nd

④ 1st

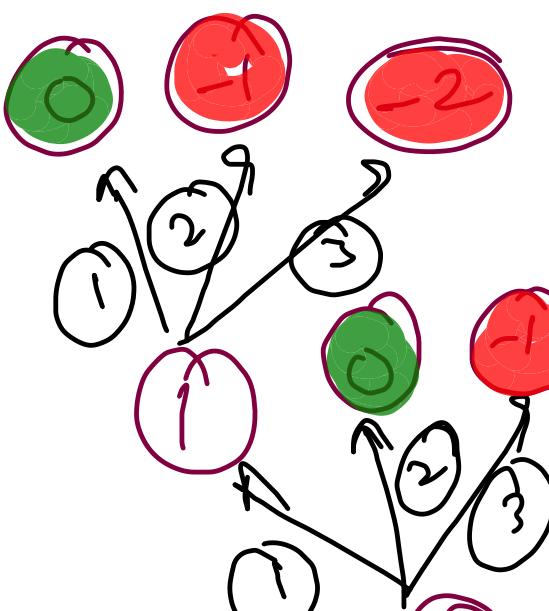
⑥ 0th

```
    static String[] dtoc = {"..", "abc", "def", "ghi", "jkl", "mno", "pqrs", "tu",
"vwx", "yz"};
```

```
public static ArrayList<String> getKPC(int idx, String str) {  
    if(idx == str.length()) {  
        ArrayList<String> base = new ArrayList<>();  
        base.add("");  
        return base;  
    }  
  
    // Faith  
    ArrayList<String> smallAns = getKPC(idx + 1, str);  
  
    ArrayList<String> ans = new ArrayList<>();  
    for(Character letter: dtoc[str.charAt(idx) - '0'].toCharArray()) {  
        for(String smallStr: smallAns) {  
            ans.add(letter + smallStr);  
        }  
    }  
    return ans;  
}
```

Get climb stairs

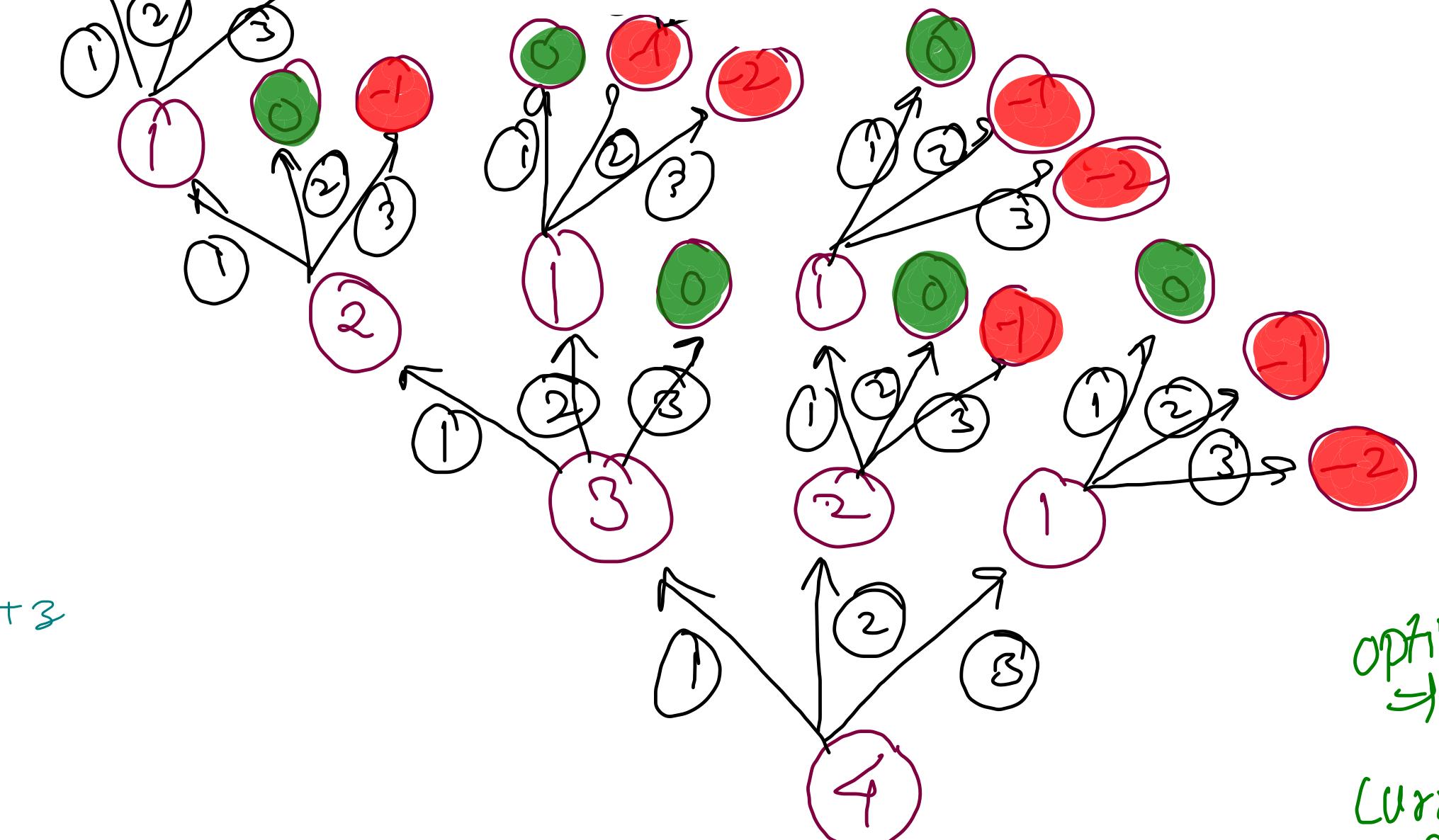
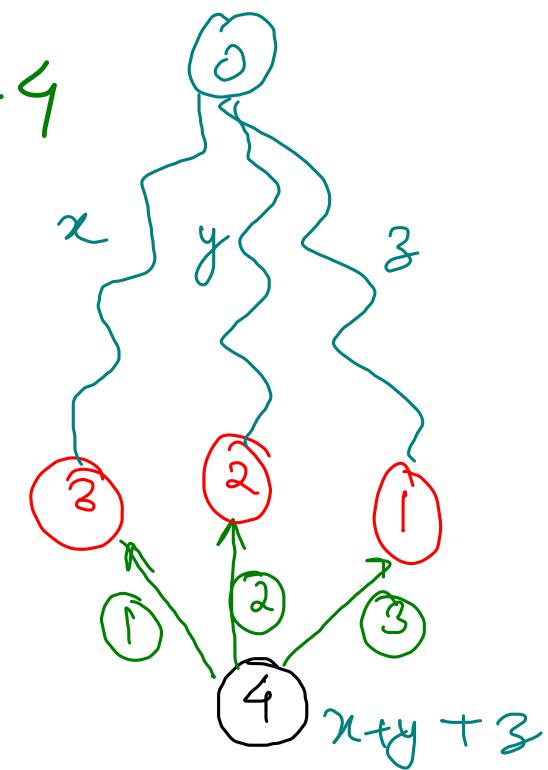
$$n = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$



A<S> gcs(n)

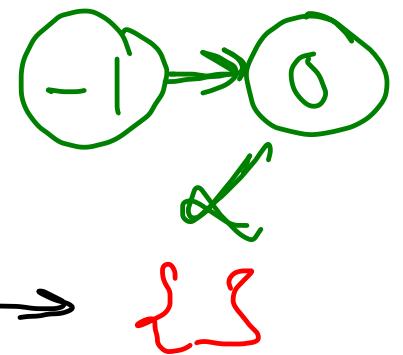
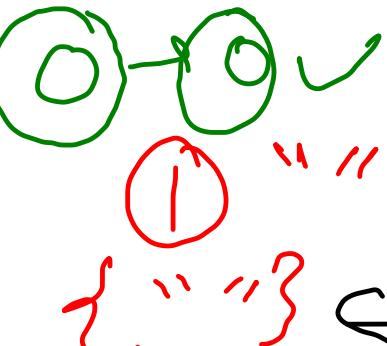
source $\Rightarrow n=4$

dest $\Rightarrow 0n$



option
 \Rightarrow move

current
Stair



```

public static ArrayList<String> getStairPaths(int n) {
    if(n == 0){
        // positive base case -> 1 size
        ArrayList<String> base = new ArrayList<>();
        base.add("");
        return base;
    } else if(n < 0){
        // negative base case -> 0 size
        ArrayList<String> base = new ArrayList<>();
        return base;
    }

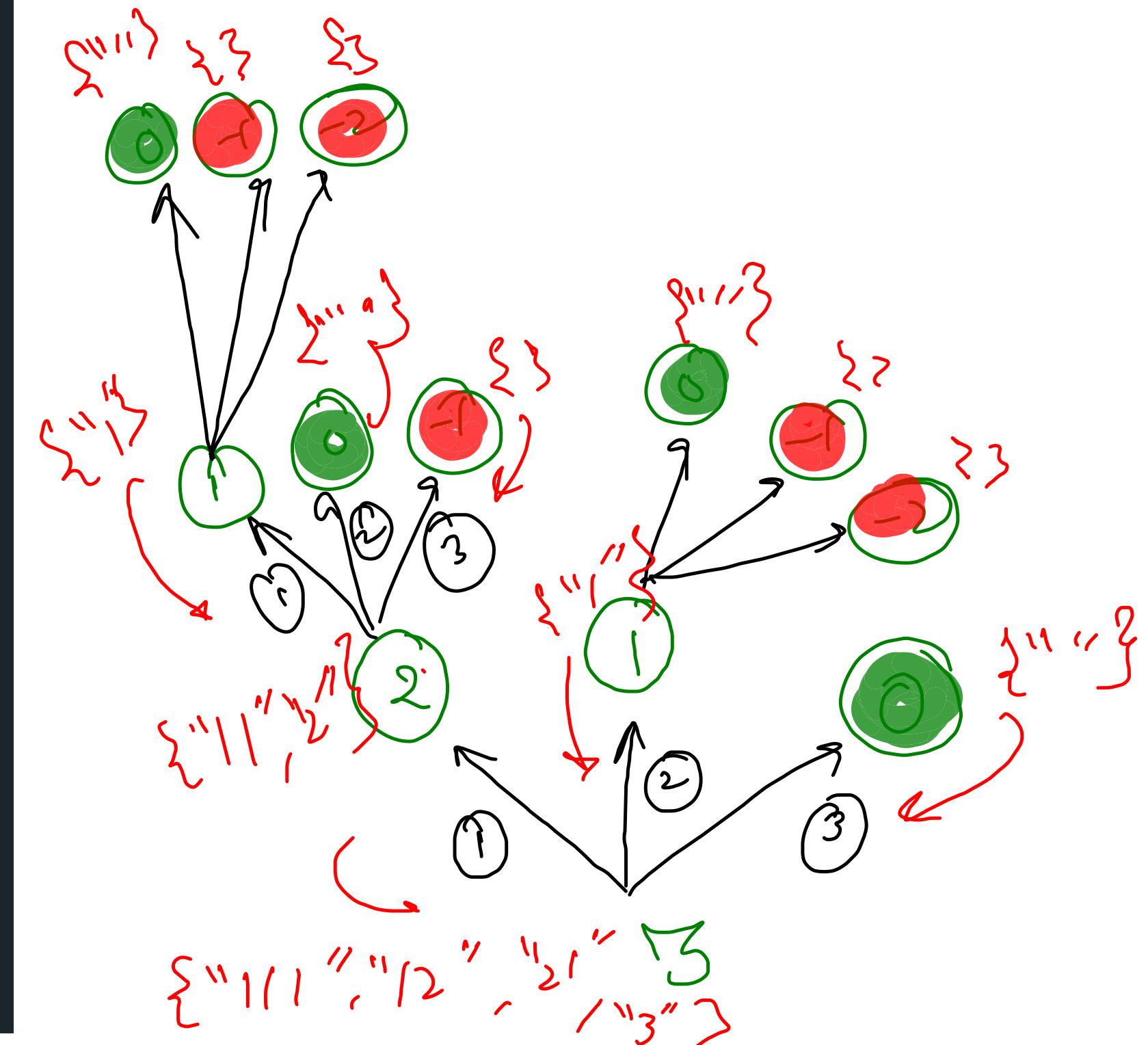
    ArrayList<String> smallAns1 = getStairPaths(n - 1);
    ArrayList<String> smallAns2 = getStairPaths(n - 2);
    ArrayList<String> smallAns3 = getStairPaths(n - 3);

    ArrayList<String> ans = new ArrayList<>();
    for(String str: smallAns1)
        ans.add(1 + str);

    for(String str: smallAns2){
        ans.add(2 + str);
    }

    for(String str: smallAns3){
        ans.add(3 + str);
    }
}

```



```

public static ArrayList<String> getStairPaths(int n) {
    if(n == 0){
        // positive base case -> 1 size
        ArrayList<String> base = new ArrayList<>();
        base.add("");
        return base;
    }

    ArrayList<String> ans = new ArrayList<>();

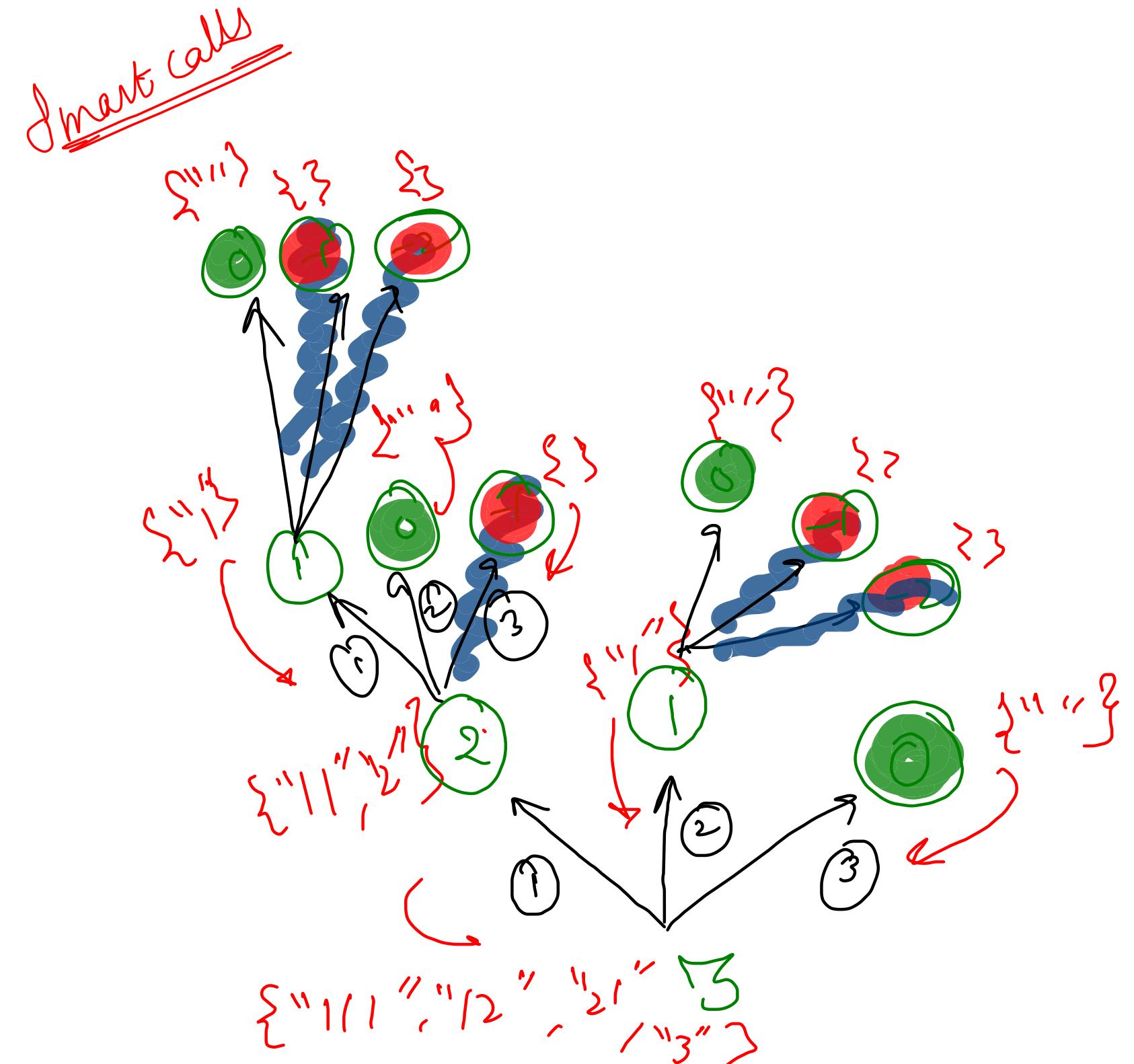
    if(n - 1 >= 0){
        ArrayList<String> smallAns1 = getStairPaths(n - 1);
        for(String str: smallAns1)
            ans.add(1 + str);
    }

    if(n - 2 >= 0) {
        ArrayList<String> smallAns2 = getStairPaths(n - 2);
        for(String str: smallAns2){
            ans.add(2 + str);
        }
    }

    if(n - 3 >= 0) {
        ArrayList<String> smallAns3 = getStairPaths(n - 3);
        for(String str: smallAns3){
            ans.add(3 + str);
        }
    }

    return ans;
}

```



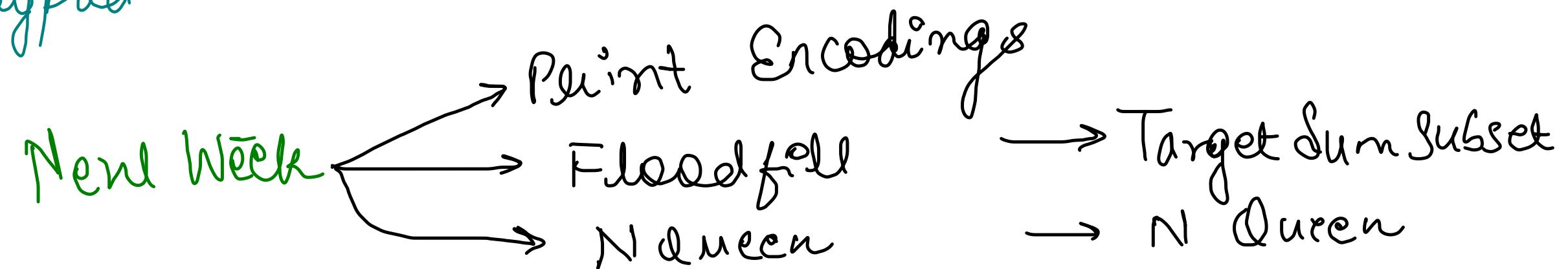
Today's Questions

- ① Get maze Paths
- ② Get maze Paths with Jumps
- ③ Print subsequences
- ④ Print Keypad

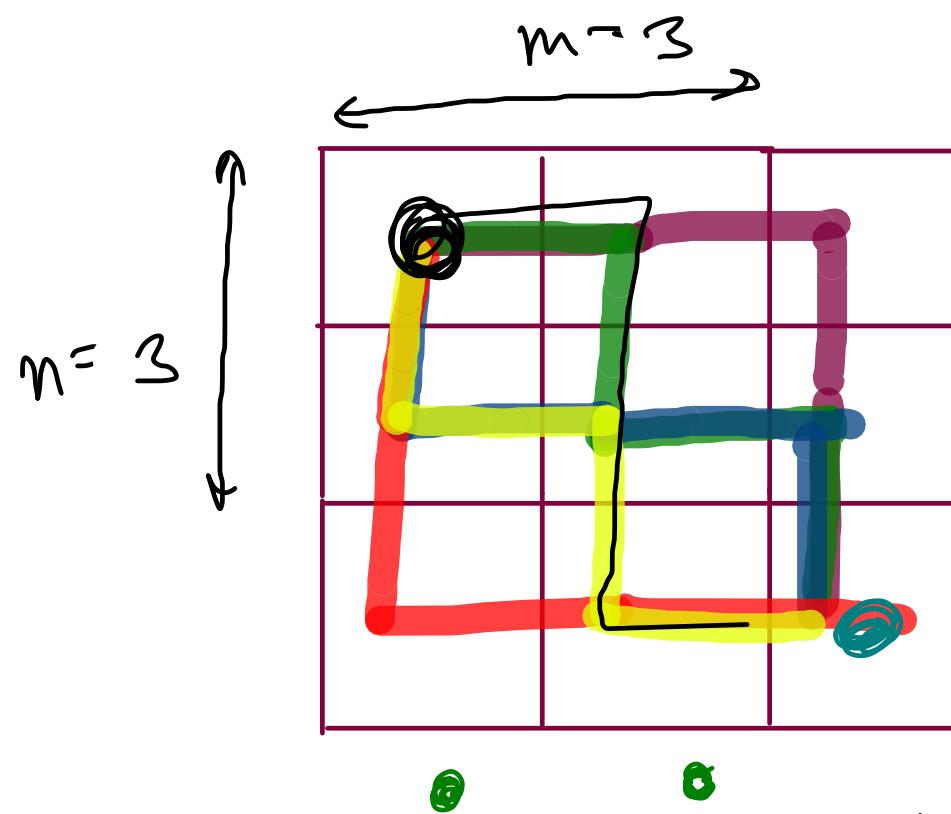
- ⑤ Print Star Paths
- ⑥ Print Maze Paths

- ⑦ Print Maze Paths with jumps (HW)

- ⑧ Print Permutations



Get Maze Paths



"rddo"
"rdrd"
"drrd"
"drdo"
"ddrs"
"rddr"

A < S > getMPaths(int sr, int sc,
int dr, int dc)

6 paths
with each path
of length 4.

$$\frac{(n-1) + (m-1)}{\text{total no. of downs} (v)}$$

Path length (4)

↓

total no. of rights (h)

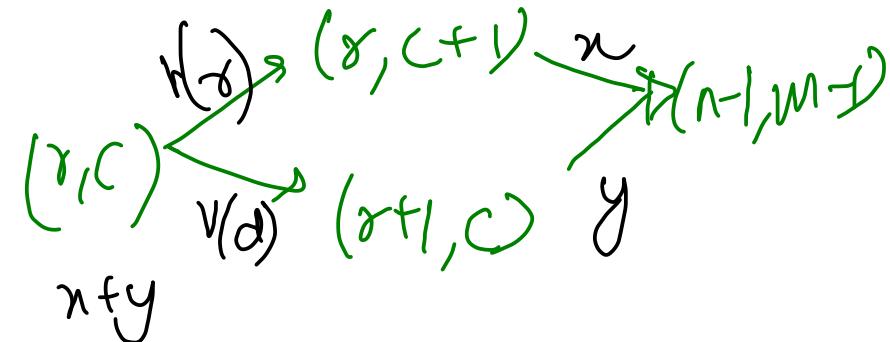
$$\frac{(n-1 + m-1)!}{(n-1)!(m-1)!}$$

No. of paths (6)

level \Rightarrow cell

options \Rightarrow neighbours

right cell down cell



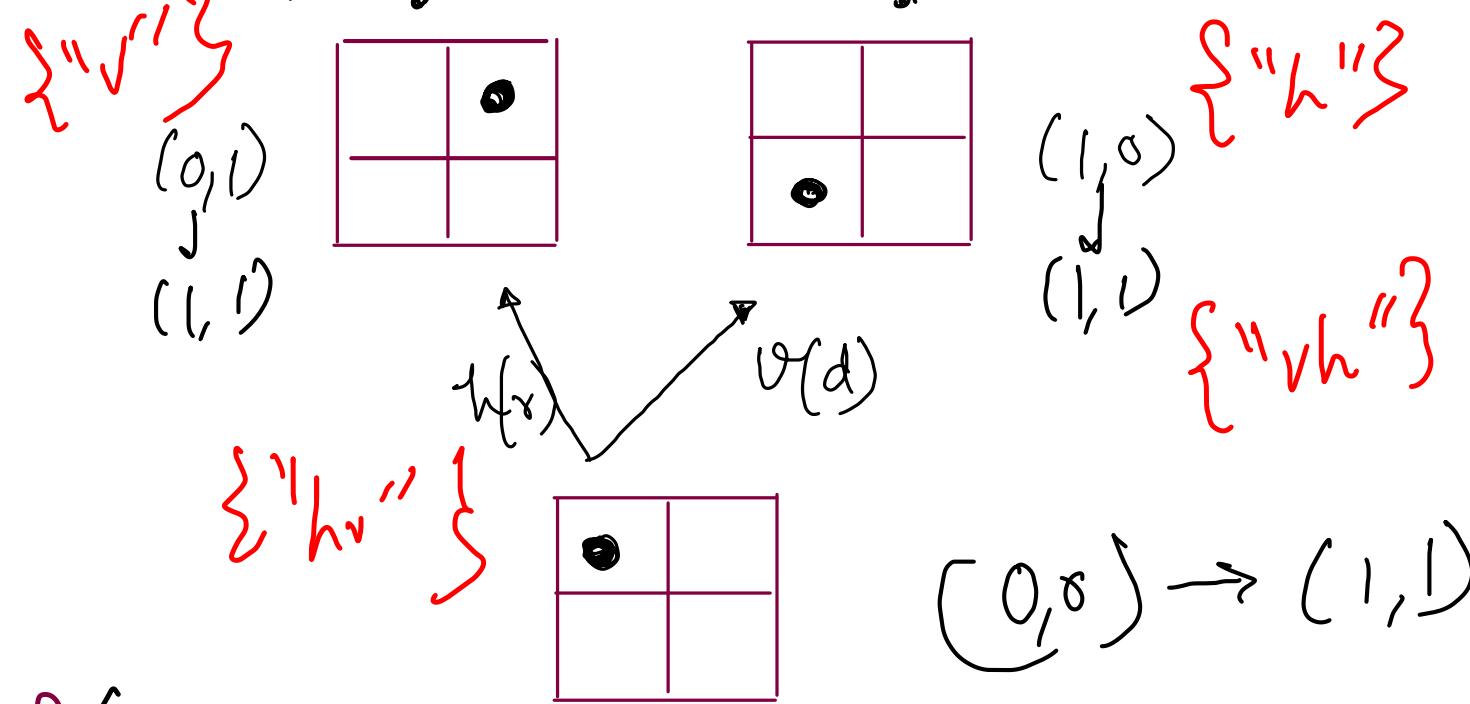
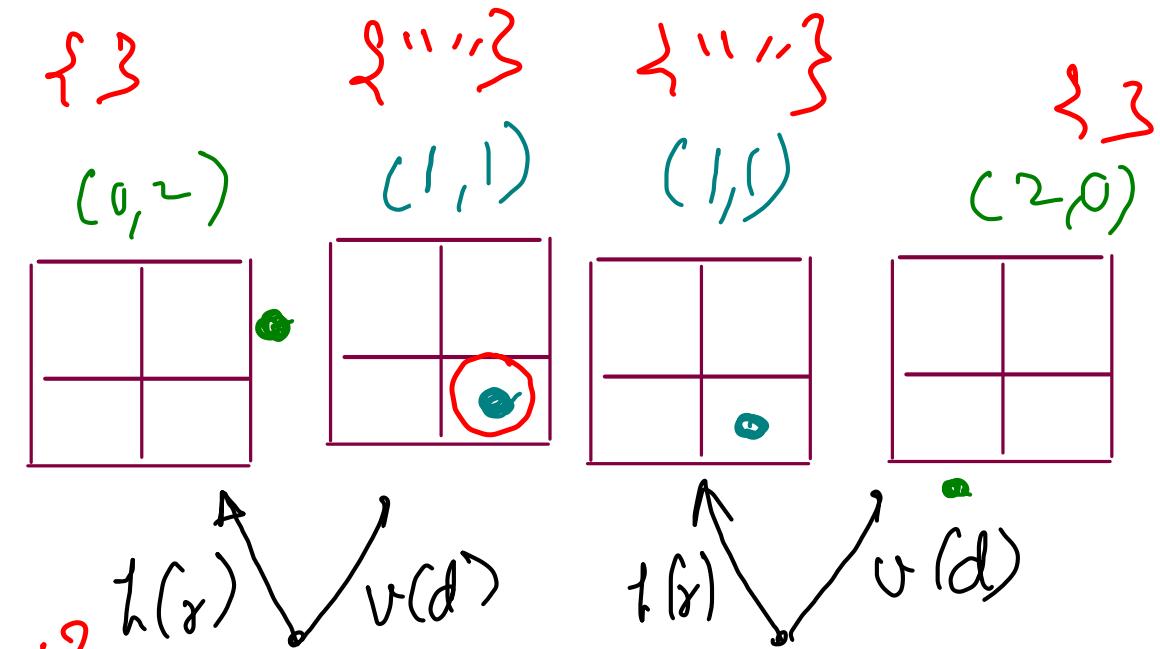
① -ve base case

② +ve base case

③ faith.

~~Path length~~ $\xrightarrow{\text{greedy}}$

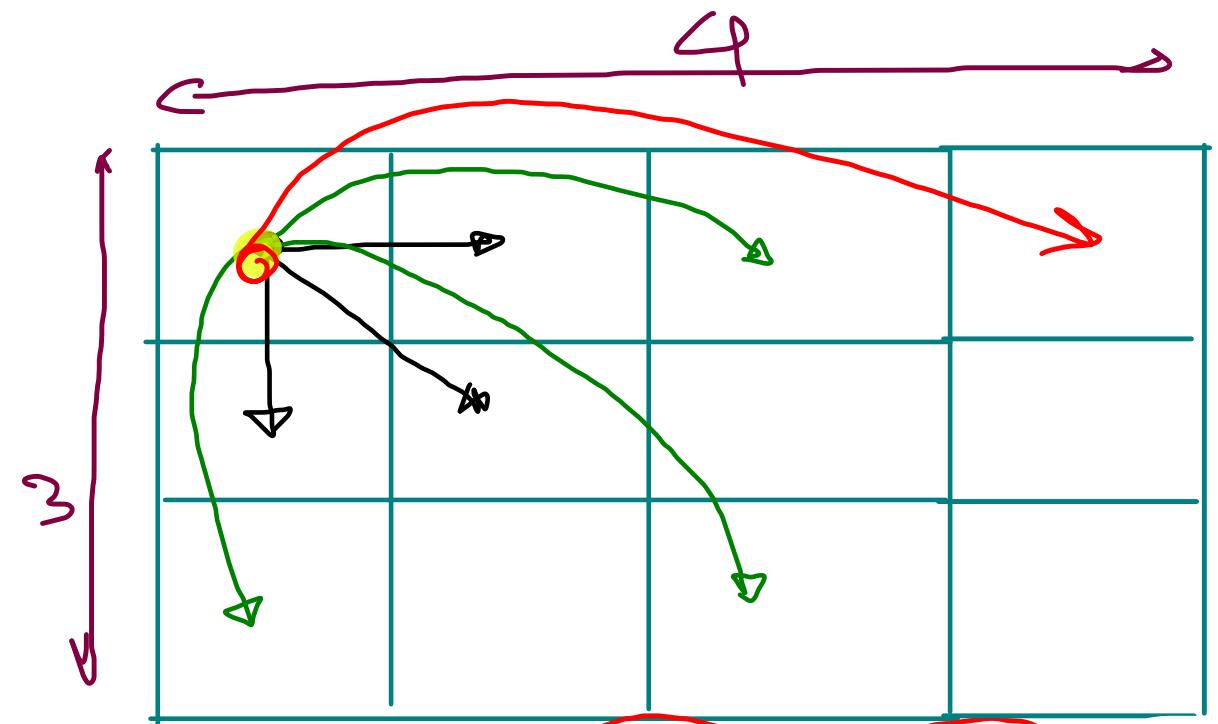
~~No of paths~~ $\xrightarrow{\text{DP}}$ Catalan



$(0, 0) \rightarrow (1, 1)$

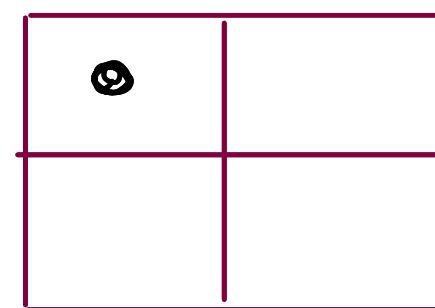
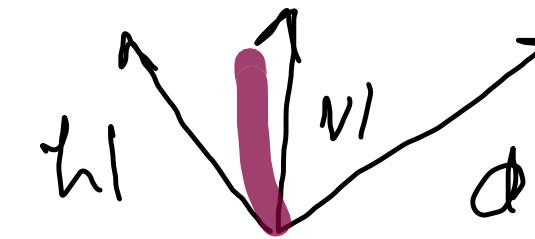
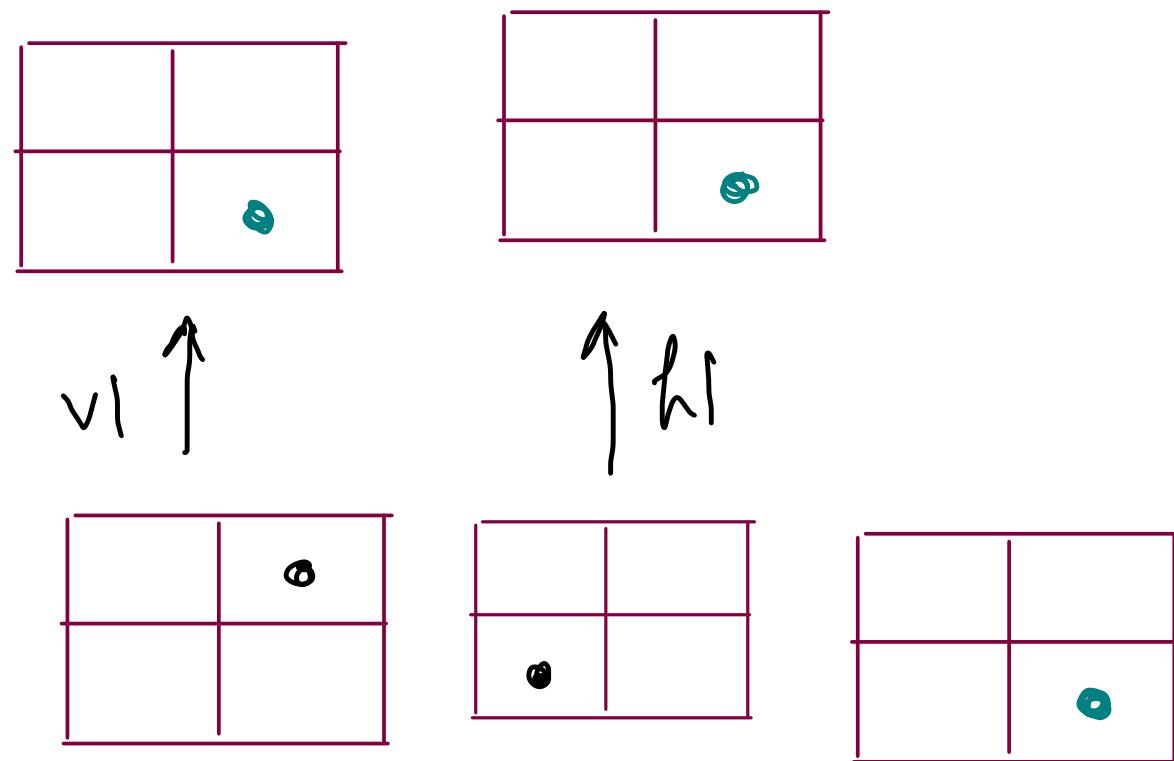
```
public static ArrayList<String> getMazePaths(int sr, int sc, int dr, int dc) {  
    if(sr > dr || sc > dc){  
        // negative base case  
        ArrayList<String> base = new ArrayList<>();  
        return base;  
    }  
  
    if(sr == dr && sc == dc){  
        // positive base case  
        ArrayList<String> base = new ArrayList<>();  
        base.add("");  
        return base;  
    }  
  
    ArrayList<String> res = new ArrayList<>();  
  
    ArrayList<String> rightPaths = getMazePaths(sr, sc + 1, dr, dc);  
    for(String str: rightPaths){  
        res.add('h' + str);  
    }  
  
    ArrayList<String> downPaths = getMazePaths(sr + 1, sc, dr, dc);  
    for(String str: downPaths){  
        res.add('v' + str);  
    }  
  
    return res;  
}
```

Get Maze Path with jumps



$$\{ \{ h_1, h_2, h_3 \}, \{ v_1, v_2 \}, \{ d_1, d_2 \} \} \\ n + m + \min(r, m)$$

$$O((n+m)^{n+m}) \Rightarrow \underline{\text{exponential}}$$



```
public static ArrayList<String> getMazePaths(int sr, int sc, int dr, int dc) {  
    if(sr == dr && sc == dc){  
        // positive base case  
        ArrayList<String> base = new ArrayList<>();  
        base.add("");  
        return base;  
    }  
  
    ArrayList<String> res = new ArrayList<>();  
  
    // Horizontal Path  
    for(int jump = 1; sc + jump <= dc ; jump++){  
        ArrayList<String> rightPaths = getMazePaths(sr, sc + jump, dr, dc);  
        for(String str: rightPaths){  
            res.add("h" + jump + str);  
        }  
    }  
  
    // Vertical Path  
    jump = 1;  
    while(sr + jump <= dr){  
        ArrayList<String> downPaths = getMazePaths(sr + jump, sc, dr, dc);  
        for(String str: downPaths){  
            res.add("v" + jump + str);  
        }  
        jump++;  
    }  
  
    // Diagonal Path  
    jump = 1;  
    while(sr + jump <= dr && sc + jump <= dc){  
        ArrayList<String> rightPaths = getMazePaths(sr + jump, sc + jump, dr, dc);  
        for(String str: rightPaths){  
            res.add("d" + jump + str);  
        }  
        jump++;  
    }  
  
    return res;  
}
```

Print Subsequences

void printSS(0, arr)

DFS
↓
Recursion

```

    printSS(0, input, "");
}

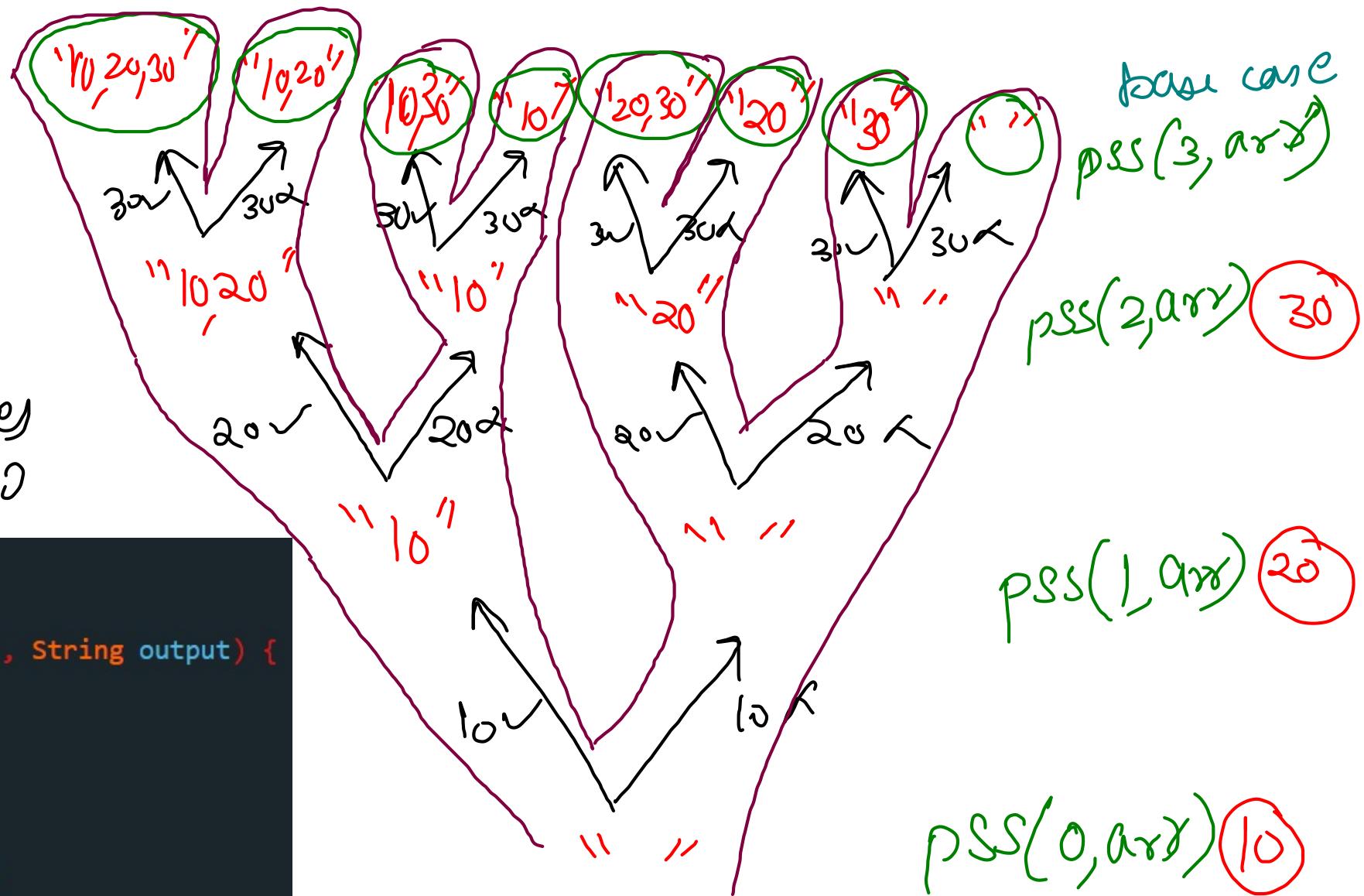
public static void printSS(int idx, String input, String output) {
    if(idx == input.length()){
        System.out.println(output);
        return;
    }

    char ch = input.charAt(idx);
    printSS(idx + 1, input, output + ch); // yes
    printSS(idx + 1, input, output); // no
}

```

{0 1 2
10, 20, 30}

Item on level
Options Yes No



$$2^N + \{k + O\}^N \Rightarrow O(2^N)$$

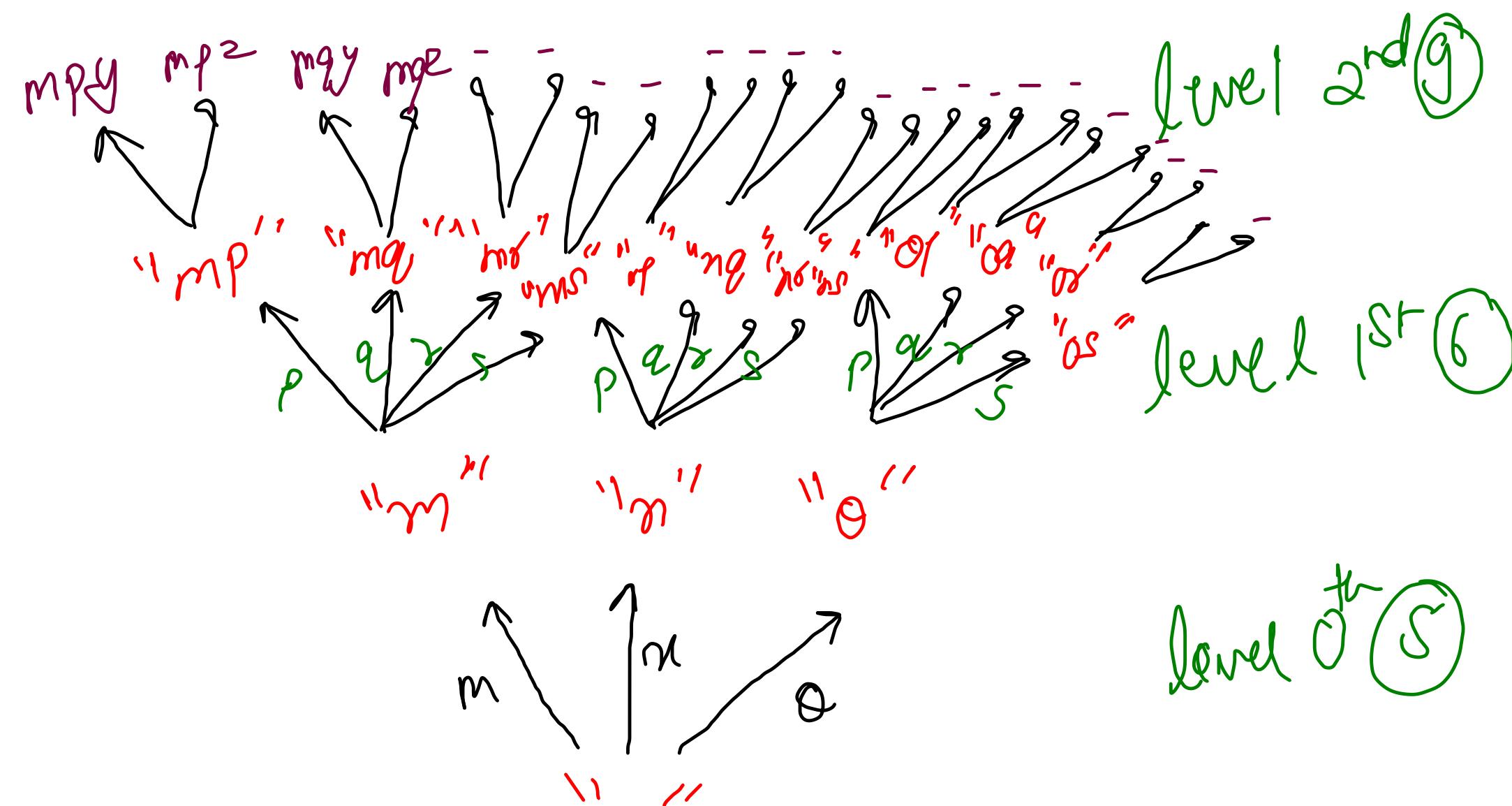
Point Reciprocate Comb

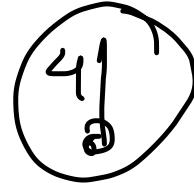
"JGG"

$$(4)^N + \{k\} * N$$

$$\Rightarrow O(4^N)$$

↓
exponential



Point Permutations {Distinct letters} "abcd" \Rightarrow 

"abcd"	"bacd"	"cabd"	"dabc"
"abdc"	"badc"	"cadb"	"dacb"
"acbd"	"bcad"	"cbad"	"dcac"
"acdb"	"bcda"	"cbda"	"dbca"
"adbc"	"bdac"	"cdab"	"dcab"
"adcb"	"bdca"	"cdba"	"dcba"

$$(N)^N + \{N + O\}^N$$

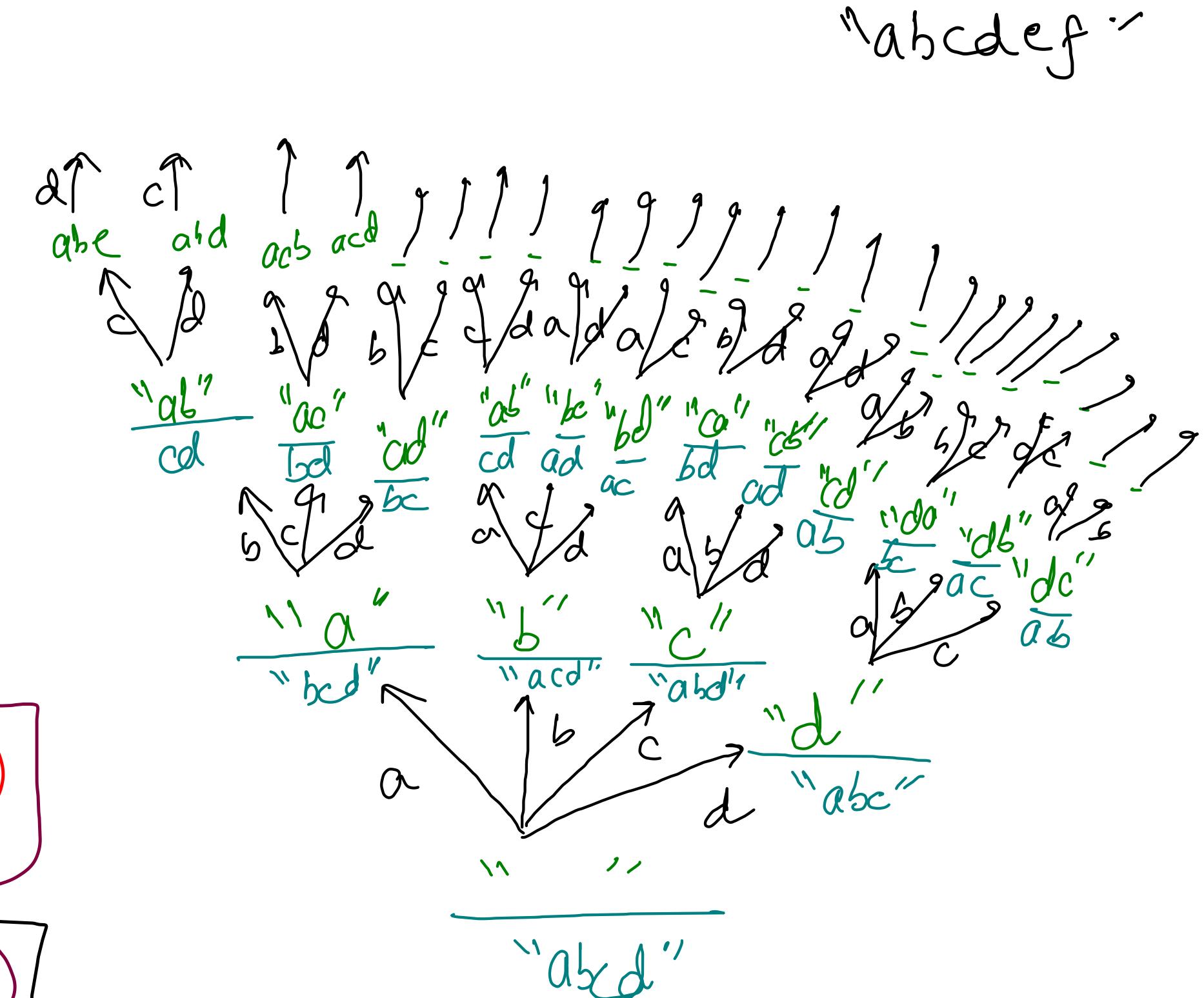
$N^N + N^2$

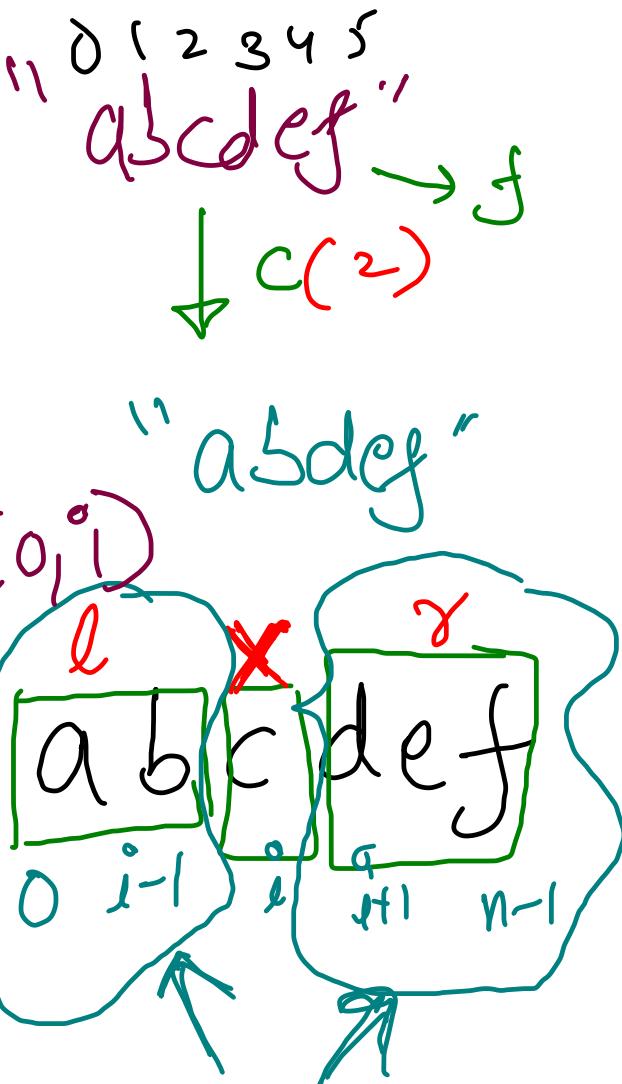
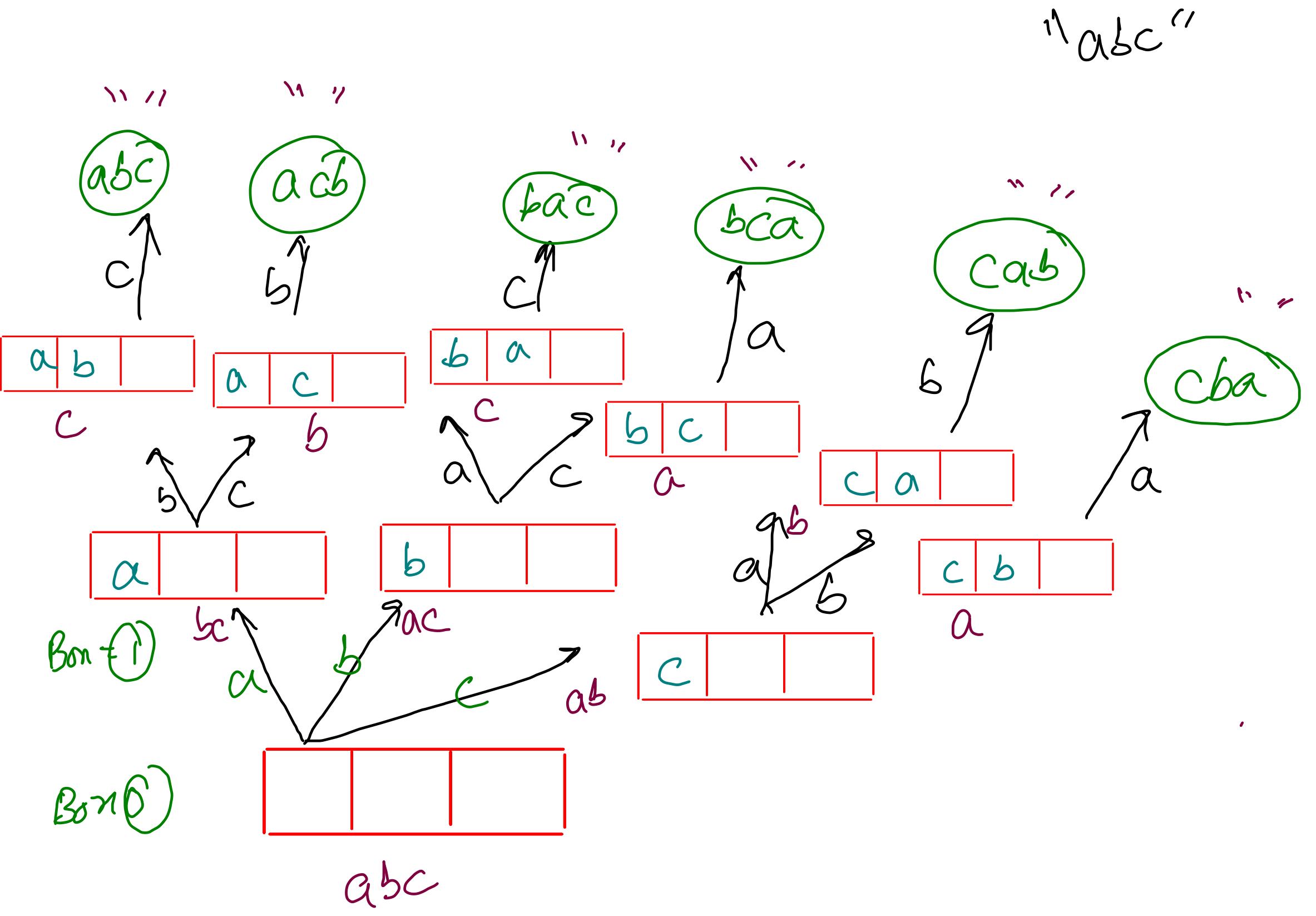
Or

$O(N^N)$

$O(n!)$

30x on level
Items as option





left = str.substring(0, i)
right = str.substring(i+1)

```

public static void main(String[] args) throws Exception {
    Scanner scn = new Scanner(System.in);
    String str = scn.next();
    printPermutations(str, "");
}

public static void printPermutations(String input, String output) {
    if(input.length() == 0){
        System.out.println(output);
        return;
    }

    for(int i=0; i<input.length(); i++){ [0, i)
        String newInput = input.substring(0, i) + input.substring(i + 1);
        printPermutations(newInput, output + input.charAt(i));
    }
}

```

abcdef
↓ f
left
0, 0 ..
(6)
i ↴

[i+1, n-1]

Tentative Schedule (FJP-2) DSA - level ①

- ① Conceptual Building Blocks → Complete
- ② Recursion & Backtracking → Complete
- ③ Searching & Sorting
Time & Space Complexity] → 3 lectures
- ④ Dynamic Programming → 8 lectures
- ⑤ linked list → 5 lectures
- ⑥ Stack & Queue, DOPS → 5 lectures
- ⑦ Generic Tree → 5 lectures
- ⑧ Binary Tree & BST → 5 lectures
- ⑨ Hashmap & Heap → 5 lectures
- ⑩ Graphs → 5 lectures

Appximate
45 lectures

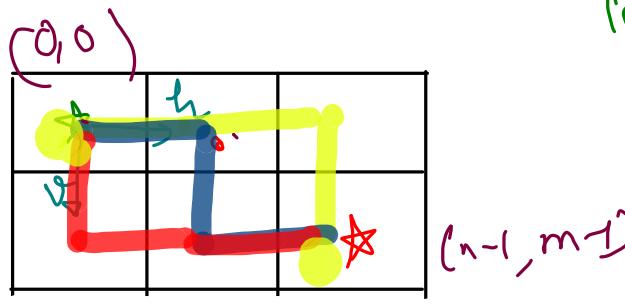
9 weeks if 4 classes/week
(March 10)

7 weeks if 5 classes/week
(Feb 15)

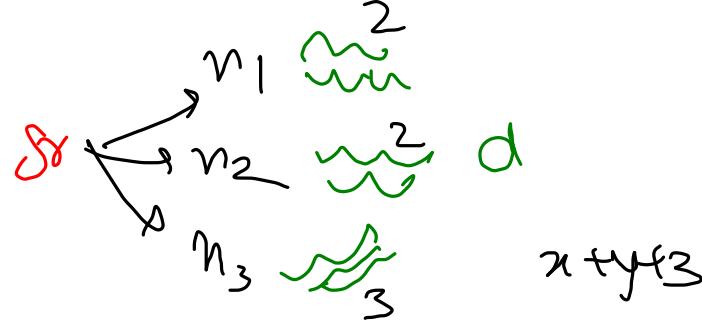
Recursion (Point \Rightarrow on the way up)

- ① Point Stair Case Paths
- ② Point Maze Paths
- ③ Point maze Paths with jumps (HW)
- A④ Point Encodings

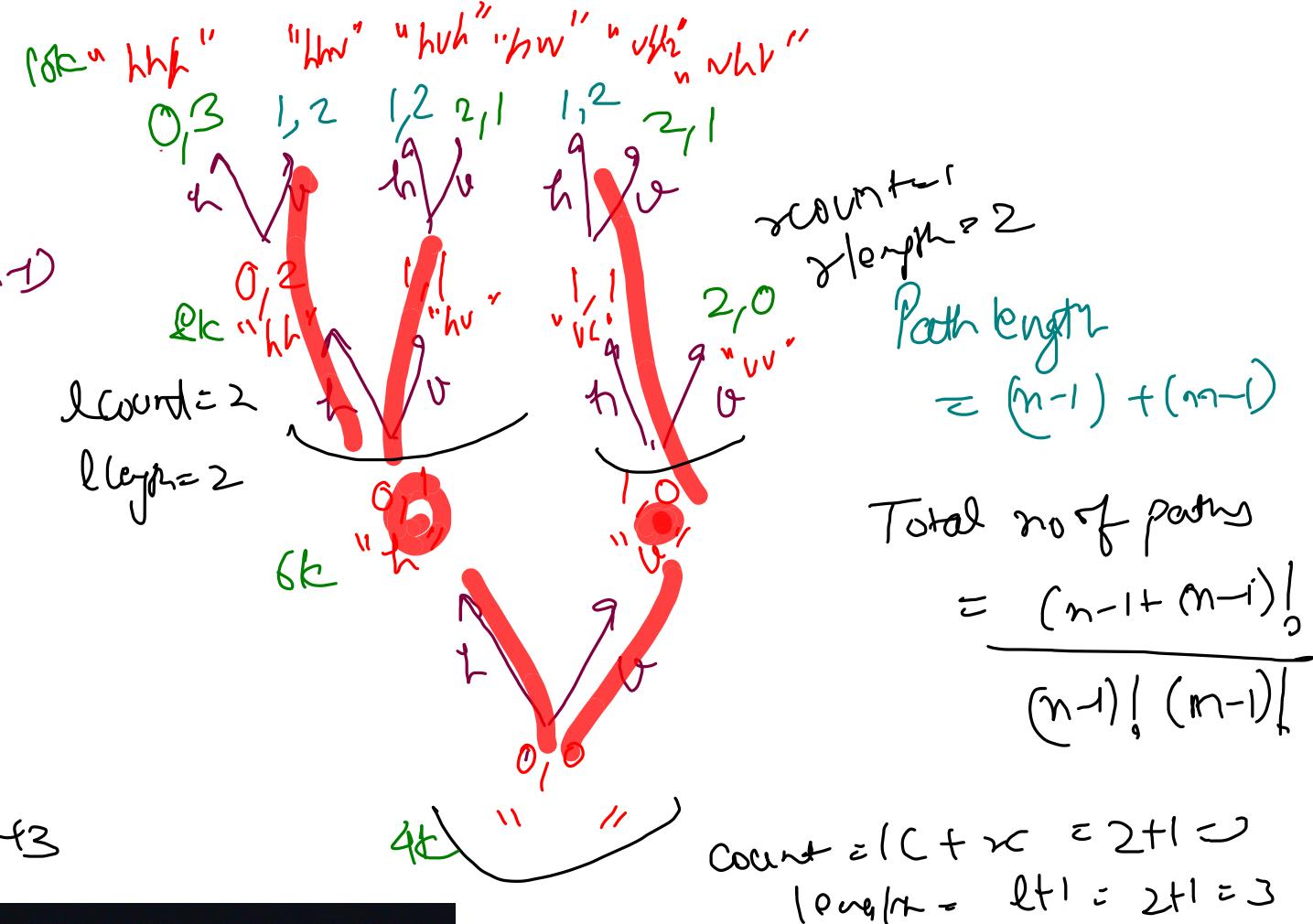
Print Maze Paths



DFS \Leftarrow Recursion



```
public static void printMazePaths(int sr, int sc, int dr, int dc, String psf) {  
    if(sr > dr || sc > dc){  
        return;  
    }  
  
    if(sr == dr && sc == dc){  
        System.out.println(psf);  
        return;  
    }  
  
    printMazePaths(sr, sc + 1, dr, dc, psf + "h");  
    printMazePaths(sr + 1, sc, dr, dc, psf + "v");  
}
```



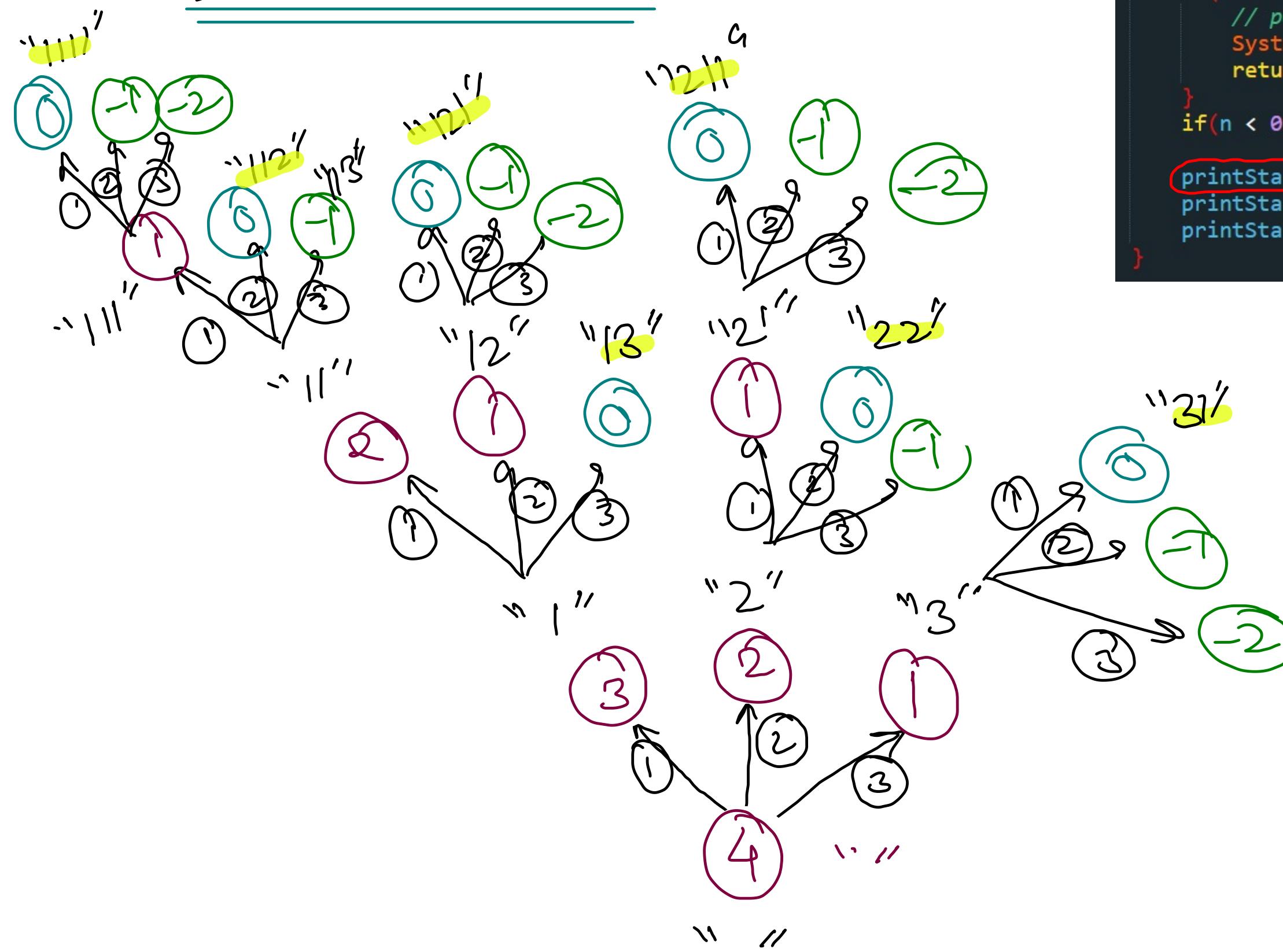
$$= \frac{(n-1+m-i)!}{(n-1)! (m-1)!}$$

$$\text{Count} = (C + x) \leq 2t + 1$$

$$\log_2(x) = t + 1 \leq 2t + 1 = 3$$

$$\Theta(2^{n+m})$$

Print Stair Paths



```

public static void printStairPaths(int n, String path) {
    if(n == 0){
        // positive base case
        System.out.println(path);
        return;
    }
    if(n < 0) return; // negative base case

    printStairPaths(n - 1, path + "1");
    printStairPaths(n - 2, path + "2");
    printStairPaths(n - 3, path + "3");
}

```

$$3^N + (k)^N$$

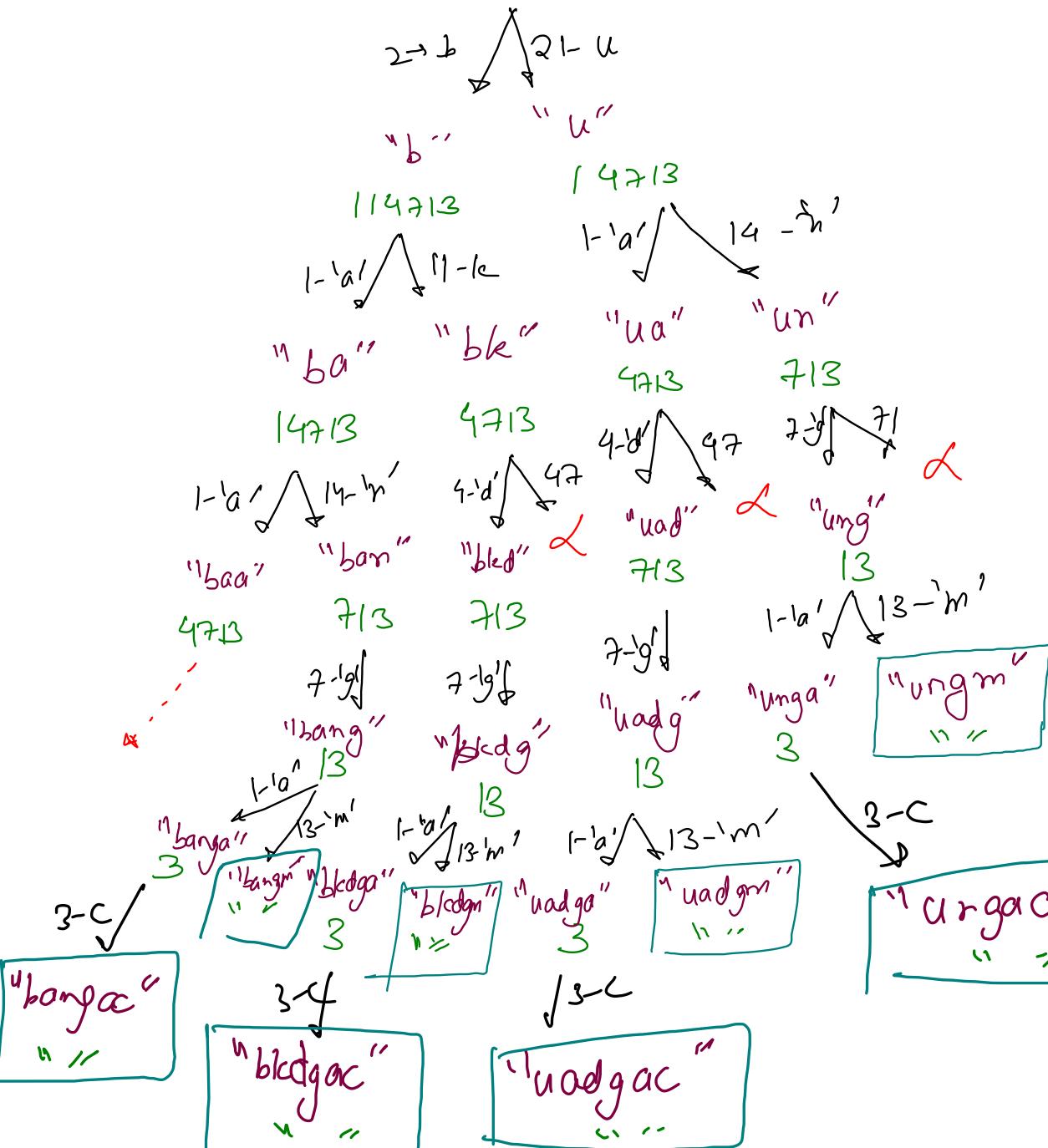
$$\Rightarrow O(3^N)$$

Point Encodings

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
 a b c d e f g h i j k l m n o p q r s t u v w x y z

Example ① "baungac"

2 1 1 4 7 1 3



str[0] == '0' return;

1 char - word

> 26 \Rightarrow 2 length

$$(2)^N + \{k+k\}^{\log N}$$

$O(2^N)$

Example ②

4610208

" "

↓ 4-'d'

610208

" d"

↓ 6-'f'

10208

" df"

↓ 1-'a'

↓ 10-'j'

208
↓ 2-'b'

↓ 2-'f'

" dfj"

↓ 2-'f'

↓ 8

↓ 8-'h'

" dfjh"

Example ③

098

0 1 09
↓ ↓

No possible encoding

```
public static void printEncodings(int idx, String input, String output) {  
    if(idx == input.length()) {  
        // positive base case  
        System.out.println(output);  
        return;  
    }  
  
    // corner cases  
    int ch1 = input.charAt(idx) - '0';  
    if(ch1 >= 1 && ch1 <= 9){  
        printEncodings(idx + 1, input, output + (char)('a' + ch1 - 1));  
    }  
  
    if(idx + 1 < input.length()) {  
        int ch2 = (input.charAt(idx) - '0') * 10 + (input.charAt(idx + 1) - '0');  
  
        if(ch2 >= 10 && ch2 <= 26){  
            printEncodings(idx + 2, input, output + (char)('a' + ch2 - 1));  
        }  
    }  
}
```

Target Sum Subset

0 1 2 3
 $\{10, 20, 30, 40\}$

target = 50

Reusing

$O(2^n)$

* distinct
 * positive

$\{10, 20, 30, 40\}$
 $\{-50, -10, -20, -30, 10, 0, 40\}$

$\{10, 20, 30\}$
 $\{0, 40\}$

$\{10, 20, 30\}$
 $\{0, 40\}$

$\{10, 20\}$
 $\{0\}$
 $\{20\}$
 $\{30\}$
 $\{30, 40\}$

$\{10\}$
 $\{20\}$
 $\{20, 40\}$

$\{10, 40\}$
 $\{\}, 50$

$\{\}, 50$

40

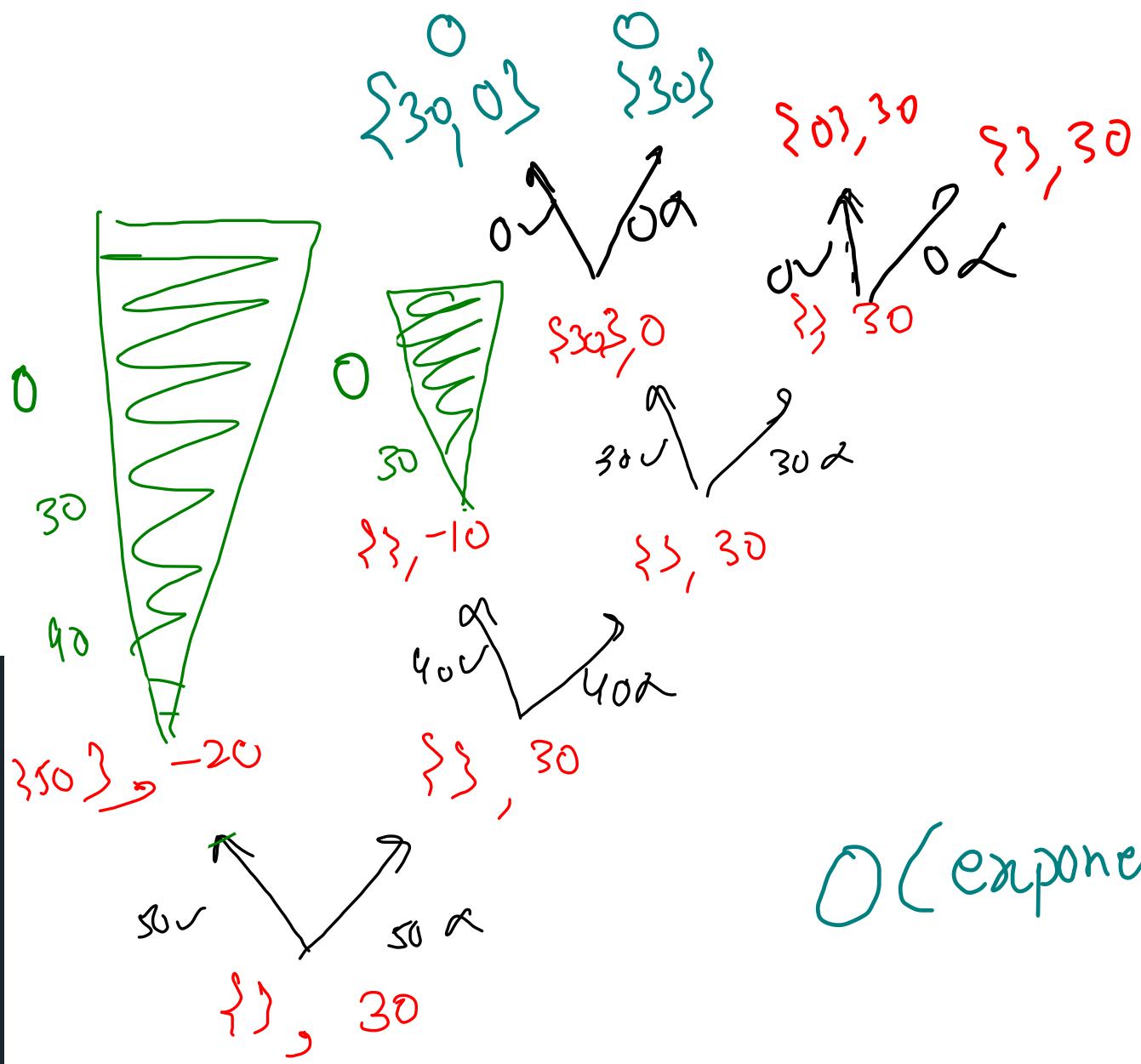
30

20

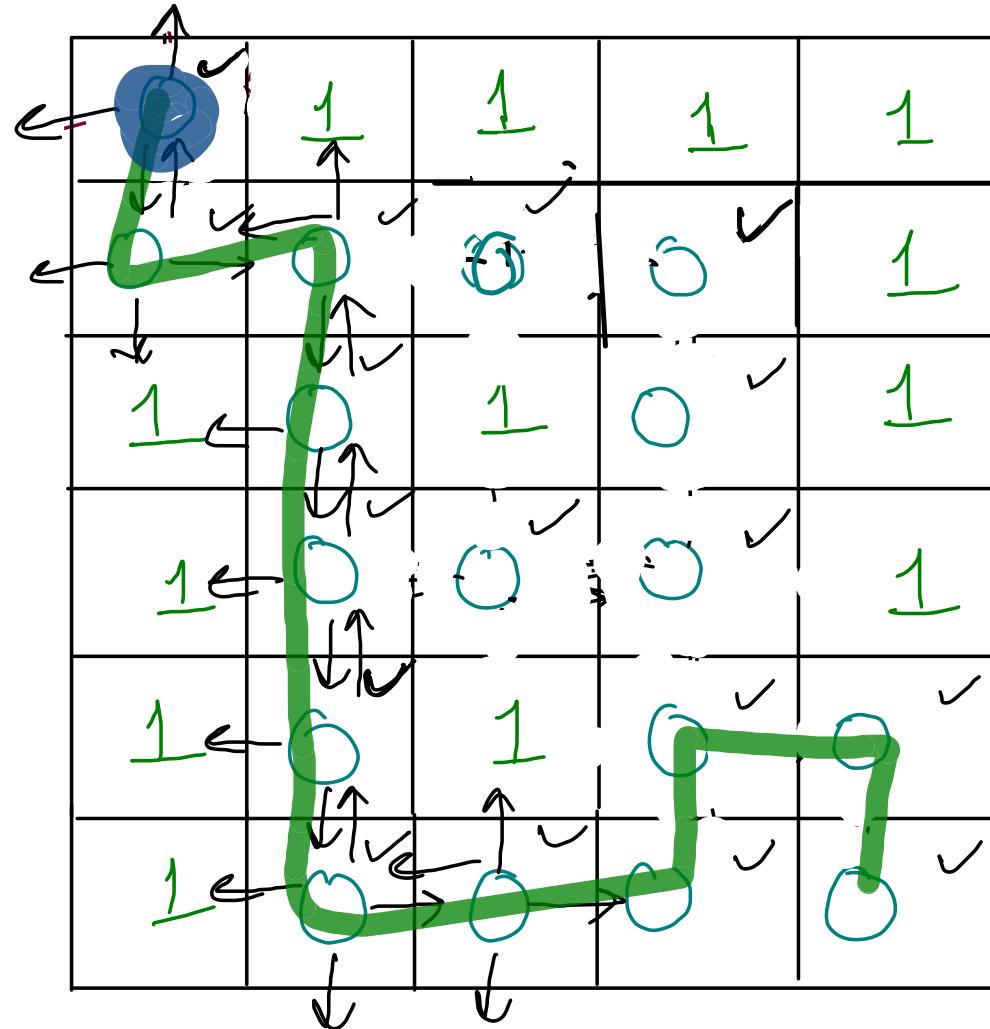
10

{ 50, 40, 30, 0 }
target = 30

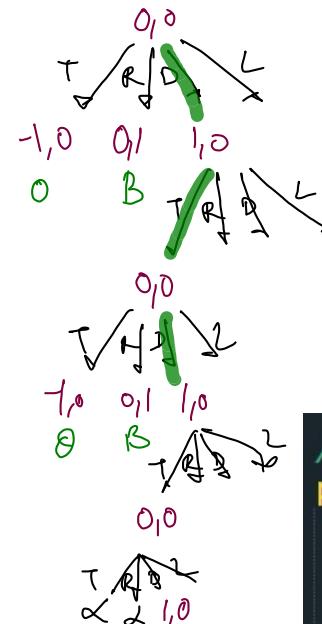
```
public static void printTargetSumSubsets(int[] arr, int idx, String set, int remTarget) {  
    if(idx == arr.length){  
        if(remTarget == 0){  
            System.out.println(set + ".");  
        }  
        return;  
    }  
  
    if(remTarget < 0){  
        // pruning step  
        return;  
    }  
  
    printTargetSumSubsets(arr, idx + 1, set + arr[idx] + ", ", remTarget - arr[idx]); // yes  
    printTargetSumSubsets(arr, idx + 1, set, remTarget); // no  
}
```



Flood fill / Rat in a Maze



$O(4^{n \times m})$



Print Single Path

```
// asf -> answer so far
public static void floodfill(int[][] maze, int sr, int sc, String psf, boolean[][] vis) {
    int dr = maze.length - 1;
    int dc = maze[0].length - 1;

    if(sr > dr || sc > dc || sr < 0 || sc < 0
        || maze[sr][sc] == 1 || vis[sr][sc] == true){
        // negative base case
        return;
    }

    if(sr == dr && sc == dc){
        // positive base case
        System.out.println(psf);
        return;
    }

    vis[sr][sc] = true;
    floodfill(maze, sr - 1, sc, psf + "t", vis); // top
    floodfill(maze, sr, sc - 1, psf + "l", vis); // left
    floodfill(maze, sr + 1, sc, psf + "d", vis); // down
    floodfill(maze, sr, sc + 1, psf + "r", vis); // right
}
```

out of bound

blockage

already visit

```

// asf -> answer so far
public static void floodfill(int[][][] maze, int sr, int sc, String psf, boolean[][] vis) {
    int dr = maze.length - 1;
    int dc = maze[0].length - 1;

    if(sr > dr || sc > dc || sr < 0 || sc < 0
        || maze[sr][sc] == 1 || vis[sr][sc] == true){
        // negative base case
        return;
    }

    ① vis[sr][sc] = true; ]> Node Pre
    if(sr == dr && sc == dc){
        // positive base case
        System.out.println(psf);
        return;
    }

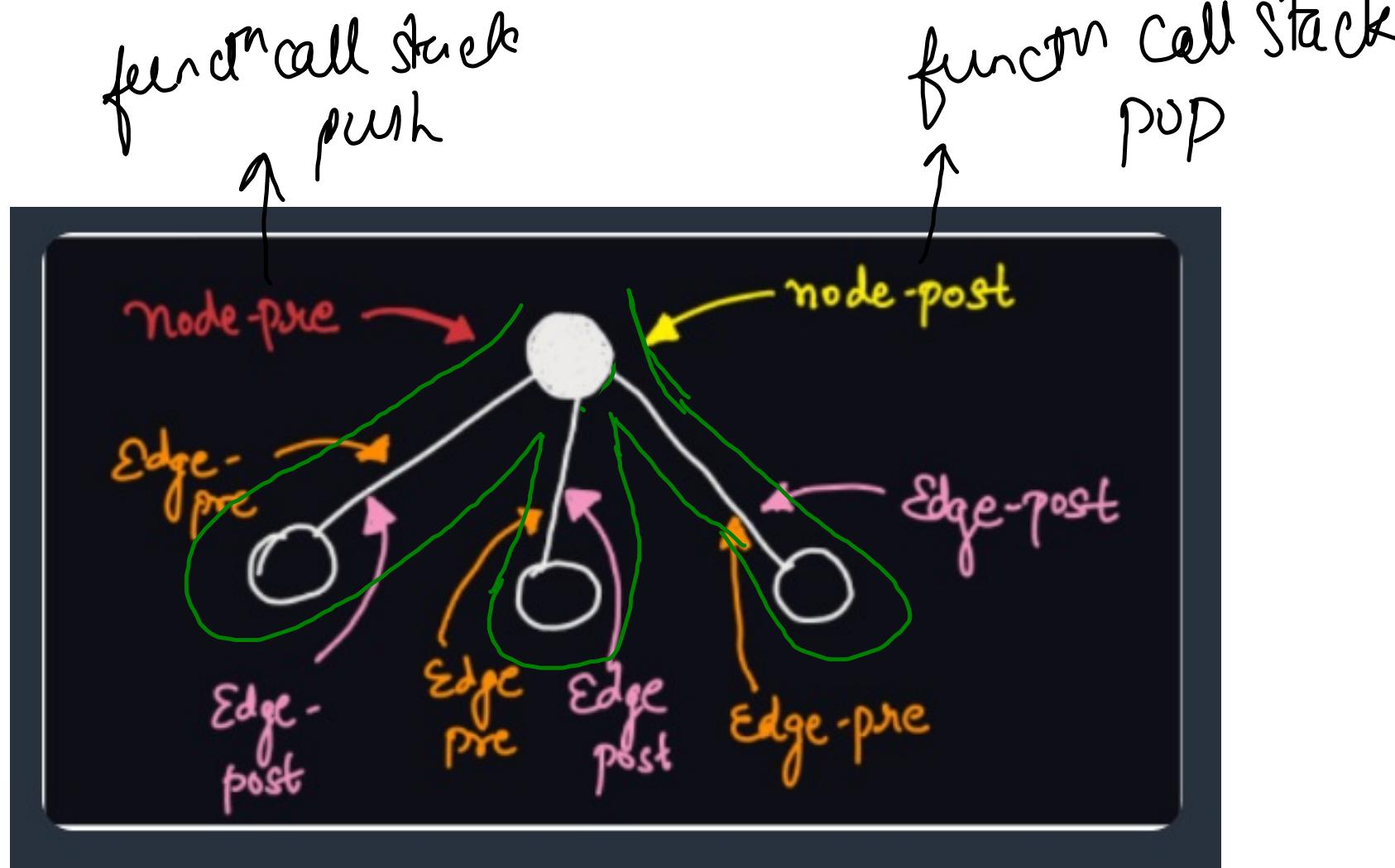
    ↗ sys0(call1 pre)
    2.1 floodfill(maze, sr - 1, sc, psf + "t", vis); // top
    2.2 floodfill(maze, sr, sc - 1, psf + "l", vis); // left
    2.3 floodfill(maze, sr + 1, sc, psf + "d", vis); // down
    2.4 floodfill(maze, sr, sc + 1, psf + "r", vis); // right

    ③ vis[sr][sc] = false; // backtracking ]> Node Post
}

```

Print All paths

Backtracking is
undoing the doings
brought by preorders
while returning
(postorder)



- R, R & B
- ① Paint All paths/
configurations/
REC
 - ② DP/Greedy
Count all paths/
config/P/C

R&B \Rightarrow Constraint

$$N \leq 20, 30$$

$2^{\geq L}$ Vec
 $2^{\geq L}$ CGO
Operation

$O(2^n)$ Exponential Time
Complexity
 \downarrow
R&B

$$N \leq 8 \log_2^{10}$$

$$N \leq 8 \times 3$$

$$N \leq 2^6$$

4 queen

q0	#	*	#
q1	#		*
q2	*		#
q3		#	*

N Queen

yellow \Rightarrow
 $Q_0(r_0, 1C)$
 $Q_1(r_1, 2C)$
 $Q_2(r_2, 3C)$
 $Q_3(r_3, 4C)$

green \Rightarrow
 $Q_0(r_0, 2C)$
 $Q_1(r_1, 0C)$
 $Q_2(r_2, 3C)$
 $Q_3(r_3, 1C)$

① Row \Rightarrow Column
 ③ Left Diag. ④ Right Diag.



level = row
 (parameter)
 base case

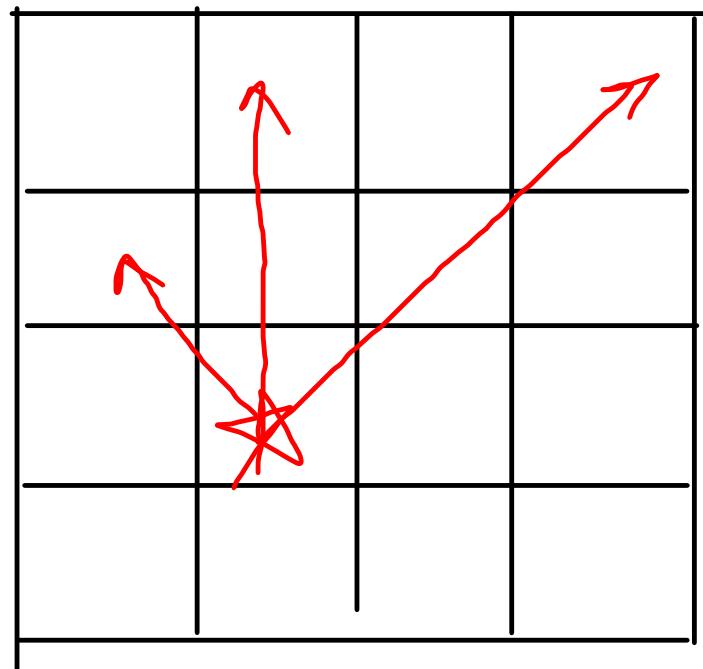
option = column
 (call)

```

public static void printNQueens(boolean[][] chess, String qsf, int row) {
    if(row == chess.length){
        System.out.println(qsf + ".");
        return;
    }

    for(int col=0; col<chess[0].length; col++){
        if(isQueenSafe(chess, row, col) == true){
            chess[row][col] = true; // edge pre
            printNQueens(chess, qsf + row + "-" + col + ", ", row + 1); // call
            chess[row][col] = false; // edge post
        }
    }
}

```



Is Queen Safe

Recursion

$$O((m^n)) + O(n) * m$$

$$O((Cn)^{n^{m^2}})$$

```

public static boolean isQueenSafe(boolean[][] chess, int row, int col){
    // upward column
    for(int i=0; i<row; i++){
        if(chess[i][col] == true){
            return false;
        }
    }

    // upward left diagonal
    int i = row, j = col;
    while(i >= 0 && j >= 0){
        if(chess[i][j] == true){
            return false;
        }
        i--; j--;
    }

    // upward right diagonal
    i = row; j = col;
    while(i >= 0 && j < chess.length){
        if(chess[i][j] == true){
            return false;
        }
        i--; j++;
    }

    return true;
}

```

Preorder

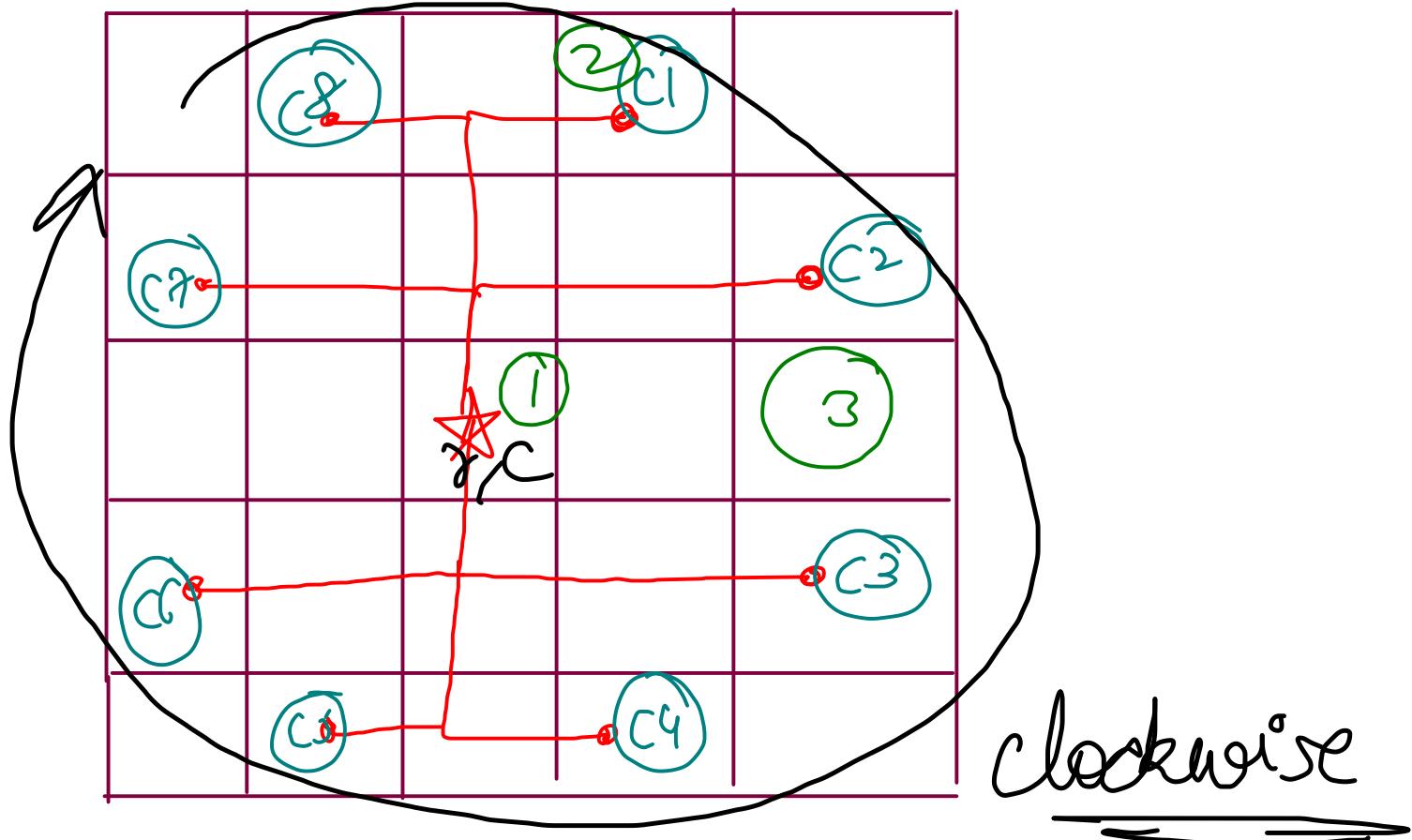
$$O(n)$$

$$O(n)$$

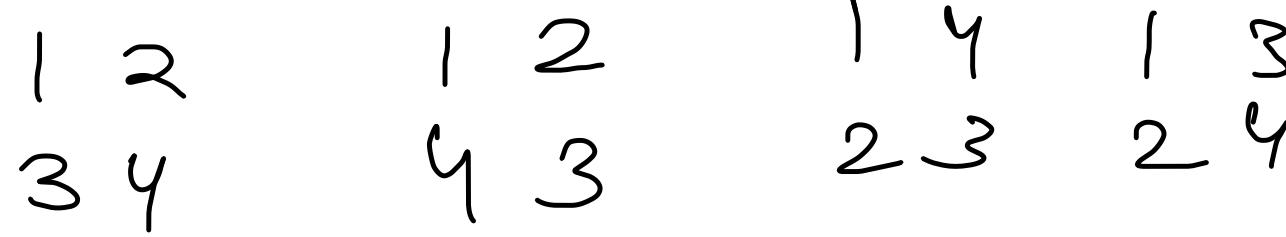
$$O(n)$$

Knight's Tour

585



King's Tour



2*2 matrix

0,0 → source node

```
public static void printKnightsTour(int[][] chess, int r, int c, int upcomingMove) {  
    if(r < 0 || c < 0 || r >= chess.length || c >= chess[0].length || chess[r][c] > 0) return;  
  
    chess[r][c] = upcomingMove;  
    if(upcomingMove == chess.length * chess.length){  
        displayBoard(chess);  
        chess[r][c] = 0;  
        return;  
    }  
  
    printKnightsTour(chess, r - 2, c + 1, upcomingMove + 1);  
    printKnightsTour(chess, r - 1, c + 2, upcomingMove + 1);  
    printKnightsTour(chess, r + 1, c + 2, upcomingMove + 1);  
    printKnightsTour(chess, r + 2, c + 1, upcomingMove + 1);  
    printKnightsTour(chess, r + 2, c - 1, upcomingMove + 1);  
    printKnightsTour(chess, r + 1, c - 2, upcomingMove + 1);  
    printKnightsTour(chess, r - 1, c - 2, upcomingMove + 1);  
    printKnightsTour(chess, r - 2, c - 1, upcomingMove + 1);  
    chess[r][c] = 0;  
}
```