Lecture - 20

Recall PMI

Similarities b/w PMI & Recursion (Introduction to Recursion)

Print Decreasing

Print Increasing

Print decreasing - increasing

factorial

Print Zig-Zag

Power

Recursion

PMI (Principle of Mathematical Induction)

PT $\sum_{i=1}^{N} i = \frac{N(N+1)}{2}$ } → Expectation

① Trivial Case (Small Problem) → Base Case

for $i = 1$

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

$RHS = \frac{1(\cancel{1+1})}{\cancel{2}} = 1$

} for $i=1$, LHS = RHS

② Assume that LHS = RHS for $i = k$ → faith

$$LHS = \sum_{i=1}^{k} i = 1+2+3+4 \dots + k$$

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

Assume that they are equal

③ for $i = k+1$ → Meeting expectation with faith.

$$LHS = \sum_{i=1}^{k+1} i = 1+2+3+4 \dots + k + (k+1)$$

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

$$RHS = \frac{N(N+1)}{2} \rightarrow \text{Put } N = k+1$$

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

LHS = RHS for $i = k+1$

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

1 → true

if it is true for k, it will be true for k+1

k k+1

1 2 3 4 . . . . N

N k+1

Recursion (Divide & Conquer)

Print Decreasing

① Expectation : printDecreasing(N) → N, N-1, N-2, N-3, ⋮, 1 for N=5: 5, 4, 3, 2, 1

② faith : pd(5) → 5, 4, 3, 2, 1 (pd(4)) pd(4) → 4, 3, 2, 1 pd(N) → N, N-1, N-2, N-3, ⋮, 1 (pd(N-1))

// We have faith that pd(N-1) will do its work.

③ Meeting Expectation with faith:

pd(N) → N ①
pd(N-1) ②

for (N=5)
5 → Print N
4, 3, 2, 1 (pd(4), pd(N-1))

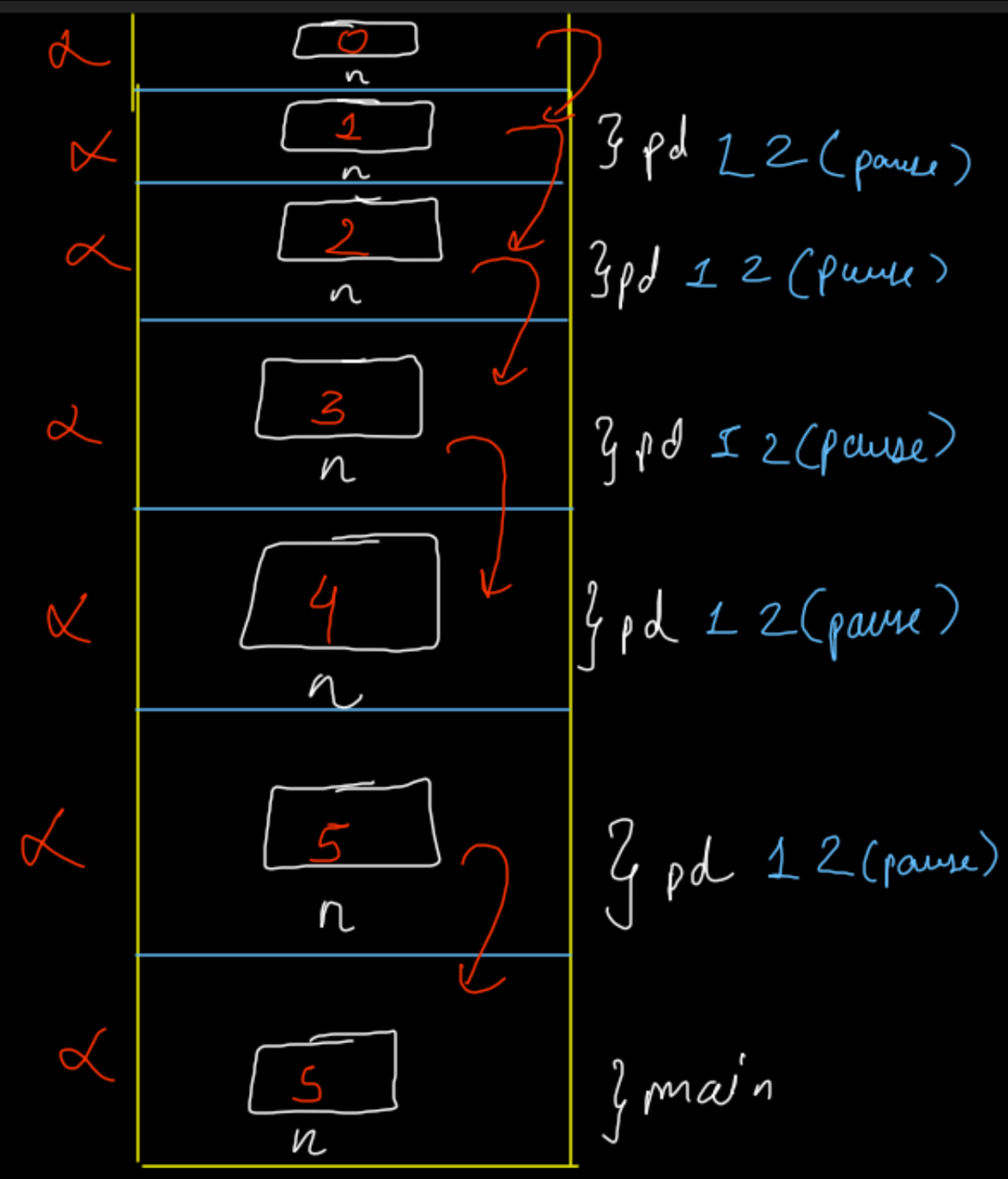
High Level Thinking

```java
public static void main(String[] args) throws Exception {
    // write your code here
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();

    printDecreasing(n);
}

public static void printDecreasing(int n){

    System.out.println(n);
    printDecreasing(n-1);
}

}
```

5
4
3
2
1

if (n == 0) → Base Case
return;

Low-level thinking



```java
import java.io.*;
import java.util.*;

public class Main {

    public static void main(String[] args) throws Exception {
        // write your code here
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();

        printDecreasing(n);
    }

    public static void printDecreasing(int n){

        if(n == 0) return; //base case

        System.out.println(n);
        printDecreasing(n-1); //recursive call
    }

}
```

} → Tail Recursion

(when recursive call is the last statement of a recursive function)

Print 1 To N Without Loop

School Accuracy: **51.9%** Submissions: **74255** Points: **0**

Print numbers from **1 to N** without the help of loops.

Example 1:

Input:
N = 10
Output: 1 2 3 4 5 6 7 8 9 10

① Expectation : pi(N) → 1, 2, 3, 4, ⋮, N

pi(5) → 1, 2, 3, 4, 5

② faith : pi(N-1) → 1, 2, 3, 4, ⋮, N-1

pi(4) → 1, 2, 3, 4

③ Meeting expectation with faith

```
pi(n-1);
Print(n);
```

pi(4) → 1, 2, 3, 4

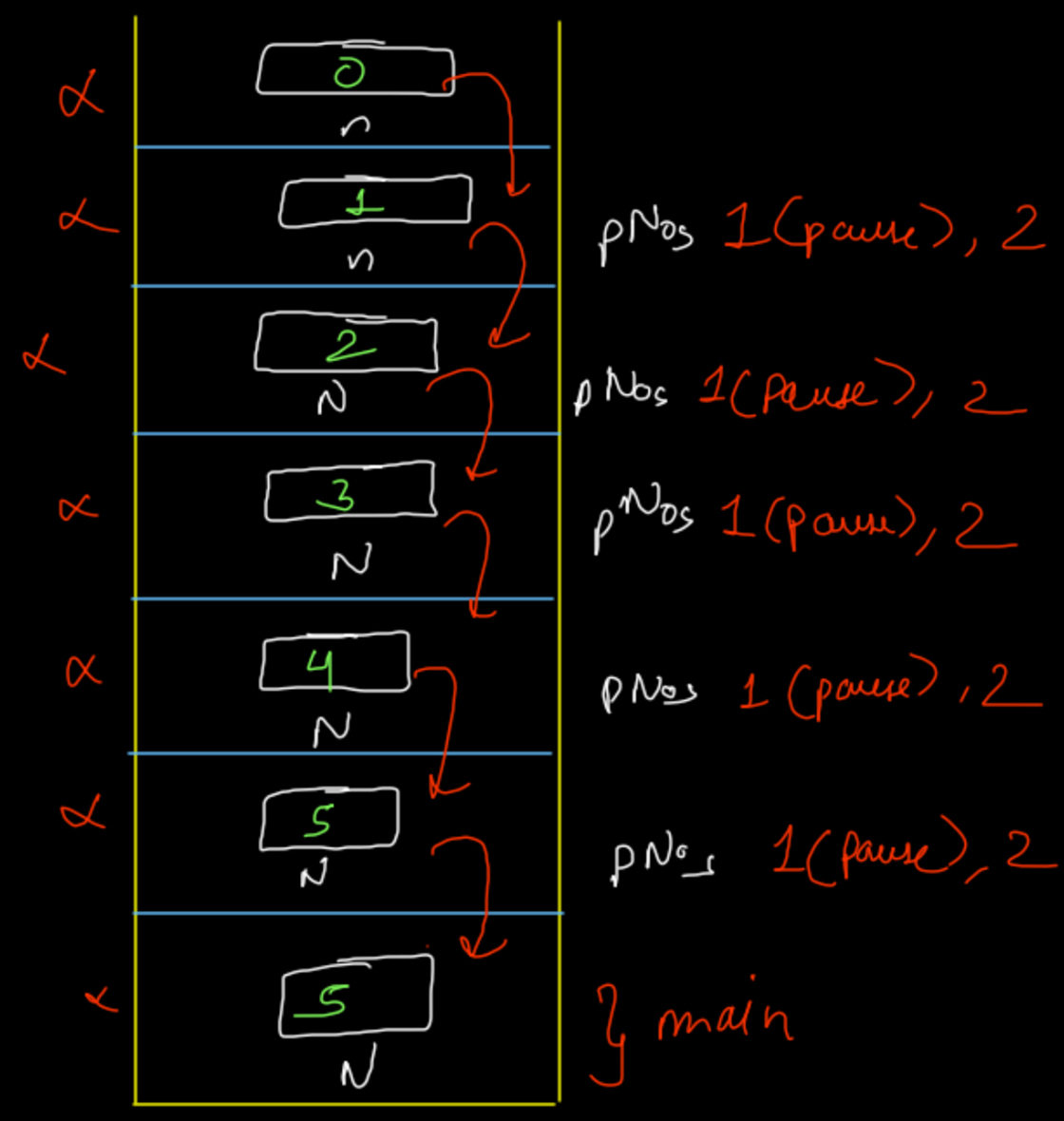
print(5) → 5

```java
class Solution
{

  public void printNos(int N
  {
      //Your code here

      printNos(N-1);
      System.out.println(N);
  }
}
```

if (n==0) return;

Base Case

1
2
3
4
5



```java
class Solution
{

  public void printNos(int N)
  {
      //Your code here

      if(N == 0) return;

      printNos(N-1);
      System.out.print(N + " ");
  }
}
```

Head Recursion

when recursion call is the first statement of a recursive function

Print Increasing Decreasing

Easy

Prev Next

1. You are given a positive number n.
2. You are required to print the counting from n to 1 and back to n again.
3. You are required to not use any loops. Complete the body of pdi function to achieve it. Don't change the signature of the function.

Note -> The online judge can't force you to write the function recursively but that is what the spirit of question is.Write recursive and not iterative logic. The purpose of the question is to aid learning recursion and not test you.

N = 3

3
2
1
1
2
3

① Expectation : pdi(N)

N
N-1
⋮
1
1
2
⋮
N

pdi(3) → 3
2
1
1
2
3

② faith : pdi(N-1) → N-1
N-2
⋮
1
1
2
⋮
N-1

pdi(2) → 2
1
1
2

③ Meeting expectation with faith:

Print(N)

pdi(N-1);

Print(N).

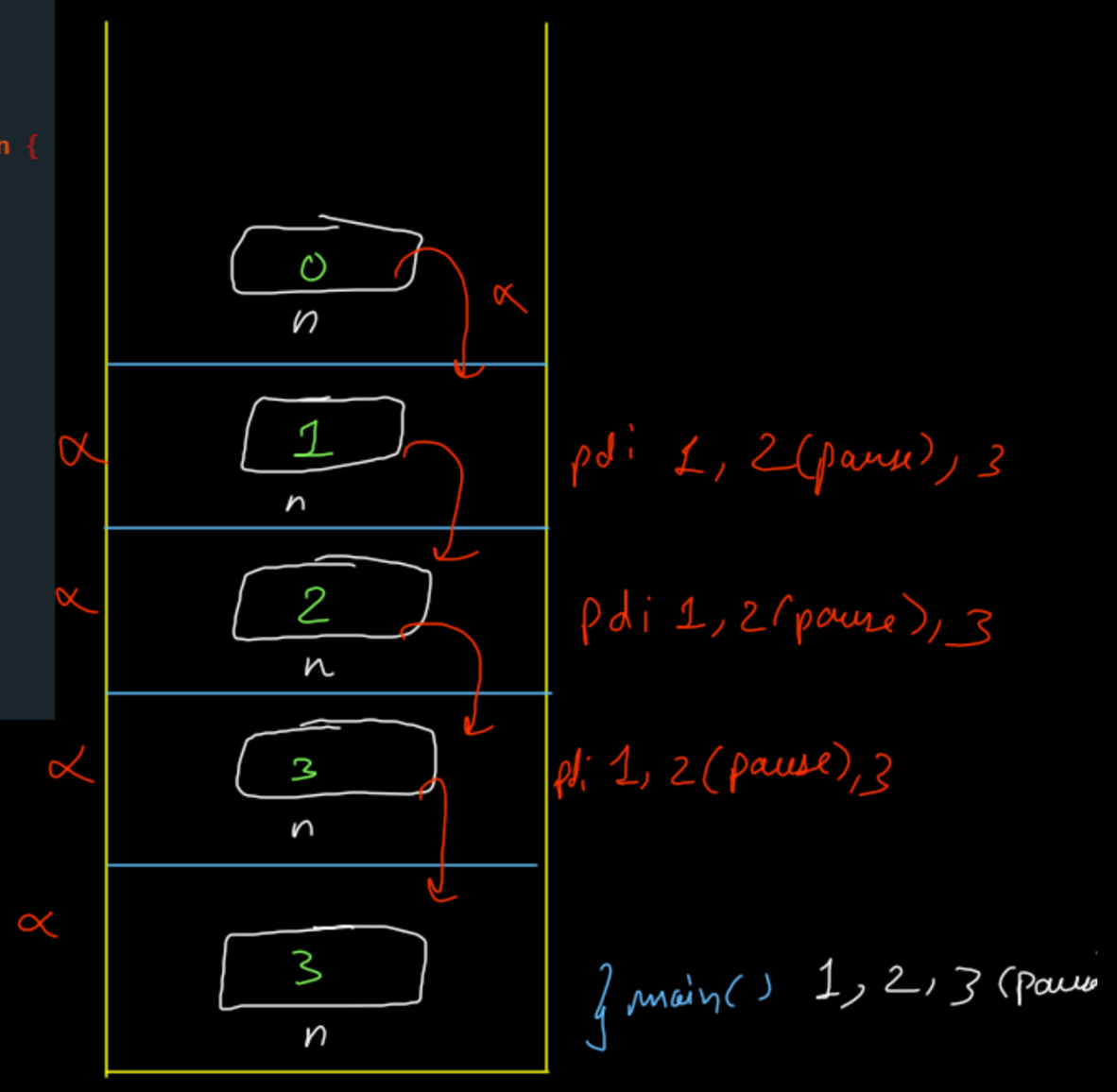
pdi(3) → Print(3)

pdi(2) → 2
1
1
2

Print(3)

```java
import java.io.*;
import java.util.*;

public class Main {

    public static void main(String[] args) throws Exception {
        // write your code here
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();
        pdi(n);
    }

    public static void pdi(int n){
        if (n == 0) return;
        System.out.println(n); 1

        //recursion call
        pdi(n-1); 2

        System.out.println(n); 3
    }

}
```

3
2
1
1
2
3



```java
import java.io.*;
import java.util.*;

public class Main {

    public static void main(String[] args) throws Exception {
        // write your code here
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();
        pdi(n);
    }

    public static void pdi(int n){

        if(n == 0) return;

        System.out.println(n);

        //recursion call
        pdi(n-1);

        System.out.println(n);
    }

}
```

Factorial

Basic Accuracy: **53.69%** Submissions: **35842** Points: **1**

Given a positive integer, **N**. Find the factorial of **N**.

$5! = 1 \times 2 \times 3 \times 4 \times 5$

or

$5 \times 4 \times 3 \times 2 \times 1$

$N! = N * (N-1) * (N-2) \ldots\ldots 1$

① Expectation : factorial (N) → N * (N-1) * (N-2) . . . 1

② faith : factorial (N-1) → (N-1) * (N-2) * (N-3) 1

③ Meeting expectation with faith : N * factorial (N-1)

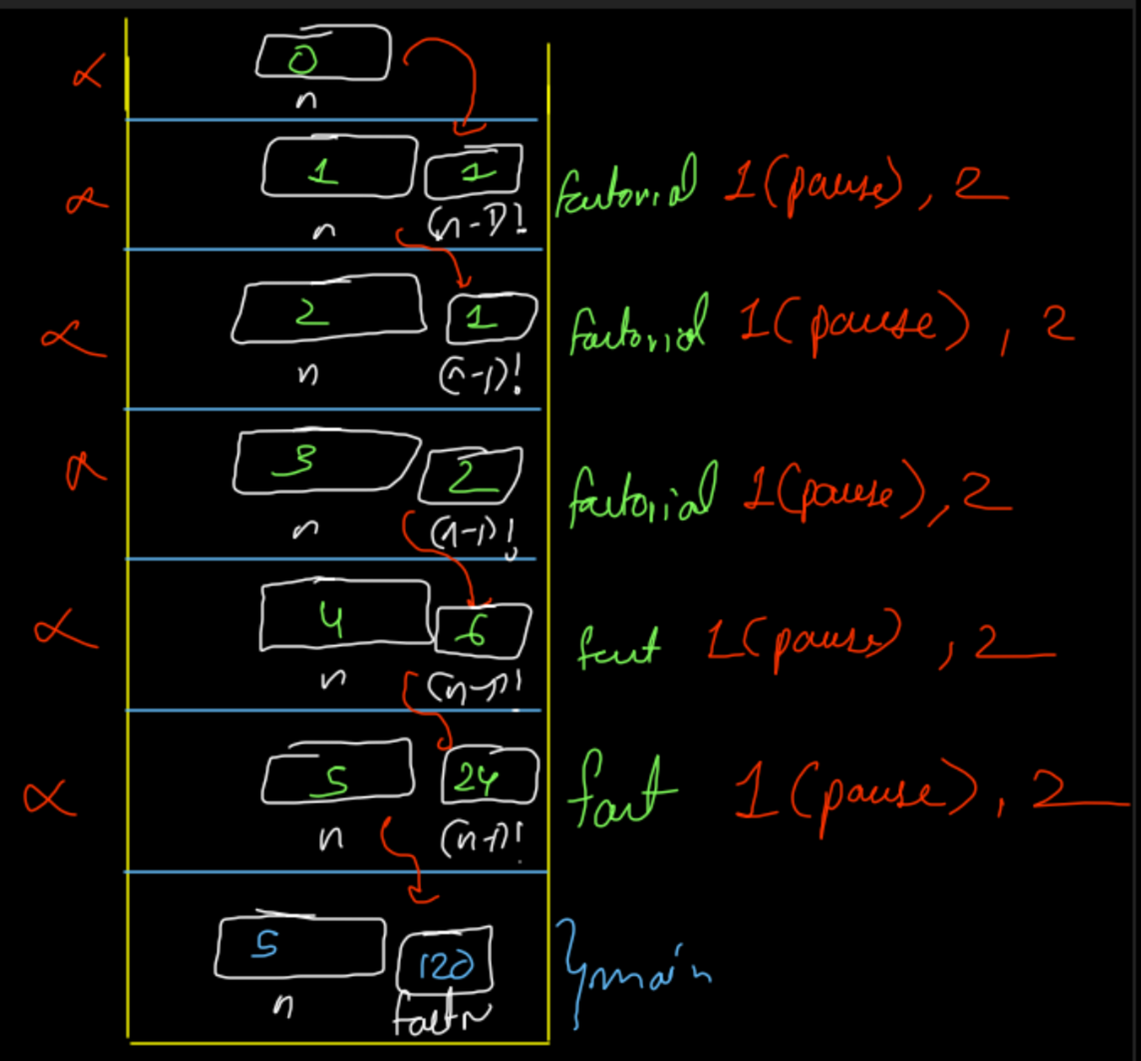
```java
class Solution{
    static long factorial(int N){
        long nm1Factorial = factorial(N-1);
        return N * nm1Factorial;
    }
}
```

(1)

1

2

if (n==0) return 1;



```java
class Solution{
    static long factorial(int N){
        // code here
        if(N == 0) return 1L;

        long nm1Factorial = factorial(N-1);
        return N * nm1Factorial;
    }
}
```

Print Zigzag

Easy

Prev Next

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

Input2 -> 3
Output3 -> 3 2 1 1 1 2 1 1 1 2 3 2 1 1 1 2 1 1 1 2 3

2. Figure out the pattern and complete the recursive function pzz to achieve the above for any positive number n.

Note -> The online judge can't force you to write the function recursively but that is what the spirit of question is.Write recursive and not iterative logic. The purpose of the question is to aid learning recursion and not test you.

Expectation

```
3211121112321112111 23
```

→ for N = 3

faith :

```
211121112
```

→ for N = 2

Meeting Expec : with faith

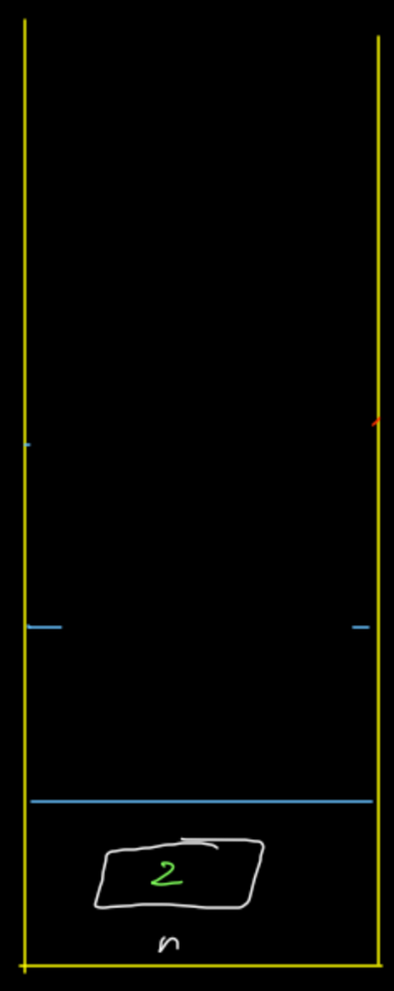
```
3211121112321112111 23
```

① Print N
② pzz(N-1)
③ Print(N)
④ pzz(N-1)
⑤ Print(N)

```java
import java.io.*;
import java.util.*;

public class Main {

    public static void main(String[] args) throws Exception {
        // write your code here
    }

    public static void pzz(int n){

        System.out.print(n + " ");  1
        pzz(n-1);  2
        System.out.print(n + " ");  3
        pzz(n-1);  4
        System.out.print(n + " ");  5
    }

}
```

2
1
1
1
2
2
1

1
1
2



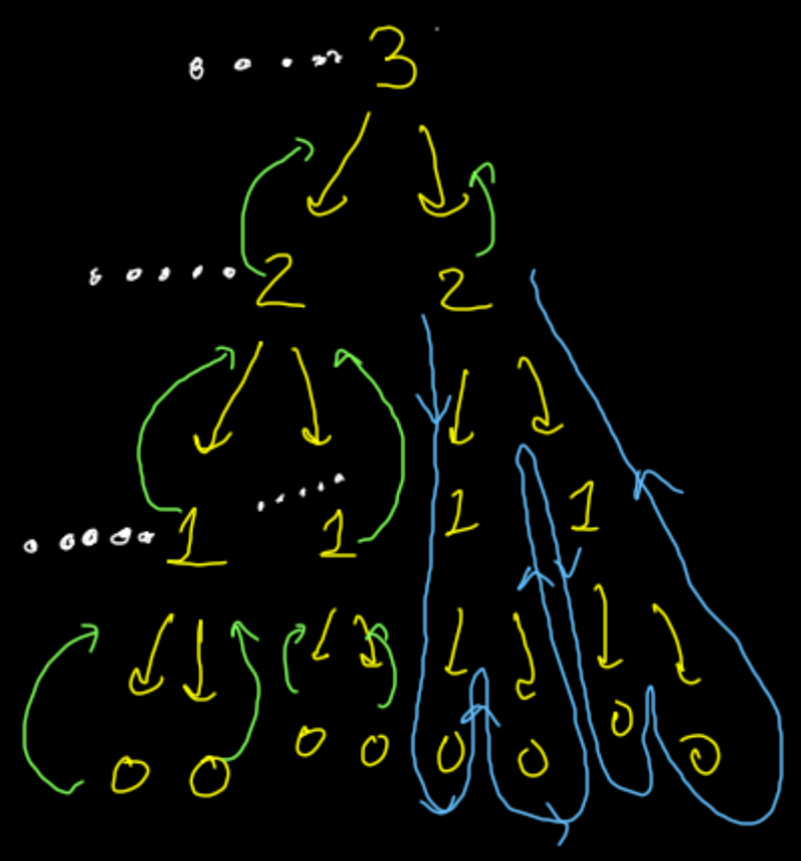
pzz 1, 2(p), 3, 4, 5

& main

Recursion Tree (Euler Tree)

3 2 1 1 1 2 1 1 1 2 3 [2 1 1 1 2 1 1 1 2] 3

```java
import java.io.*;
import java.util.*;

public class Main {

    public static void main(String[] args) throws Exception {
        // write your code here
    }

    public static void pzz(int n){

        System.out.print(n + " ");  1
        pzz(n-1);  2
        System.out.print(n + " ");  3
        pzz(n-1);  4
        System.out.print(n + " ");  5
    }

}
```



```java
import java.io.*;
import java.util.*;

public class Main {

    public static void main(String[] args) throws Exception {
        // write your code here
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();
        pzz(n);
    }

    public static void pzz(int n){

        if(n == 0) return;

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

}
```

factorial (Recursion Tree)

5! (120)

↓

(24) 4! * 5

↓

(6) 3! * 4

↓

(2) 2! * 3

↓

(1) 1! * 2

↓

0! * 1

(1)

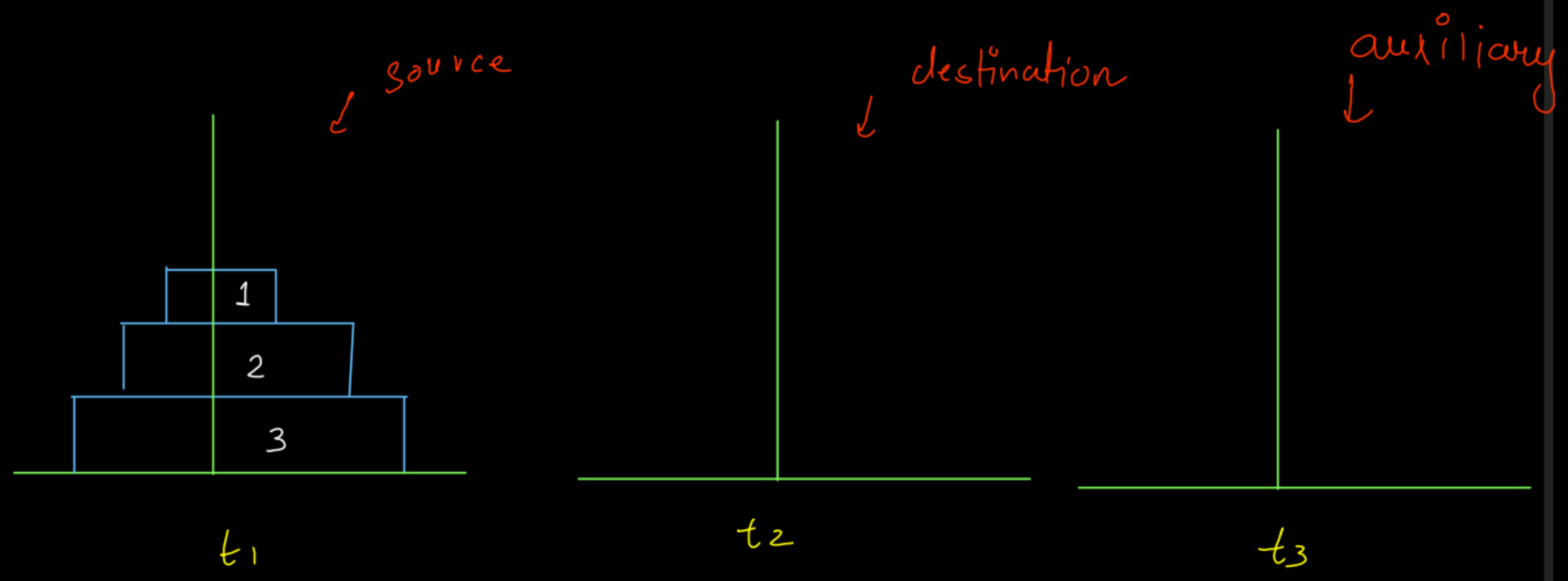
Lecture - 21

Tower of Hanoi Problem

- Understanding the Problem Statement
- High Level Thinking
- Program (With Base case)
- Low level thinking → Using Call Stack
- Low level thinking → Using Euler Tree
- Conclusion

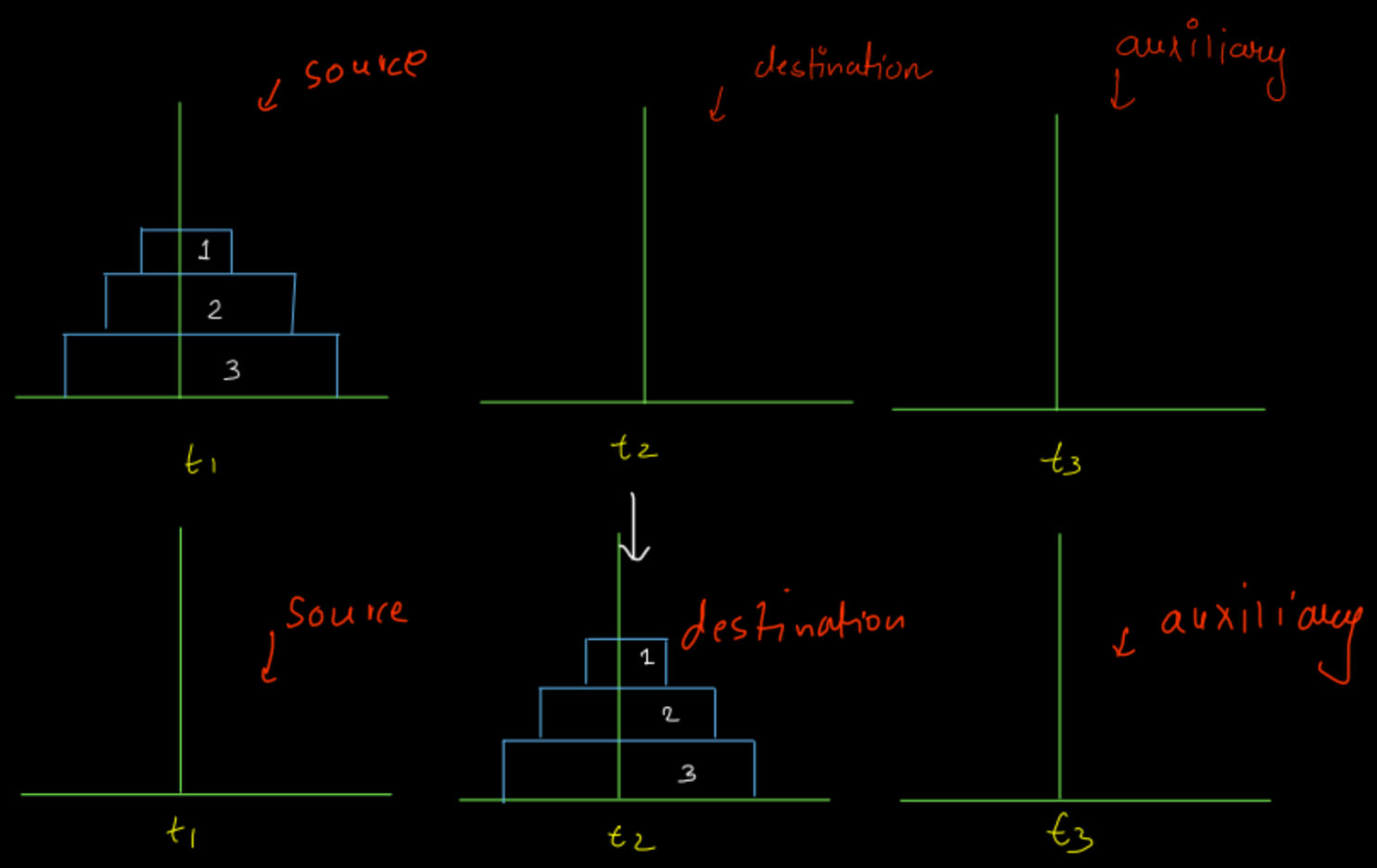
Toh Problem

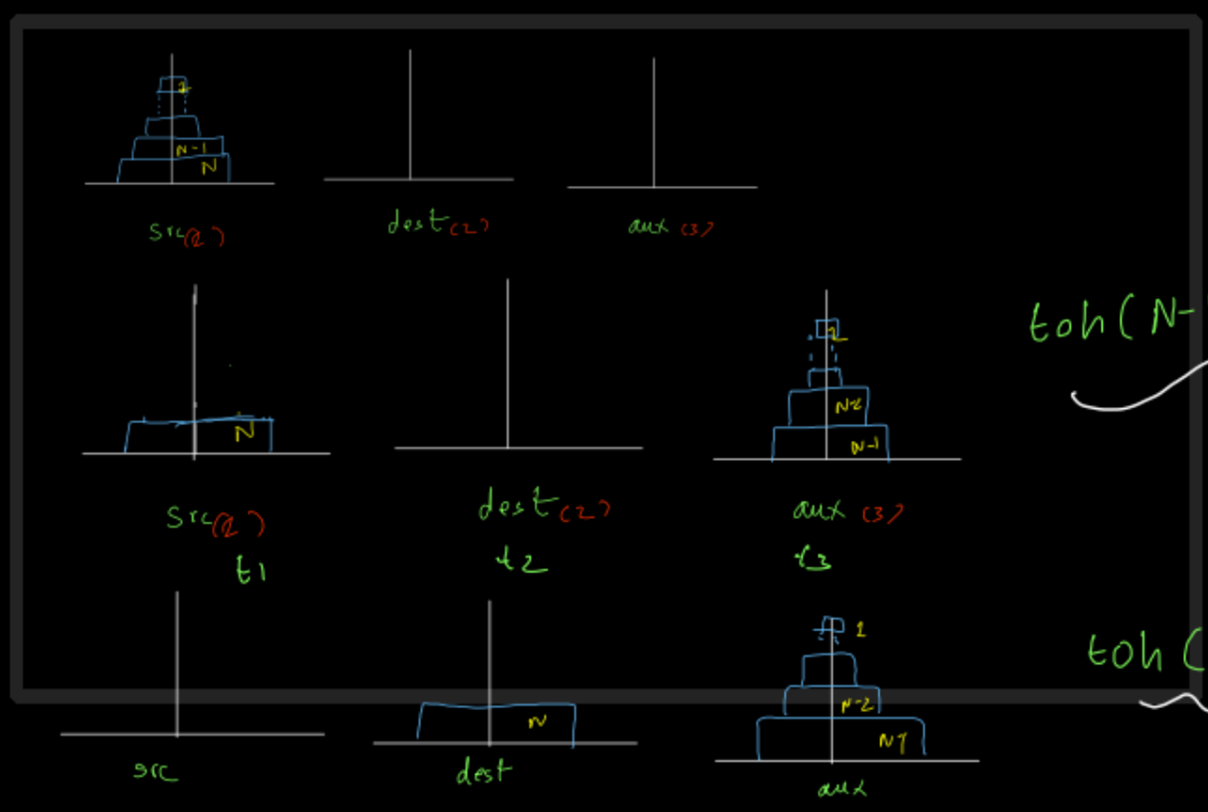
3.3. following the rules
- 3.3.1 move 1 disk at a time.
- 3.3.2 never place a smaller disk under a larger disk.
- 3.3.3 you can only move a disk at the top.



Expectation (for N=3) toh(N, t_1, t_2, t_3)

** following the rules





src dest aux

$toh(N-1, t_1, t_3, t_2)$ {faith 1}

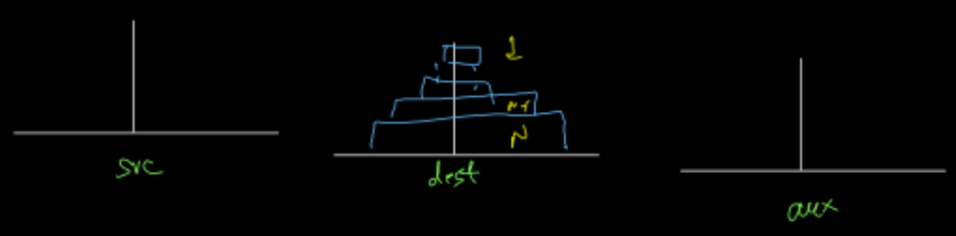
$x \rightarrow$ no of moves

src dest aux

$toh(1, t_1, t_2, t_3)$ 1

$x + y + 1$

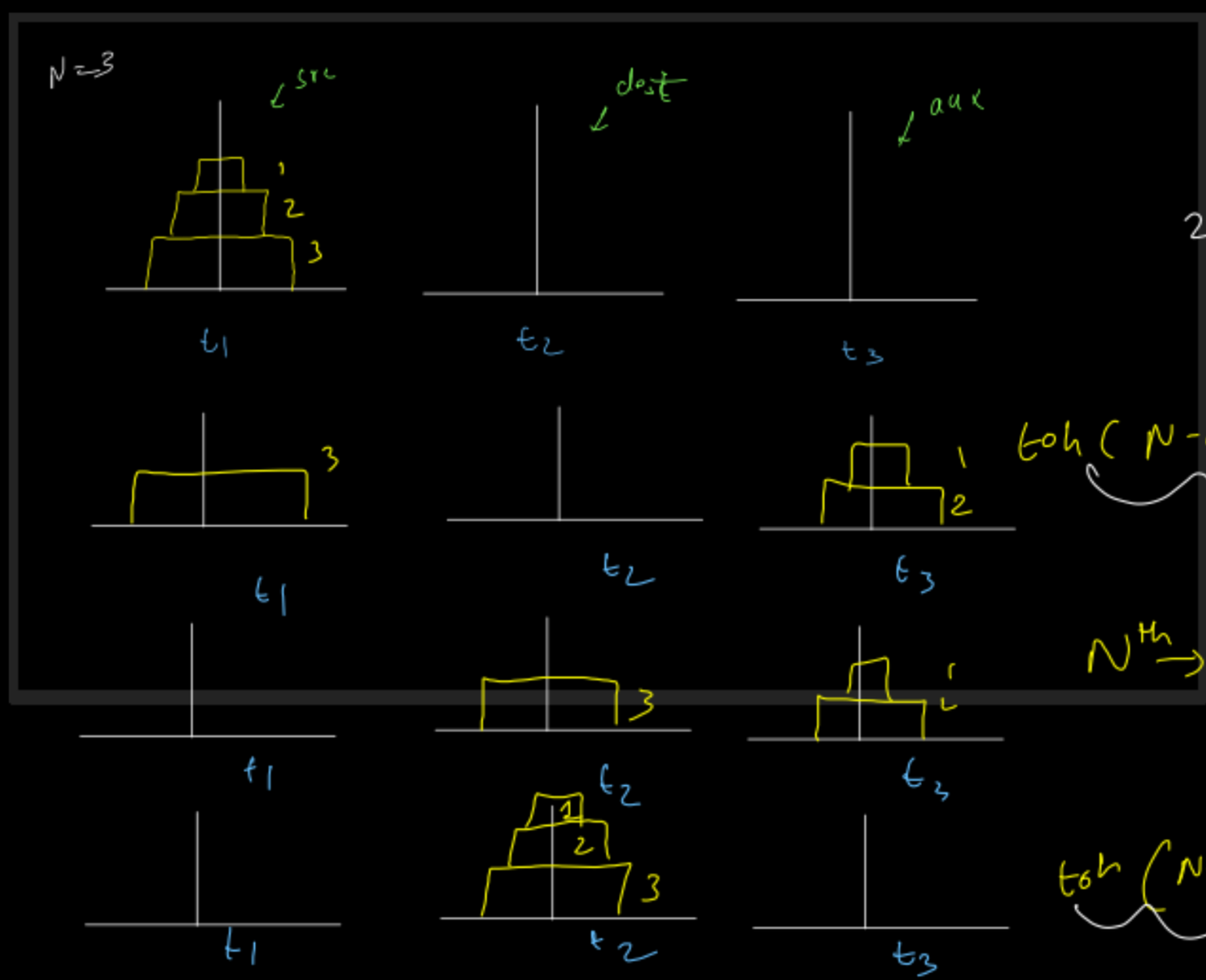
& $x = y$



src dest aux

$toh(N-1, t_3, t_2, t_1)$ {faith 2}

$y \rightarrow$ no of moves.



$2 * x + 1$

src dest aux

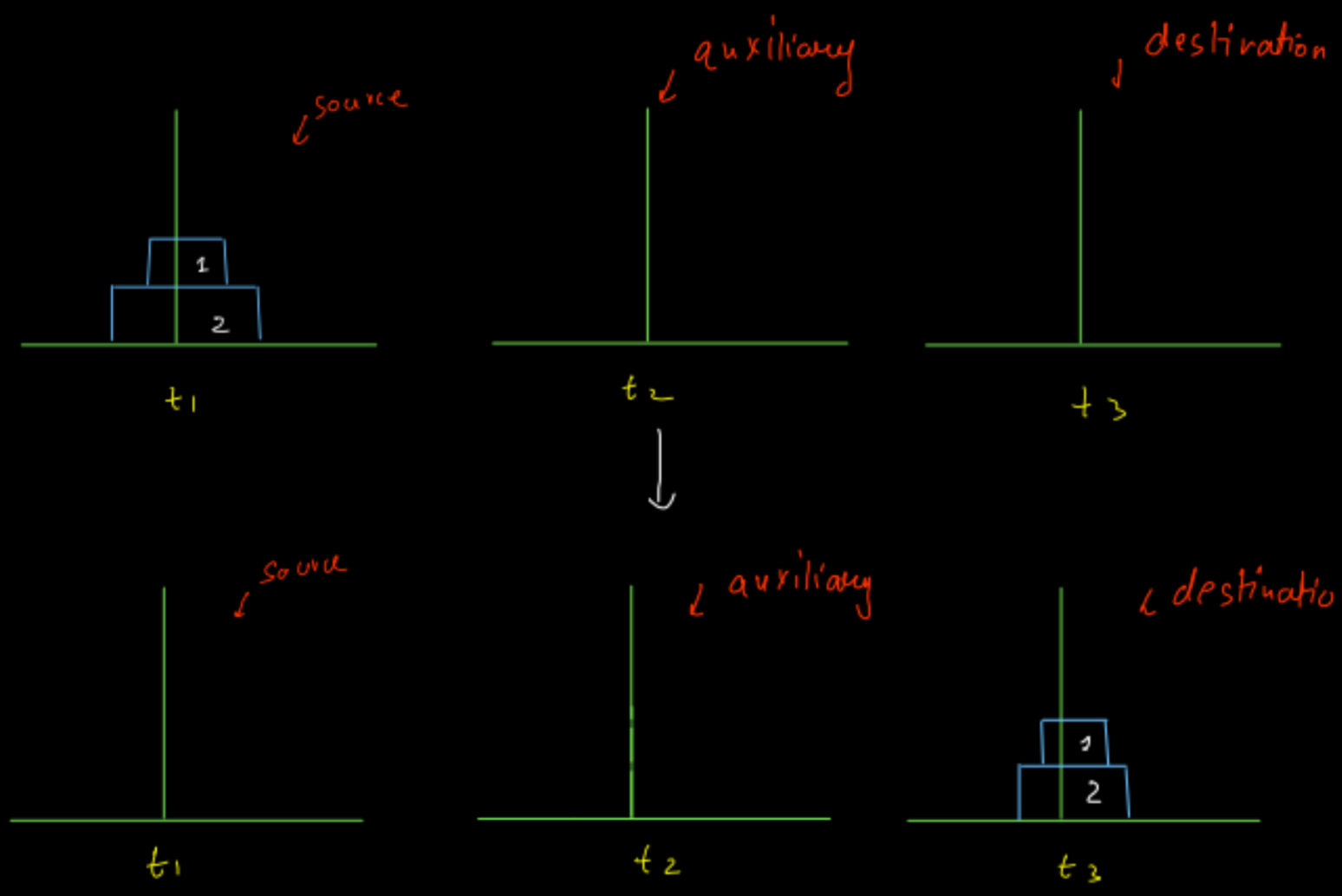
$\underbrace{toh(N-1, t_1, t_3, t_2)}_{x}$

$N^{th} \rightarrow t_1$ to t_2

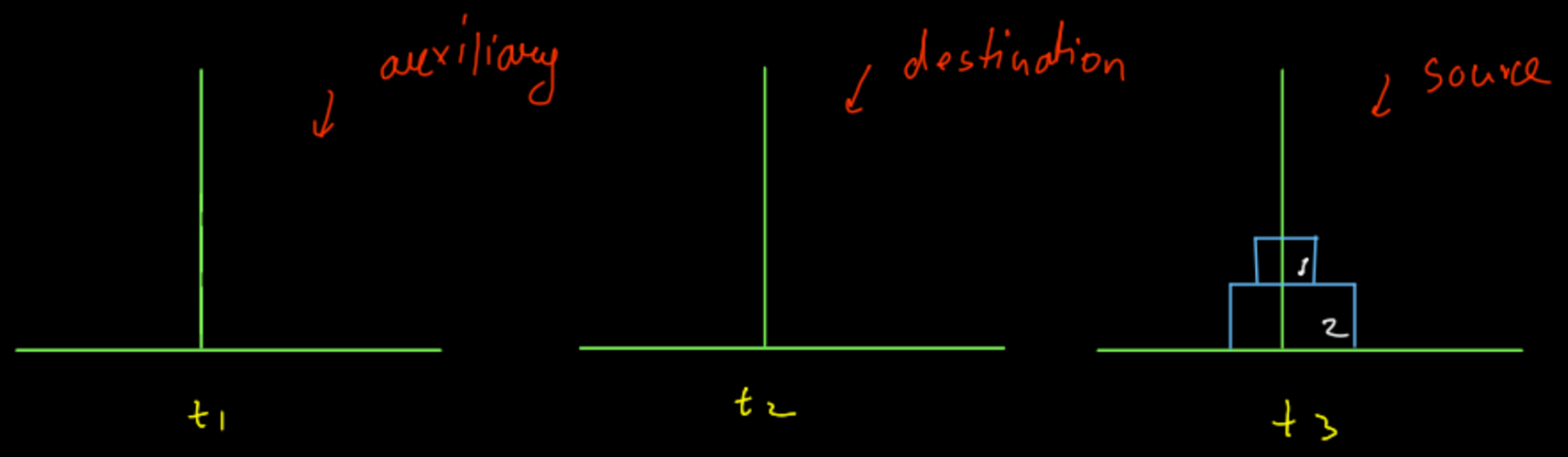
src dest aux

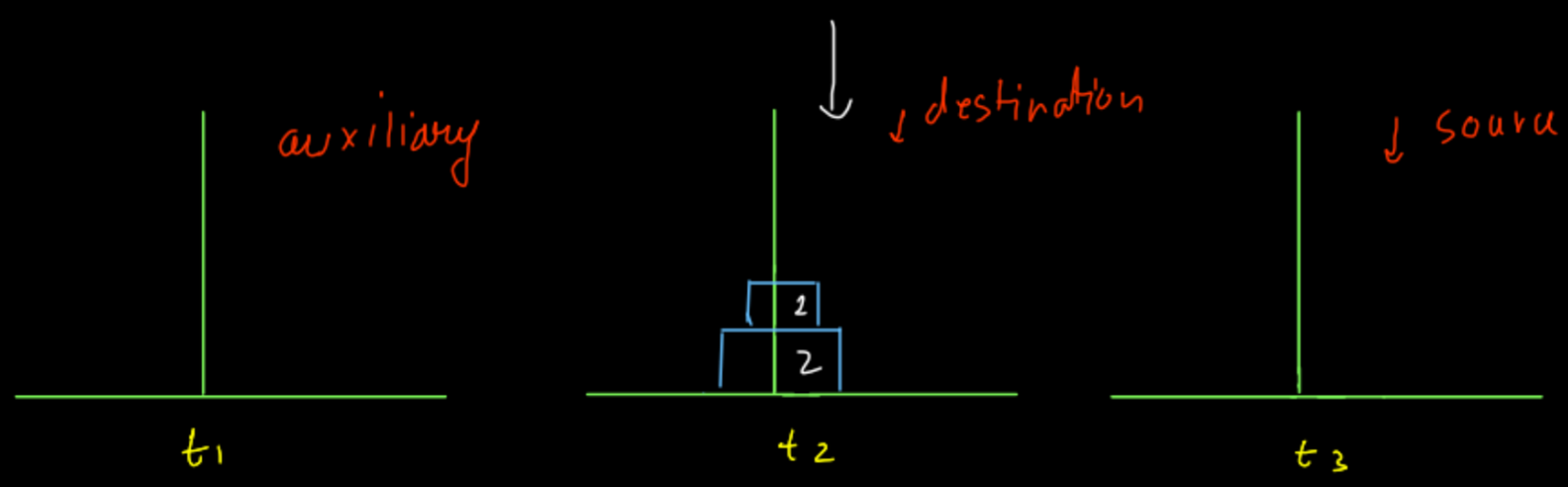
$\underbrace{toh(N-1, t_3, t_2, t_1)}_{x}$

faith1 (N = 2) toh(N-1, t1, t3, t2)



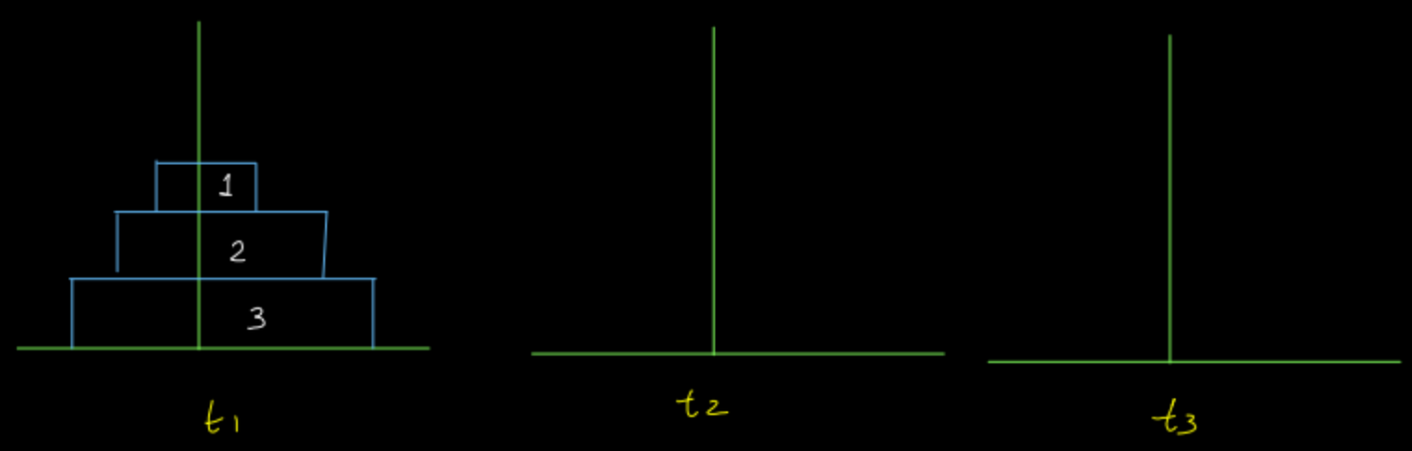
faith 2 (N = 2) toh (N-1 , t3 , t2 , t1)

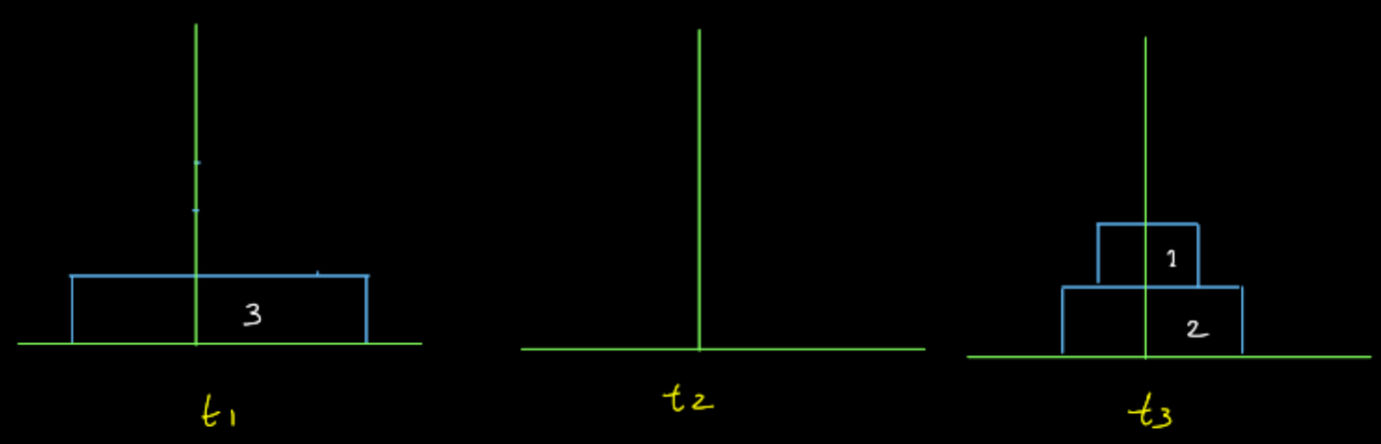




Meeting Expectation with faith

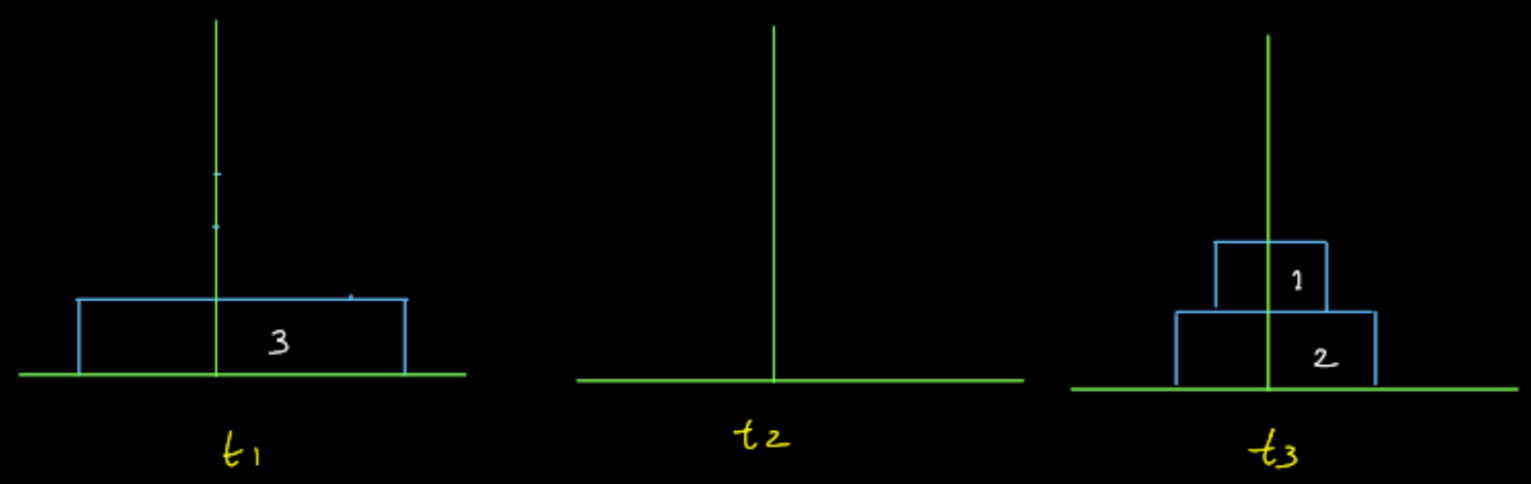
① Move (N-1) disks from t_1 to t_3 using t_2. (faith 1)

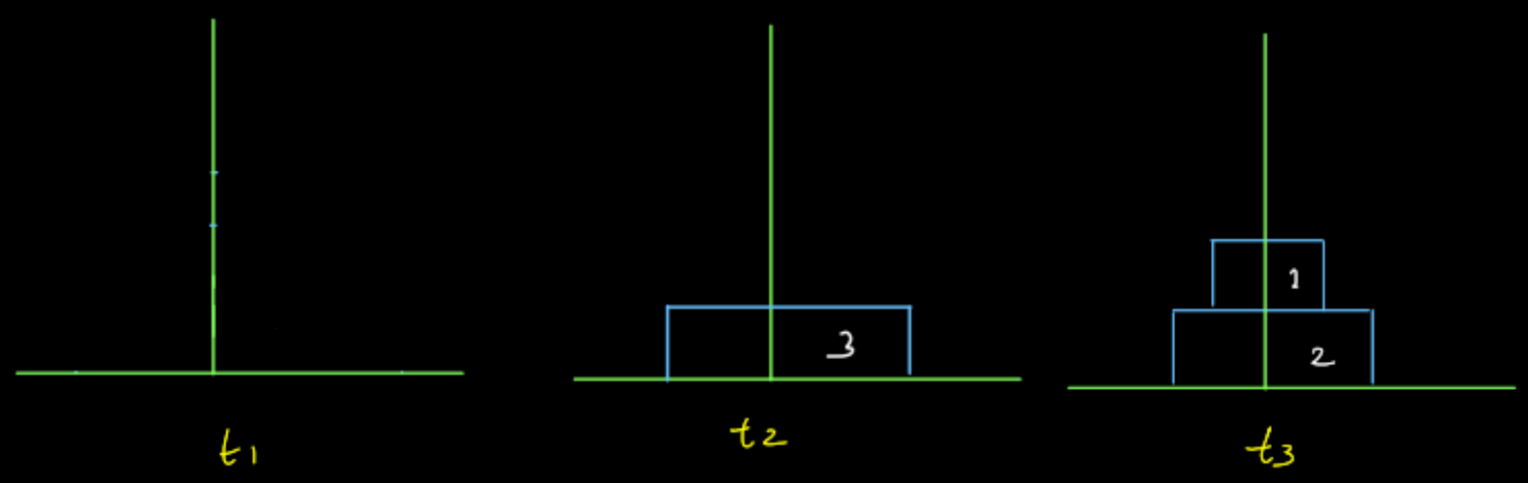




Meeting Expectation with faith

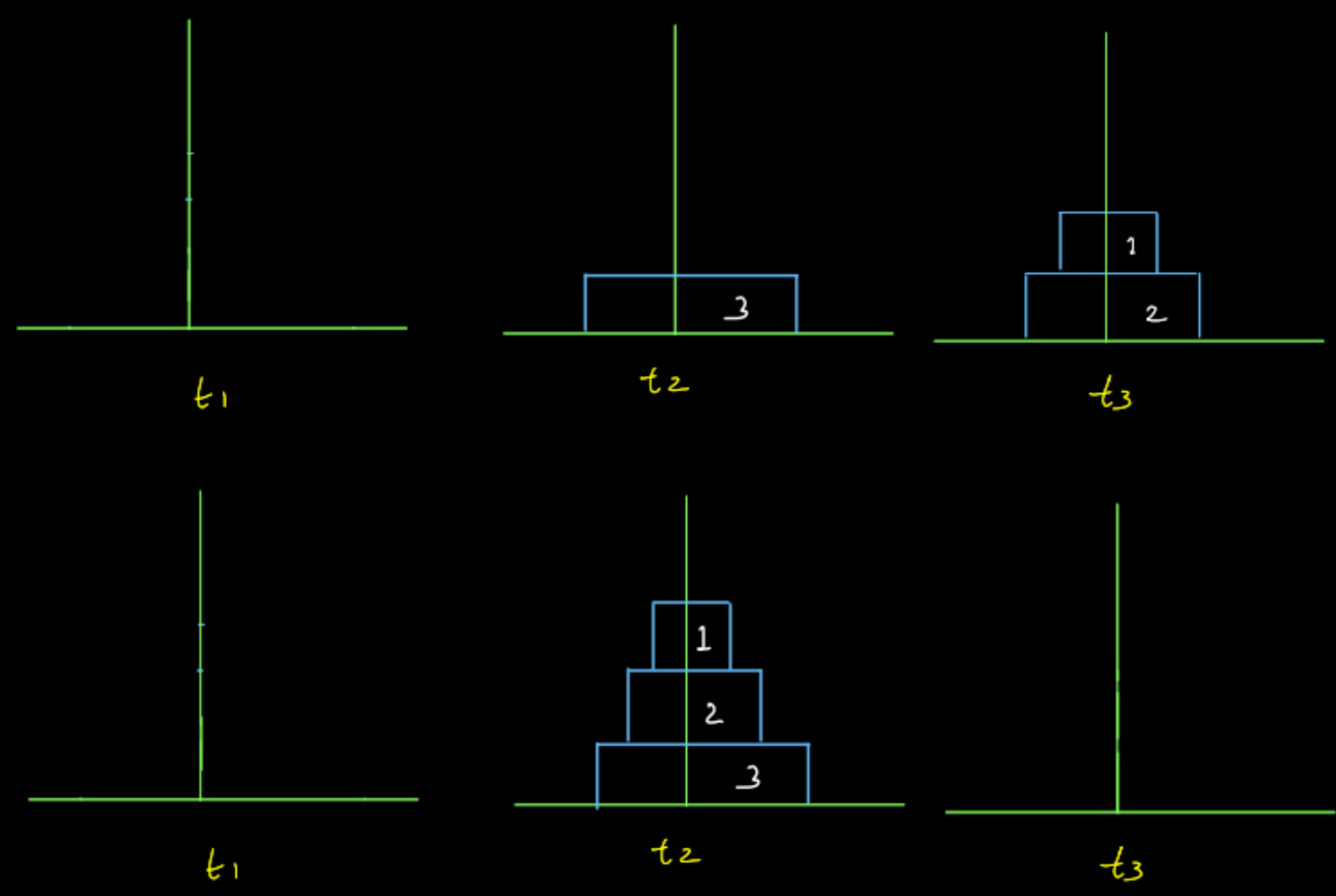
② Move N^{th} disk from t_1 to t_2.





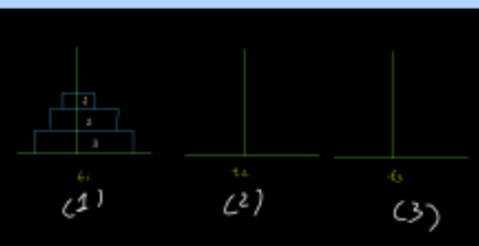
Meeting Expectation with faith

③ Move (N-1) disks from t3 to t2 using t1. (faith 2)

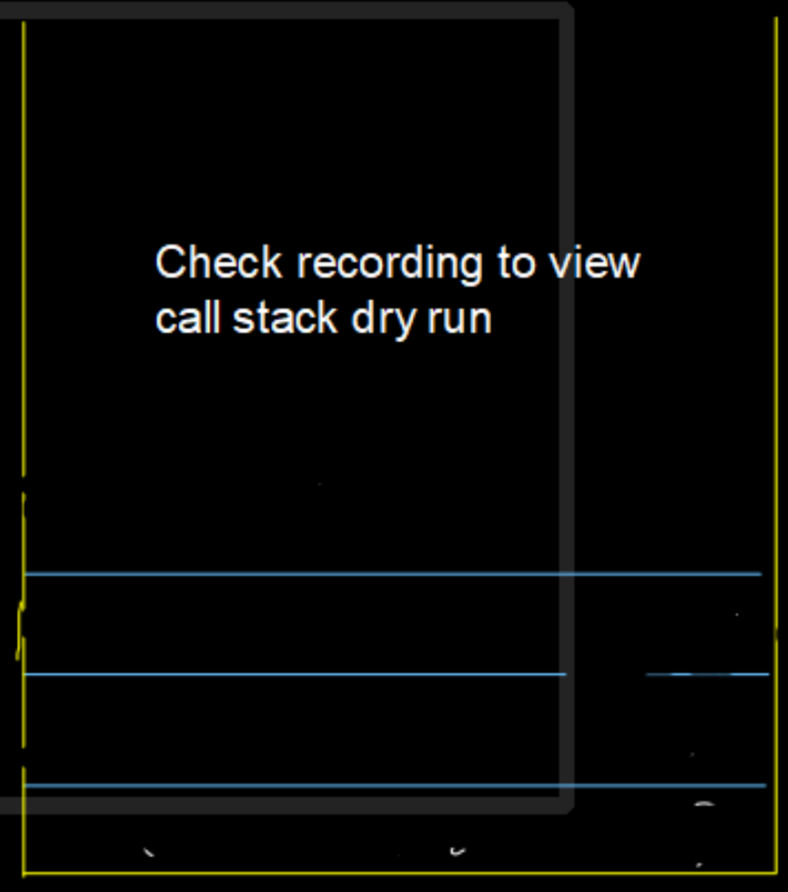


N = 3

```
// t1 -> src t2-> dest t3-> aux
public long toh(int N, int t1, int t2, int t3) {
    // Your code here

    if(N == 0) return 0;

    long ans1 = toh(N-1,t1,t3,t2);  1
    System.out.println("move disk " + N + " from rod " + t1 + " to rod " + t2);  2
    long ans2 = toh(N-1,t3,t2,t1);  3

    return ans1 + ans2 + 1;  4
}
```



Check recording to view call stack dry run



1, 2, 3, 1

① from 1 to 2
② from 1 to 3
① from 2 to 3
③ from 1 to 2

① from 3 to 1
② from 3 to 2
① from 1 to 2

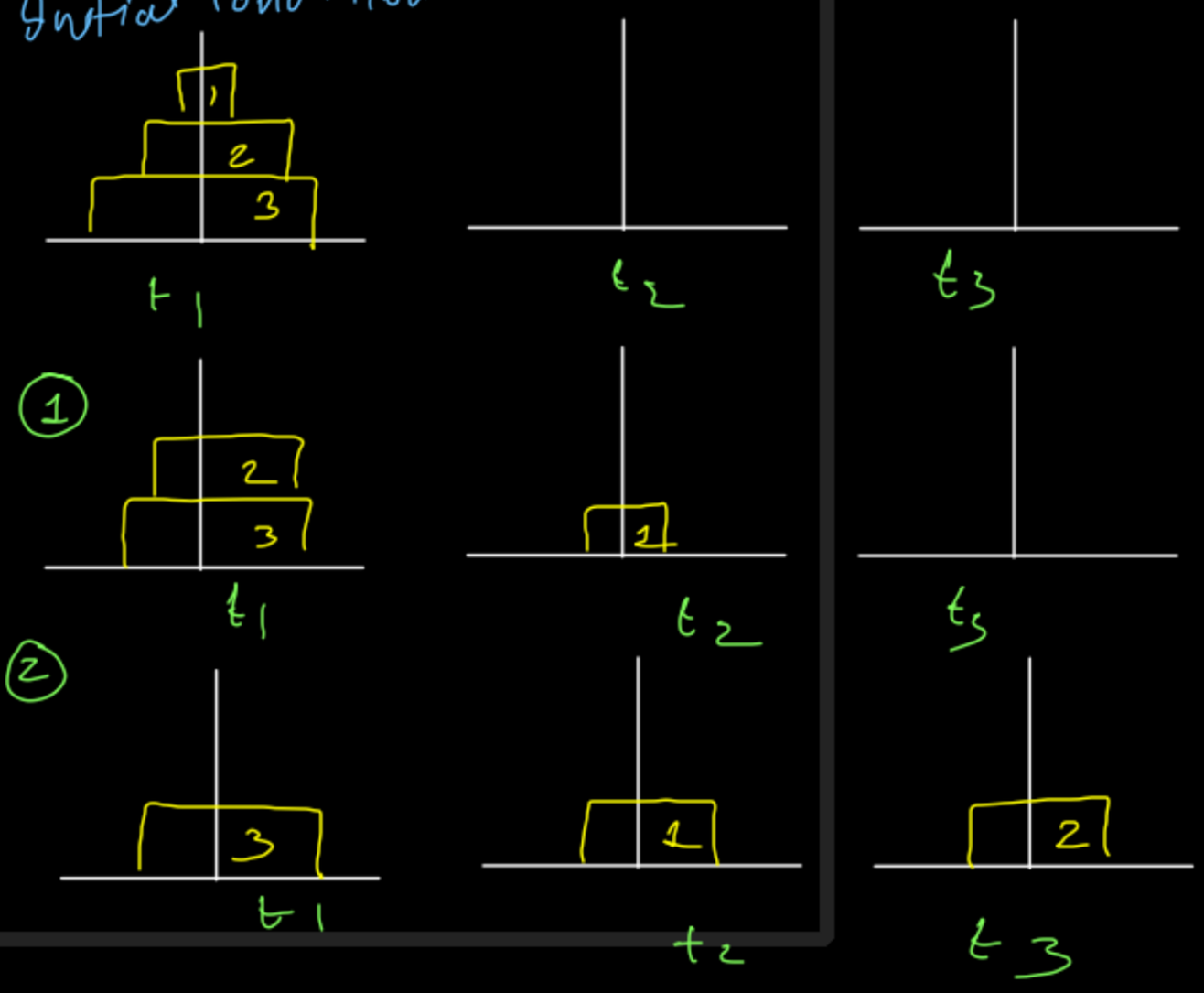
ans 1 = 3

ans 2 = 3

+1 = 7

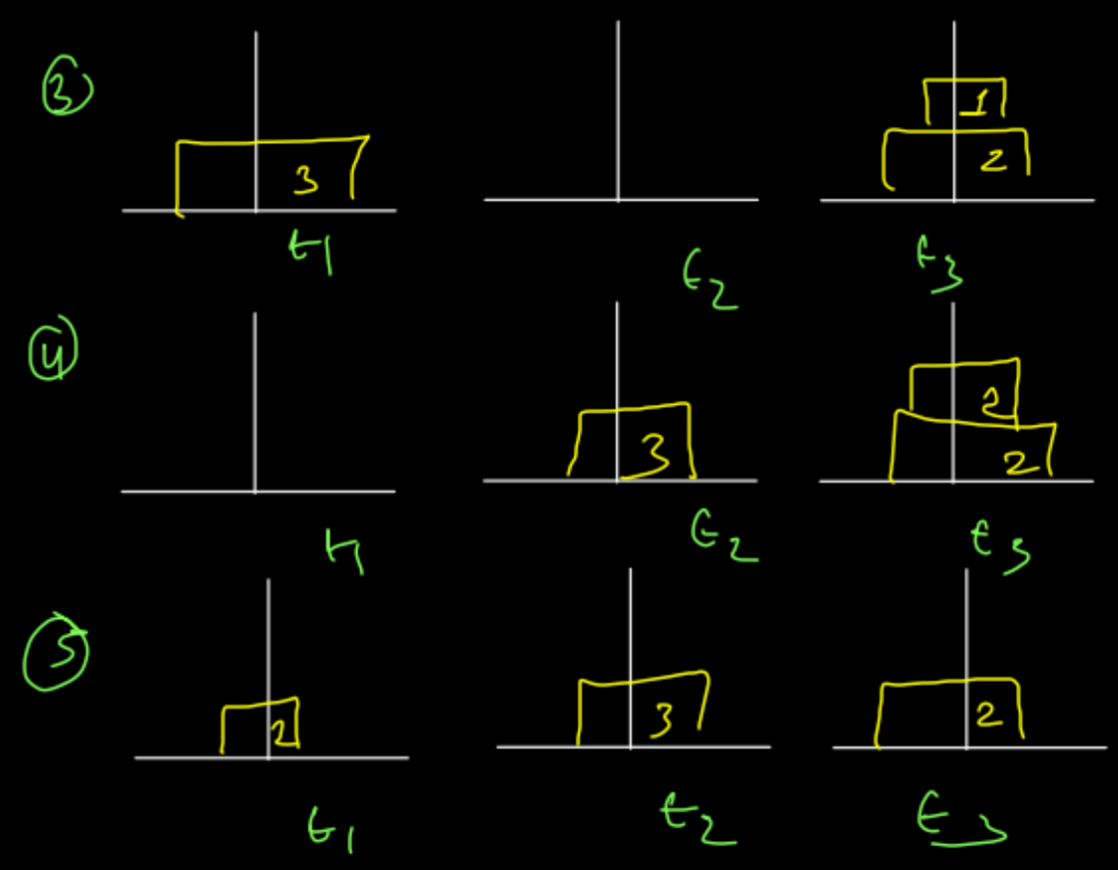
1. ~~1~~ from 1 to 2
2. ~~2~~ from 1 to 3
3. 1 from 2 to 3
4. 3 from 1 to 2
5. 1 from 3 to 1
6. 2 from 3 to 2
7. 1 from 1 to 2

Initial condition

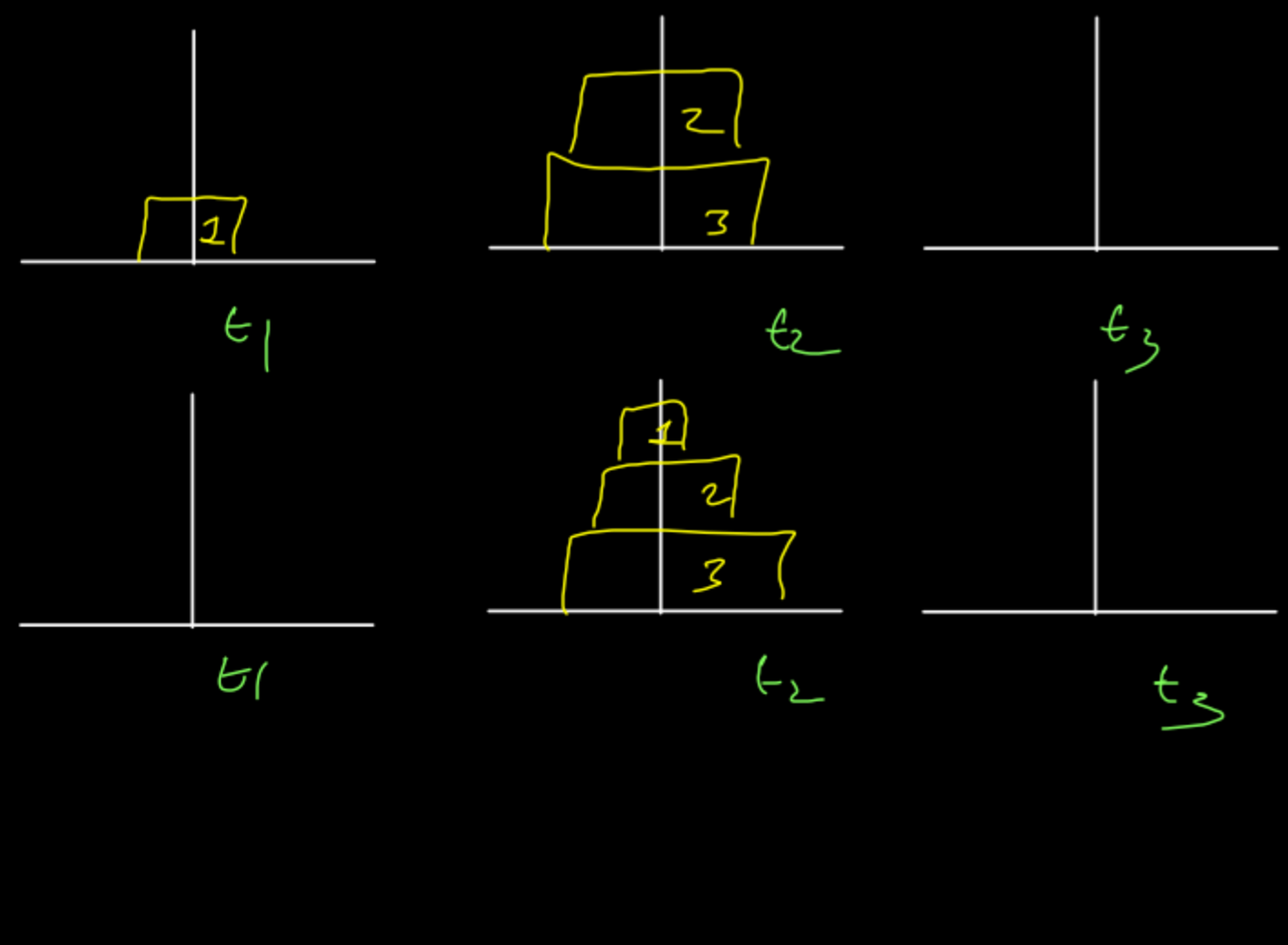


faith(4)

1. from 1 to 2
2. from 1 to 3
1. from 2 to 3
3. from 1 to 2
1. from 3 to 1
2. from 3 to 2
1. from 1 to 2



1. from 1 to 2
2. from 1 to 3
1. from 2 to 3
3. from 1 to 2
1. from 3 to 1
2. from 3 to 2
1. from 1 to 2



$t_1 = 1 \quad t_2 = 2 \quad t_3 = 3$

```java
// t1 -> src t2-> dest t3-> aux
public long toh(int N, int t1, int t2, int t3) {
    // Your code here

    if(N == 0) return 0;

    long ans1 = toh(N-1,t1,t3,t2); 1
    System.out.println("move disk " + N + " from rod " + t1 + " to ro
    long ans2 = toh(N-1,t3,t2,t1); 3

    return ans1 + ans2 + 1; 4
}
```

1. from 1 to 2
2. from 1 to 3
1. from 2 to 3
3. from 1 to 2
1. from 3 to 1
2. from 3 to 2
1. from 1 to 2

src dest aux

toh(N, t1, t2, t3)

... toh(3, t1, t2, t3) → 7

3

.... toh(2, t1, t3, t2) .... toh(2, t3, t2, t1)

1 1 1 1

.... toh(1, t1, t2, t3) : toh(1, t2, t3, t1) : toh(1, t3, t1, t2) : toh(1, t1, t2, t3)

0 0 0 0 0 0 0 0

toh(0) toh(0) toh(0) toh(0) toh(0) toh(0) toh(0) toh(0)

Tree

1. from 1 to 2
2. from 1 to 3
1. from 2 to 3
3. from 1 to 2
1. from 3 to 1
2. from 3 to 2
1. from 1 to 2

Stack

1. from 1 to 2
2. from 1 to 3
1. from 2 to 3
3. from 1 to 2
1. from 3 to 1
2. from 3 to 2
1. from 1 to 2

Lecture - 21

→ Display Array

→ Reverse Array

→ Master formula Introduction (Preorder, Inorder & Postorder)

→ Power **

- → Linear 1
- → Linear 2
- → Logarithmic

Print Elements of Array

School Accuracy: **52.5%** Submissions: **31753** Points: **0**

Given an array **Arr** of size **N**, print all its elements.

Example 1:

Input:
N = 5
Arr[] = {1, 2, 3, 4, 5}
Output: 1 2 3 4 5

① Expectation:

$\{9, 8, 6, 3, 4, 7, 2\}$ (indices 0 1 2 3 4 5 6)

display(arr, 0) → print space seperated elements. (N)
idx

9 8 6 3 4 7 2

② faith : display(arr, 1) → print space seperated elements (N-1)
idx+1

③ Meeting expec with faith :

idx 2nd

$\{9, 8, 6, 3, 4, 7, 2\}$ (indices 0 1 2 3 4 5 6)

Print yourself + " "
↳ 9 _

display(arr, idx+1)
↳ 8 6 3 4 7.

```java
void display(int[] arr, int idx) {

    |

    System.out.print(arr[idx] + " ");   1
    display(arr,idx + 1);               2
}
```

idx idx idx idx idx idx idx idx

0 1 2 3 4 5 6

{ 9, 8, 6, 3, 4, 7, 2 }

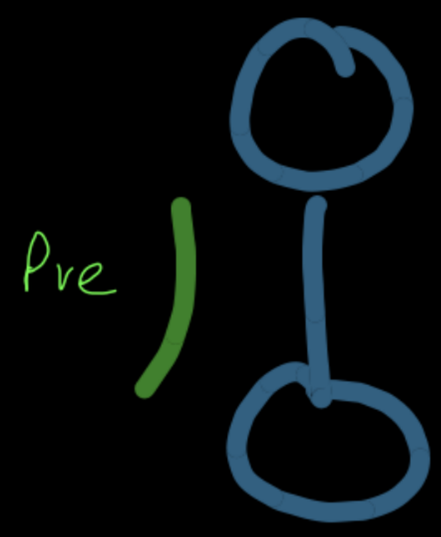
9_ 8_ 6_ 3_ 4_ 7_ 2_

```
if (idx = = arr.length)
    return;
```

→ Base Case

display (arr, 0)
↓
display (arr, 1)
↓
display (arr, 2)
↓
display (arr, 3)
↓
display (arr, 4)
↓
display (arr, 5)
↓
display (arr, 6)
↓
display (arr, 7)

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

    System.out.println(arr[idx]);
    displayArr(arr,idx + 1);
}
```



Reverse an Array

School Accuracy: **40.0%** Submissions: **100k+** Points: **0**

Given an array **A** of size **N**, print the reverse of it.

Example:

Input:
1
4
1 2 3 4
Output:
4 3 2 1

Expectation :

lo, hi

0 1 2 3 4

{ 1, 2, 3, 4, 5 }

↓

lo, hi

0 1 2 3 4

{ 5, 4, 3, 2, 1 }

reverse (arr, lo, hi);

lo → 0, hi → arr.length - 1

faith :

lo, hi

0 1 2 3 4

{ 1, 2, 3, 4, 5 }

↓

lo, hi

0 1 2 3 4

{ 1, 4, 3, 2, 5 }

reverse(arr, lo+1, hi-1);

Meeting expectation with faith :

① Swap lo & hi

② reverse (arr, lo+1, hi-1)

lo hi

0 1 2 3 4

{ 5, 2, 3, 4, 1 }

1

0 1 2 3 4

① { 5, 2, 3, 4, 1 }

lo hi

0 1 2 3 4

② { 5, 2, 3, 4, 1 } → { 5, 4, 3, 2, 1 }

0 1 2 3 4

```java
public static void reverse(int[] arr, int lo, int hi) {

    //swap yourself
    int temp = arr[lo];
    arr[lo] = arr[hi];
    arr[hi] = temp;

    reverse(arr,lo+1,hi-1);
}
```

① ②

lo lo lo hi hi hi

0 1 2 3 4

{ 1, 2, 3, 4, 5 }

5 4 2 1

..reverse (arr, 0, 4)

↓

..reverse (arr, 1, 3)

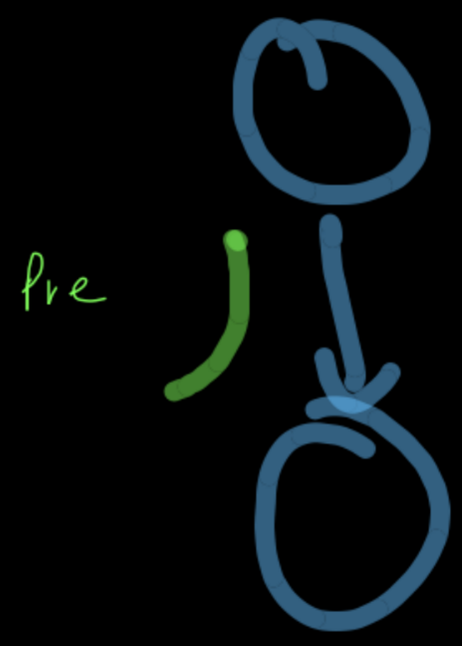
↓

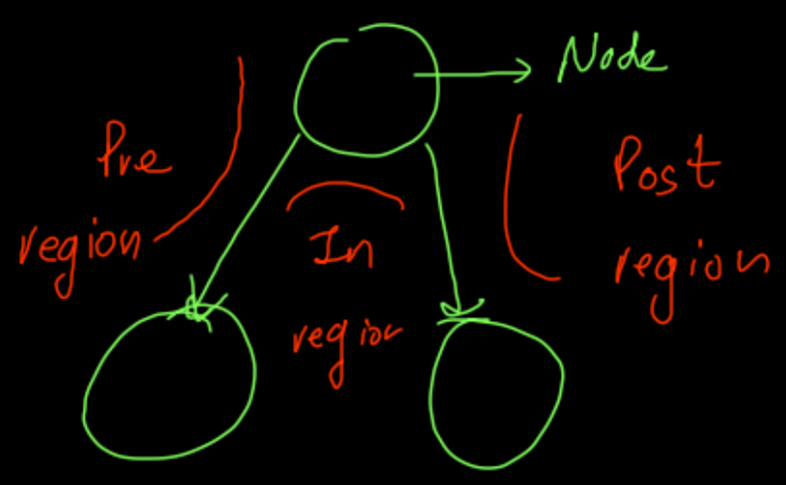
reverse (arr, 2, 2)

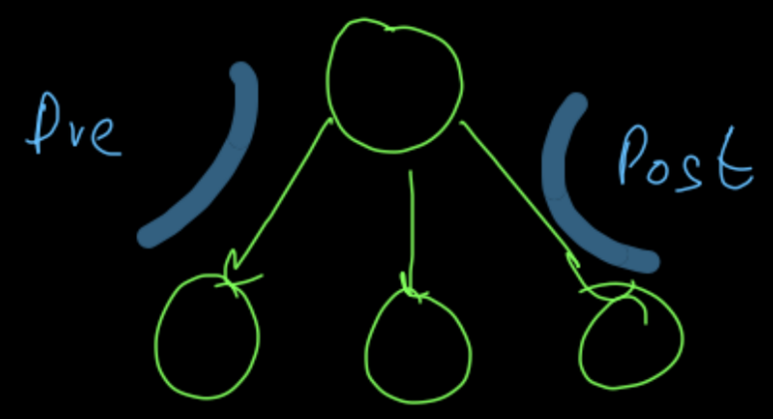
if (lo >= hi) return;

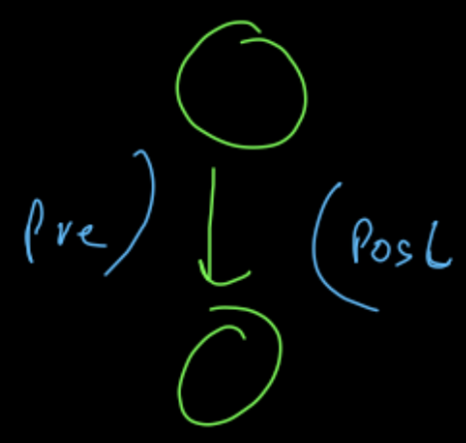
Base case

```java
public static void reverse(int[] arr, int lo, int hi) {

    if(lo >= hi) return;

    //swap yourself
    int temp = arr[lo];
    arr[lo] = arr[hi];
    arr[hi] = temp;

    reverse(arr,lo+1,hi-1);
}
```




Node
Pre
region
In
region
Post
region


Pre
Post


Pre
Post

Recursion Master formula

$$\text{Time Complexity} = (\text{calls})^{\text{height}} + (\text{Pre} + \text{Post}) * \text{height}$$

This is max no of calls from a node in the tree.

50. Pow(x, n)

Medium 5801 6346 Add to List Share

Implement pow(x, n), which calculates `x` raised to the power `n` (i.e., x^n).

Example 1:

```
Input: x = 2.00000, n = 10
Output: 1024.00000
```

① Expectation : power(x, n) → $\underbrace{x * x * x \cdots x}_{n \text{ times}}$

power(2, 7) → 2 * 2 * 2 * 2 * 2 * 2 * 2

② faith : power(x, n-1) → $\underbrace{x * x * x \cdots x}_{(n-1) \text{ times}}$

power(2, 6) → 2 * 2 * 2 * 2 * 2 * 2

③ Meeting expectation with faith : power(x, n) = power(x, n-1) * x

$x^n = x * x^{n-1}$

x^N

$\downarrow$

x^{N-1}

$\downarrow$

x^{N-2}

$\vdots$

$(num)^0 \rightarrow 1$

$0 \longrightarrow$ if (n == 0) return 1

$$x^{-N} = \frac{1}{x^N}$$

$$2^{-6} = \frac{1}{2^6}$$

$$2^{-6} = \frac{1}{2^{-(-6)}}$$

```java
class Solution {

    public double power(double x, int n) {
        if(n == 0) return 1;

        return x * power(x,n-1);

    }

    public double myPow(double x, int n) {
        if(n < 0) {
            return 1.0 / power(x,-n);
        }

        return power(x,n);
    }
}
```

This gives stack overflow

```java
class Solution {

    public double power(double x, int n) {
        if(n == 0) return 1;

        return x * power(x,n-1);

    }

    public double myPow(double x, int n) {
        if(n < 0) {
            return 1.0 / power(x,-n);
        }

        return power(x,n);
    }
}
```

$(\text{calls})^{h} + (\text{pre} + \text{post}) *$

$(1)^{N} + (\underbrace{k1 + k2}) * N = 1 + k^{N} \approx \underline{O(N)}$

pow(2,5) $\uparrow$ $2*16 = \underline{32}$

$\downarrow$

pow(2,4) $\uparrow$ $2*8 = 16$

$\downarrow$

pow(2,3) $\uparrow$ $2*4 = 8$

$\downarrow$

pow(2,2) $\uparrow$ $2*2 = 4$

$\downarrow$

pow(2,1) $\uparrow$ $2*1 = 2$

$\downarrow$

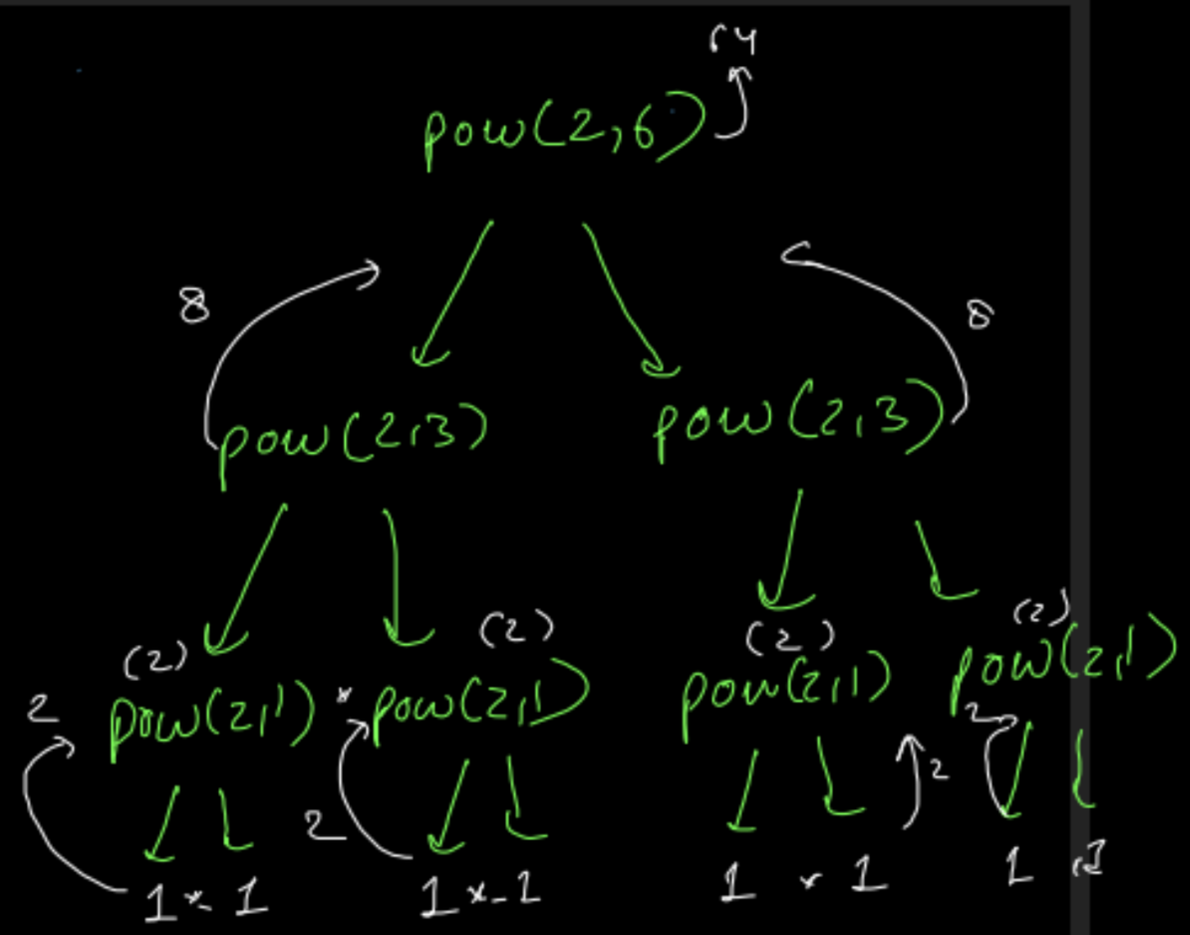
pow(2,0) $\uparrow$ 1

power(2,6) → power(2,3) * power(2,3)

power(2,7) → 2* pow(2,3) * pow(2,3)

```
# x^N
    if (N % 2 == 0)
        pow(x,n/2) * pow(x,n/2)
else
    x * pow(x,n/2) * pow(x,n/2)
```

```java
class Solution {

    public double power(double x, int n) {
        if(n == 0) return 1;

        if(n % 2 == 0)
            return power(x,n/2) * power(x,n/2);

        return x * power(x,n/2) * power(x,n/2);

    }

    public double myPow(double x, int n) {
        if(n < 0) {
            return 1.0 / power(x,-n);
        }

        return power(x,n);
    }
}
```



$(calls)^h + (pre + post) * h$

$(2)^h + (k) * h =$

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

$N + \log_2 N = O(N)$

pow(8)

↓

4

↓

2

↓

1

4

$\log_2 8 + 1$

$\approx O(\log_2 N)$

```
public double power(double x, int n) {
    if(n == 0) return 1;

    double smallAns = power(x,n/2);

    if(n % 2 == 0)
        return smallAns * smallAns;

    return x * smallAns * smallAns;

}

public double myPow(double x, int n) {
    if(n < 0) {
        return 1.0 / power(x,-n);
    }

    return power(x,n);
}
```

pow(2,6)

↓

pow(2,3)

↓

pow(2,2)

↓

∂

$(\text{calls})^h + (\text{pre} + \text{post}) * h$

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

$1 + \log_2 N \approx O(\log_2 N)$

Lecture - 23

1. Reverse Print Array
2. Check if an Array is Sorted
3. Max in an Array
4. find Index
5. All indices.

Reverse Print

Display Array In Reverse

Easy

Prev Next

1. You are given a number n, representing the size of array a.
2. You are given n numbers, representing elements of array a.
3. You are required to print the elements of array from end to beginning each in a separate line.
4. For the above purpose complete the body of displayArrReverse function. Don't change the signature.

Note -> The online judge can't force you to write the function recursively but that is what the spirit of question is. Write recursive and not iterative logic. The purpose of the question is to aid learning recursion and not test you.

① Expectation

0 1 2 3 4 5 6
{9, 8, 6, 3, 4, 7, 2}

displayReverse(arr, arr.length - 1)
6 idx

2
7
4
3
6
8
9

② faith : displayReverse(arr, idx - 1)
5

7
4
3
6
8
9

③ Meeting expectation with faith :

Print(arr[idx])

displayReverse(arr, idx - 1)

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

        System.out.println(arr[idx]);   1
        displayArrReverse(arr,idx-1);   2
    }
```

0 1 2 3 4 5 6

{9, 8, 6, 3, 4, 7, 2}

2
7
4
3
6
8
9

Base Case

```
if (idx == -1)
    return;
```

display(arr, 6)
↓
display(arr, 5)
↓
display(arr, 4)
↓
display(arr, 3)
↓
display(arr, 2)
↓
display(arr, 1)
↓
display(arr, 0)
↓
display(arr, -1)

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

        if(idx == -1) return;

        System.out.println(arr[idx]);
        displayArrReverse(arr,idx-1);
    }
```

Check if array is sorted

Easy Accuracy: **49.4%** Submissions: **47542** Points: **2**

Given an array **arr[]** of size **N**, check if it is sorted in non-decreasing order or not.

Example 1:

Input:
N = 5
arr[] = {10, 20, 30, 40, 50}
Output: 1
Explanation: The given array is sorted.

① Expectation :

$\{\overset{0}{10}, \overset{1}{20}, \overset{2}{30}, \overset{3}{30}, \overset{4}{40}, \overset{5}{50}\}$ → true

isSorted (arr, idx) → starting index

0

② faith :

$\{\overset{0}{10}, \overset{1}{20}, \overset{2}{30}, \overset{3}{30}, \overset{4}{40}, \overset{5}{50}\}$

isSorted (arr, idx+1) → starting index

③ Meeting expectation with faith :

$\{\overset{0}{10}, \overset{1}{20}, \overset{2}{30}, \overset{3}{30}, \overset{4}{40}, \overset{5}{50}\}$

```
if (arr[idx] > arr[idx+1])
    return false;
return isSorted(arr, idx+1);
```

```java
class Solution {

    boolean isSorted(int[] arr, int idx) {

        if(arr[idx] > arr[idx + 1]) return false;  1

        return isSorted(arr,idx + 1);  2
    }

    boolean arraySortedOrNot(int[] arr, int n) {
        // code here

    }
}
```

isSorted(arr, 0)
↓
isSorted(arr, 1)
↓
isSorted(arr, 2)
↓
isSorted(arr, 3)
↓
isSorted(arr, 4)
↓
isSorted(arr, 5)

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

Base Case

```
if(idx == arr.length - 1)
    return true;
```

$\{\overset{0}{10}, \overset{1}{20}, \overset{2}{5}, \overset{3}{30}\}$

isSorted(arr, 0) false

↓

isSorted(arr, 1) false

```
class Solution {

    boolean isSorted(int[] arr, int idx) {

        if(arr[idx] > arr[idx + 1]) return false;  1

        return isSorted(arr,idx + 1);  2
    }

    boolean arraySortedOrNot(int[] arr, int n) {
        // code here

    }
}
```

```
class Solution {

    boolean isSorted(int[] arr, int idx) {

        if(idx == arr.length -1) return true;

        if(arr[idx] > arr[idx + 1]) return false;

        return isSorted(arr,idx + 1);
    }

    boolean arraySortedOrNot(int[] arr, int n) {
        // code here
        return isSorted(arr,0);
    }
}
```

Largest Element in Array

Basic Accuracy: **68.83%** Submissions: **100k+** Points: **1**

Given an array **A[]** of size **n**. The task is to find the largest element in it.

Example 1:

Input:
n = 5
A[] = {1, 8, 7, 56, 90}
Output:
90
Explanation:
The largest element of given array is 90.

① Expectation :

$\{\overset{0}{10}, \overset{1}{7}, \overset{2}{6}, \overset{3}{3}, \overset{4}{9}, \overset{5}{4}, \overset{6}{2}\} \longrightarrow 9$

$\max(arr, \underset{(0)}{idx}) \longrightarrow 10$

② faith :

$\{\overset{0}{10}, \overset{1}{7}, \overset{2}{6}, \overset{3}{3}, \overset{4}{9}, \overset{5}{4}, \overset{6}{2}\} \longrightarrow 9$

$\max(arr, \underset{(1)}{idx} + 1)$

③ Meeting expectation with faith :

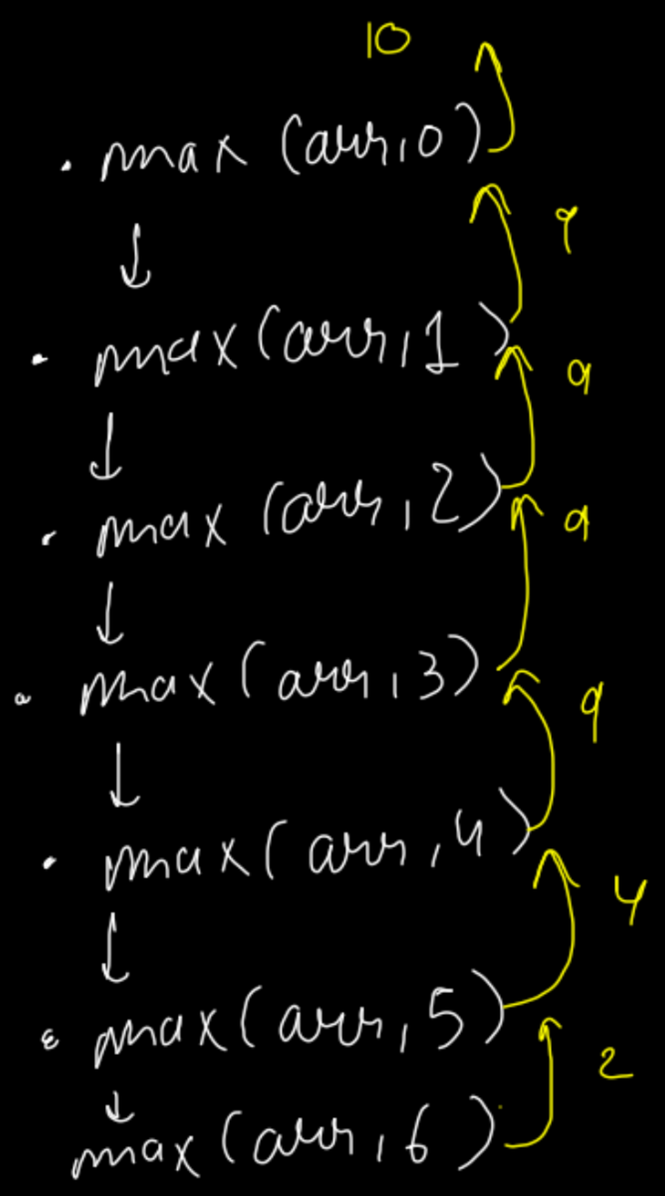
max of Rem Array = $\max(arr, idx+1) \rightarrow 9$

return Math.max(arr[idx], max of Rem Array);

```java
public int maximum(int[] arr,int idx) {

    int maxOfRemArray = maximum(arr,idx + 1); 1
    return Math.max(arr[idx],maxOfRemArray); 2
}
```

0	1	2	3	4	5	6
10	7	6	3	9	4	2

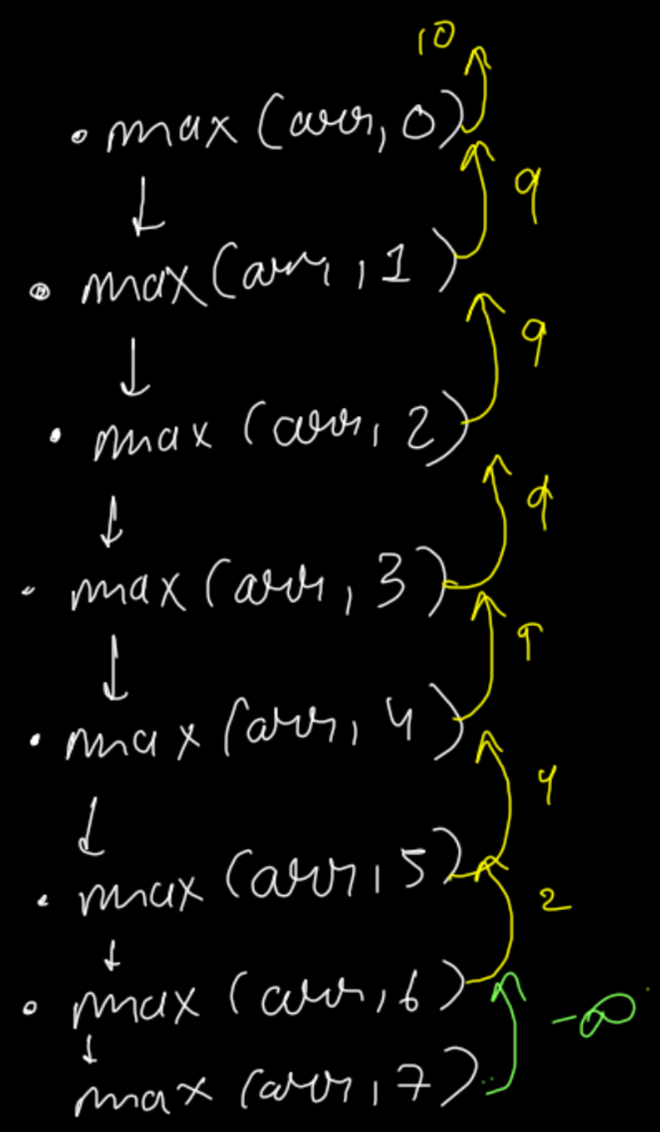
```
# Base Case
    if(idx == arr.length-1);
        return arr[idx];
```



```java
public int maximum(int[] arr,int idx) {

    int maxOfRemArray = maximum(arr,idx + 1); 1
    return Math.max(arr[idx],maxOfRemArray); 2
}
```

0	1	2	3	4	5	6
10	7	6	3	9	4	2

```
#Base Case
if (idx == arr.length)
```



```java
class Compute {

    public int maximum(int[] arr,int idx) {

        if(idx == arr.length -1) return arr[idx];

        int maxOfRemArray = maximum(arr,idx + 1);
        return Math.max(arr[idx],maxOfRemArray);
    }

    public int largest(int arr[], int n)
    {
        return maximum(arr,0);
    }
}
```

Find Index

School Accuracy: **48.64%** Submissions: **21766** Points: **0**

Given an unsorted array **Arr[]** of **N** integers and a **Key** which is present in this array. You need to write a program to find the **start index**(index where the element is first found from left in the array) and **end index**(index where the element is first found from right in the array).

first Index

{ 1, 4, 5, 2, 3, 5, 8, 9 } target = 5

findIndex (arr, target) [2, 5]

① Expectation : { 1, 4, 5, 2, 3, 5, 8 } target = 5 → 2

firstIdx (arr, idx, target)

② faith : firstIdx (arr, idx+1, target)

③ Meeting expec : if (arr[idx] == target) return idx;

return firstIdx (arr, idx+1, target)

```java
static int firstIndex(int[] arr, int idx, int target) {

    |

    if(arr[idx] == target) return idx;

    return firstIndex(arr,idx + 1,target);
}
```

{ 1, 4, 5, 2, 3, 5, 8, 9 }

0 1 2 3 4 5 6

target=5

fi(arr, 0, 5) 2

↓

fi(arr, 1, 5) 2

↓

fi(arr, 2, 5)

```java
static int firstIndex(int[] arr, int idx, int target) {

    |

    if(arr[idx] == target) return idx;

    return firstIndex(arr,idx + 1,target);
}
```

{ 1, 4, 5, 2, 3, 5, 8, 9 }

0 1 2 3 4 5 6

target=10

fi(arr, 0, 10)

↓

fi(arr, 1, 10)

↓

fi(arr, 2, 10)

↓

fi(arr, 3, 10)

↓

fi(arr, 4, 10)

↓

fi(arr, 5, 10)

↓

fi(arr, 6, 10)

↓

fi(arr, 7, 10)

Base Case

if(idx==arr.length)

return -1

Last Index

Expectation :

0	1	2	3	4	5	6
1	4	5	2	3	5	8

target = 5

5

lastIndex(arr, idx, target)
(6)

faith :

0	1	2	3	4	5	6
1	4	5	2	3	5	8

lastIndex(arr, idx - 1, target)

Meeting expec :

```
if (arr[idx] == target) return idx
    return lastIndex(arr, idx - 1, target)
```

```java
static int firstIndex(int[] arr, int idx, int target) {

    if(idx == arr.length) return -1;

    if(arr[idx] == target) return idx;

    return firstIndex(arr,idx + 1,target);
}

static int lastIndex(int[] arr, int idx, int target) {

    if(idx == -1) return -1;

    if(arr[idx] == target) return idx;

    return lastIndex(arr,idx -1,target);
}
```

```java
    // Function to find starting and end index
    static int[] findIndex(int arr[], int N, int key)
    {
        //code here.
        int[] res = new int[2];
        res[0] = firstIndex(arr,0,key);
        res[1] = lastIndex(arr,N-1,key);
        return res;
    }
}
```

All Indices Of Array

Easy

Prev Next

1. You are given a number n, representing the count of elements.
2. You are given n numbers.
3. You are given a number x.
4. You are required to find the all indices at which x occurs in array a.
5. Return an array of appropriate size which contains all indices at which x occurs in array a.

Note -> The online judge can't force you to write the function recursively but that is what the spirit of question is. Write recursive and not iterative logic. The purpose of the question is to aid learning recursion and not test you.

idx idx idx idx idx idx idx
0 1 2 3 4 5 6

{15, 11, 40, 4, 4, 9, 4}

target = 4

```
public static int[] allIndices(int[] arr, int x, int idx, int fsf) {
    // write ur code here
}
```

arr = 4k	tar = 4	idx = 7	fsf = 3
arr = 4k	tar = 4	idx = 6	fsf = 2
arr = 4k	tar = 4	idx = 5	fsf = 2
arr = 4k	tar = 4	idx = 4	fsf = 2
arr = 4k	tar = 4	idx = 3	fsf = 0
arr = 4k	tar = 4	idx = 2	fsf = 0
arr = 4k	tar = 4	idx = 1	fsf = 0
arr = 4k	tar = 4	idx = 0	fsf = 0

→ create an array of size fsf = 3

[3, 4, 6]

if (arr[idx] == x

res[fsf] = idx;

```java
    public static int[] allIndices(int[] arr, int x, int idx,
int fsf) {
        // write ur code here

        if(idx == arr.length) {
            int[] res = new int[fsf];
            return res;
        }

        if(arr[idx] == x) {
            int[] res = allIndices(arr,x,idx + 1,fsf + 1);
            res[fsf] = idx;
            return res;
        } else {
            int[] res = allIndices(arr,x,idx + 1,fsf);
            return res;
        }
    }

}
```

Recursion & Backtracking - Lecture 5

Subsets
- → Iteration
- → Print
- → Get

Difference b/w Get & Print

Subset (Leetcode)

Get Subsequence

Print Subsequence

Subsets Of Array

Easy

Prev Next

1. You are given a number n, representing the count of elements.
2. You are given n numbers.
3. You are required to print all subsets of arr. Each subset should be
on separate line. For more clarity check out sample input and output.

$\{\overset{0}{10}, \overset{1}{20}, \overset{2}{30}\}$

Subset : Every element has 2 choices.

No of subsets $= 2^N$

No Yes

- - -
- - 30
- 20 -
- 20 30
10 - -
10 - 30
10 20 -
10 20 30

Iterative Method

$\{\overset{0}{10}, \overset{1}{20}, \overset{2}{30}\}$

- - -	0 0 0	→ 0
- - 30	0 0 1	→ 1
- 20 -	0 1 0	→ 2
- 20 30	0 1 1	→ 3
10 - -	1 0 0	→ 4
10 - 30	1 0 1	→ 5
10 20 -	1 1 0	→ 6
10 20 30	1 1 1	→ 7

int → 4 bytes

32 bit

$i = 6$

		rem	
2	6	0	1
2	3	1	2
2	1	1	3
	0	1	

110

110

```java
int noOfSubsets = (int)(Math.pow(2,n));

for(int i=0;i<noOfSubsets;i++) {

    int temp = i;
    String subset = "";

    for(int j=n-1;j>=0;j--) {
        int rem = temp % 2;
        temp /= 2;

        if(rem == 0) {
            subset = "-\t" + subset;
        } else {
            subset = arr[j] + "\t" + subset;
        }
    }

    System.out.println(subset);
}
```

$\{10, 20, 30\}$

i = 6

temp = 6/2 = 3/2 = 1/2 = 0

subset = " 20 - "

j = 2 1 0

rem = 1 % 2 = 1

subset = " 20 - "

" 10 20 - "

Subsets of an Array (Get Method)

$\{\overset{0}{10}, \overset{1}{20}, \overset{2}{30}\}$

① Expectation : subsets(arr, $\underset{0}{idx}$) → ["", "30", "20", "20,30", "10", "10,30", "10,20", "10,20,30"]

② faith : subsets(arr, $\underset{1}{idx+1}$) → ["", "20", "30", "20,30"]

$\{20, 30\}$

③ Meeting Expectation with faith :

idx

$\{\overset{0}{10}, \overset{1}{20}, \overset{2}{30}\}$

{ }	{10}
{20}	{10,20}
{30}	{10,30}
{20,30}	{10,20,30}

idx

$\{\overset{0}{10}, \overset{1}{20}, \overset{2}{30}\}$

{ }
{20}
{30}
{20,30}

```java
public static ArrayList<String> subsets(int[] arr, int idx)
{
    ArrayList<String> smallAns = subsets(arr,idx + 1);
    ArrayList<String> res = new ArrayList<>();


}
```

idx

0 1 2

{10, 20, 30} $\xrightarrow{\text{faith}}$ ["--", "-30", "20-", "2030"]

no

idx

0 1 2

{10, 20, 30} $\longrightarrow$ ["---", "--30", "-20-", "-2030", "10--", "10-30", "1020-", "1020 30"]

```java
public static ArrayList<String> subsets(int[] arr, int idx) {
    ArrayList<String> smallAns = subsets(arr,idx + 1);
    ArrayList<String> res = new ArrayList<>();

    //no call
    for(String str : smallAns) {
        str = "-\t" + str;
        res.add(str);
    }

    //yes call
    for(String str : smallAns) {
        str = arr[idx] + "\t" + str;
        res.add(str);
    }

    return res;
}
```

31k

. Subsets (arr, 0)

3k res = ["---", "--30", "-20-", "-2030", "10--", "10-30", "1020-", "102030"]

11k

SA = ["--", "-30", "20-", "2030"]

. Subsets (arr, 1)

1k res = ["--", "-30", "20-", "2030"]

7k

SA = ["-", "30"]

. Subsets (arr, 2)

2k RS = ["-", "30"]

SA = [" "]

Subsets (arr, 3)

[" "]

4k

idx

0 1 2

{10, 20, 30}

∅

∅

```java
public static ArrayList<String> subsets(int[] arr, int idx) {

    if(idx == arr.length) {
        ArrayList<String> baseRes = new ArrayList<>();
        baseRes.add("");
        return baseRes;
    }

    ArrayList<String> smallAns = subsets(arr,idx + 1);
    ArrayList<String> res = new ArrayList<>();

    //no call
    for(String str : smallAns) {
        str = "-\t" + str;
        res.add(str);
    }

    //yes call
    for(String str : smallAns) {
        str = arr[idx] + "\t" + str;
        res.add(str);
    }

    return res;
}
```

012
"abc"

```java
public static ArrayList<String> gss(String str,int idx) {

    if(idx == str.length()) {
        ArrayList<String> res = new ArrayList<>();
        res.add("");
        return res;
    }

    ArrayList<String> smallAns = gss(str,idx + 1);
    ArrayList<String> res = new ArrayList<>();

    for(String subsequence : smallAns) {
        res.add(subsequence);
    }

    for(String subsequence : smallAns) {
        subsequence = str.charAt(idx) + subsequence;
        res.add(subsequence);
    }

    return res;
}
```

14k gss(abc, 0)
14k res = ["", "c", "b", "bc", "a", "ac", "ab", "abc"]
7k sA = ["", "c", "b", "bc"]

gss(abc, 1)
7k res = ["", "c", "b", "bc"]
11k sA = ["", "c"]

gss(abc, 2)
11k res = ["", "c"]
4k sA = ["", "c"]

gss(abc, 3)
4k ["", "c"]

In get category, work will be the least in base case. Width of the tree will be the least in B.C.

abc ⟶ ["", "a", "b", "c", "ab", "ac", "bc", "abc"]

n

Total No of chars in AL $= \frac{2^n * n}{2} = 2^{n-1} \cdot n$

If $n = 32 = 2^{32-1} \cdot 32$

$= (2^{31}) \cdot 32 \approx (2)^{30} \cdot 32$

$= (2^{10})^3 \cdot 32$

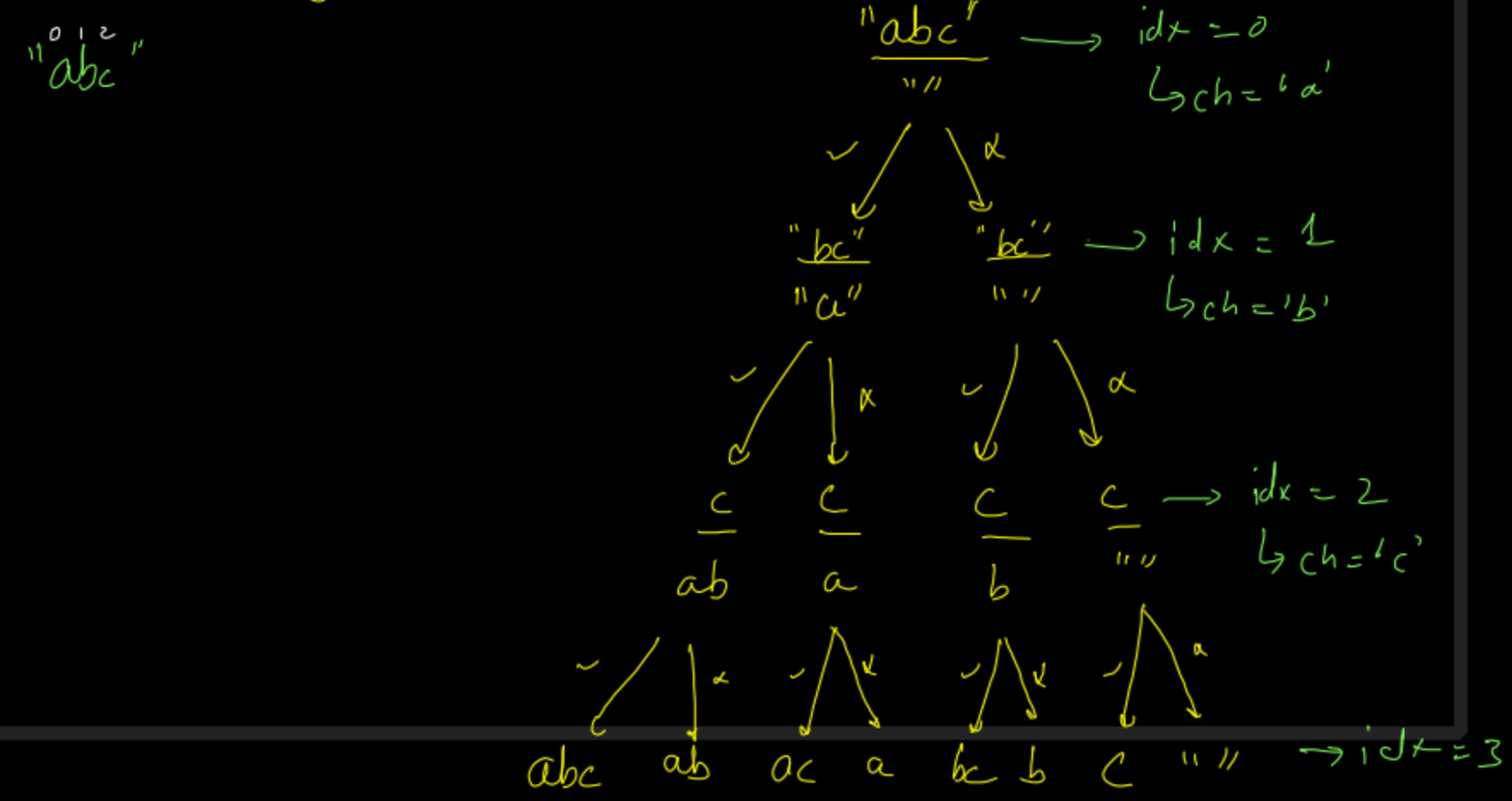
$(1024)^3 \cdot 32$

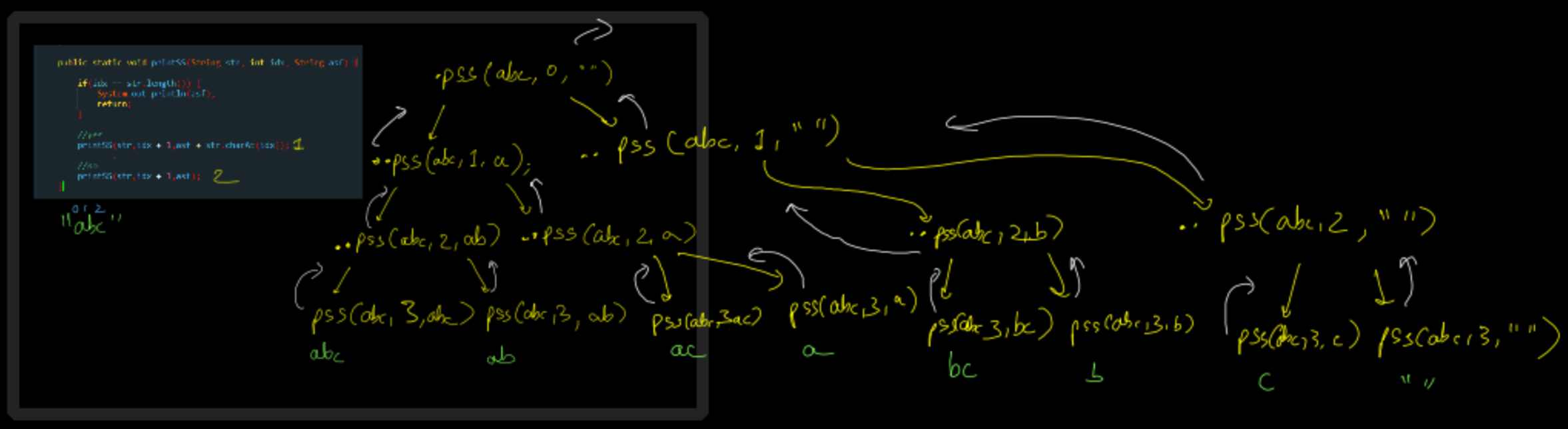
$\approx 10^9 \times 32$

64×10^9 Bytes

$\boxed{64\ GB}$

Print Category (Print Subsequence)






```java
public static void printSS(String str, int idx, String asf) {

    if(idx == str.length()) {
        System.out.println(asf);
        return;
    }

    //yes
    printSS(str,idx + 1,asf + str.charAt(idx)); 1

    //no
    printSS(str,idx + 1,asf);   2
}
```

Lecture-6

Subsets of An Array (Pepcoding) → Print Method

Subsets (Leetcode) → Get

→ Print

- Introduction to Backtracking
- When to deep copy?
- Global/static variable strategy.

Subsets-2 (Leetcode) → Print Method.

Subsets (Print Method)

Subsets Of Array

Easy

Prev Next

1. You are given a number n, representing the count of elements.
2. You are given n numbers.
3. You are required to print all subsets of arr. Each subset should be on separate line. For more clarity check out sample input and output.

idx idx idx

0 1 2

{10, 20, 30}

— — —

— — 30

— 20 —

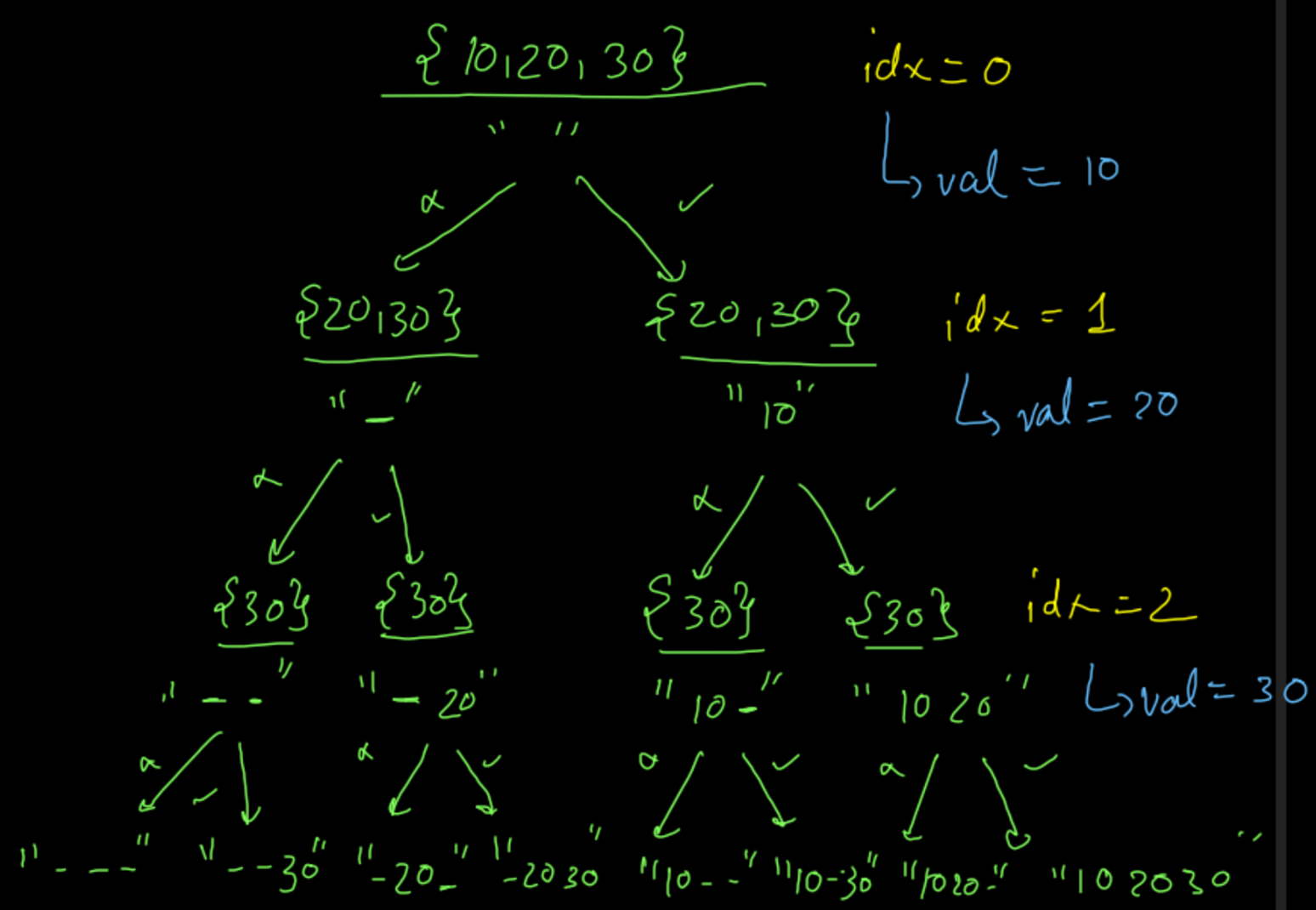
— 20 30

10 — —

10 — 30

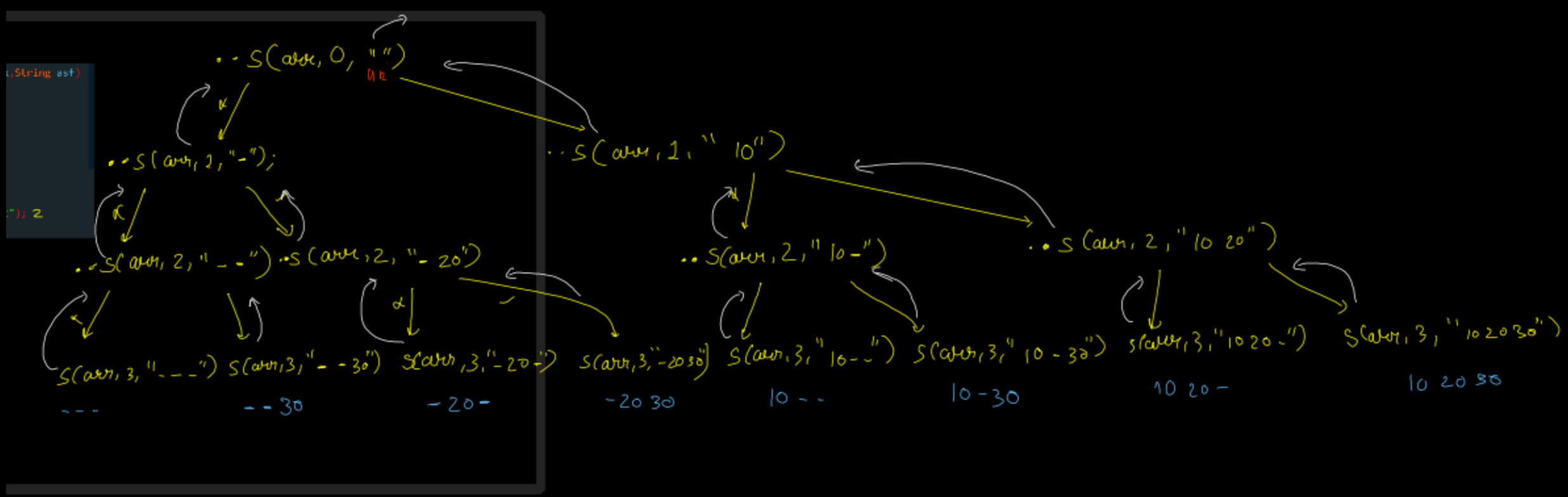
10 20 —

10 20 30



```java
public static void subsets(int[] arr,int idx,String asf)
{
    if(idx == arr.length) {
        System.out.println(asf);
        return;
    }

    //no
    subsets(arr,idx + 1,asf + "-\t");  1

    //yes
    subsets(arr,idx + 1,asf + arr[idx] + "\t"); 2
}
```

{ 10, 20, 30 }
0 1 2



Subsets (Get Method)

78. Subsets

Medium 12222 176 Add to List Share

Given an integer array `nums` of **unique** elements, return *all possible subsets (the power set)*.

The solution set **must not** contain duplicate subsets. Return the solution in **any order**.

Expectation:

idx 0 1 2

{10, 20, 30} → [[], [10], [20], [30], [10,20], [20,30], [10,30], [10,20,30]]

↳ subsets(nums, 0)

Faith

idx 0 1 2

{10, 20, 30} → [[], [20], [30], [20,30]] → smallAns

↳ subsets(nums, 1)

Meeting expectation with faith

↳ Make your own ans AL.

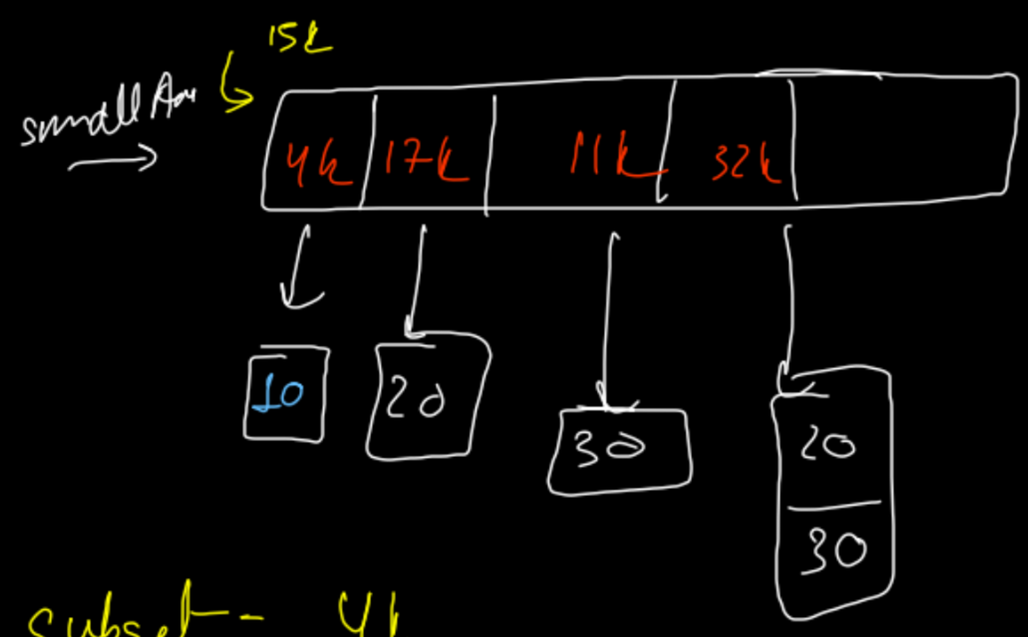
↳ Add all ALs from smallAns to ans.

↳ Add all ALs from smallAns to ans by adding nums[idx] to every AL in smallAns

```java
for(List<Integer> subset : smallAns) {
    res.add(subset);

    subset.add(0,nums[idx]);
    res.add(subset);
}
```

4k 17k 11k 32k

→ [[ ], [20], [30], [20,30]]



[4k 4k

[[ ][10]

4k 4k

[[10] [10]

subset = 4k

[4k

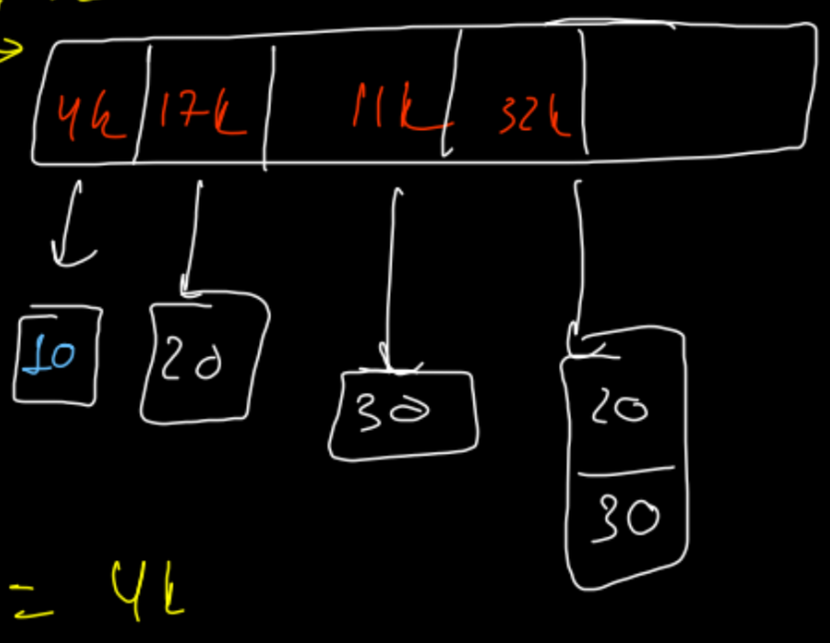
[10]

[[ ] [10]]

4k 17k 11k 32k

→ [[ ] , [20] , [30] , [20, 30]]

small Arr → 15k



subset = 4k

```java
public List<List<Integer>> subsets(int[] nums,int idx) {

    if(idx == nums.length) {
        List<List<Integer>> res = new ArrayList<>();
        res.add(new ArrayList<>());
        return res;
    }

    List<List<Integer>> smallAns = subsets(nums,idx + 1);
    List<List<Integer>> res = new ArrayList<>();

    for(List<Integer> subset : smallAns) {
        res.add(subset);

        //deep copy the subset
        List<Integer> completeSubset = new ArrayList<>();
        completeSubset.add(nums[idx]);

        for(int val : subset) {
            completeSubset.add(val);
        }

        res.add(completeSubset);
    }

    return res;
}

public List<List<Integer>> subsets(int[] nums) {
    return subsets(nums,0);
}
```

{10, 20, 30}

32 k

• S(nums, 0)

res = [[] [10] [20] [10 20] [30] [10 30] [20,30] [10,20,30]]

↳ 32k

SA = [[] [20] [30] [20,30]]

• S(nums, 1)

res = [[] [20] [30] [20,30]] → 17k

SA = [[] [30]]

• S(nums, 2)

res [[] [30]] → 11k

SA = [[]]

S(nums, 3)

4k [[]]

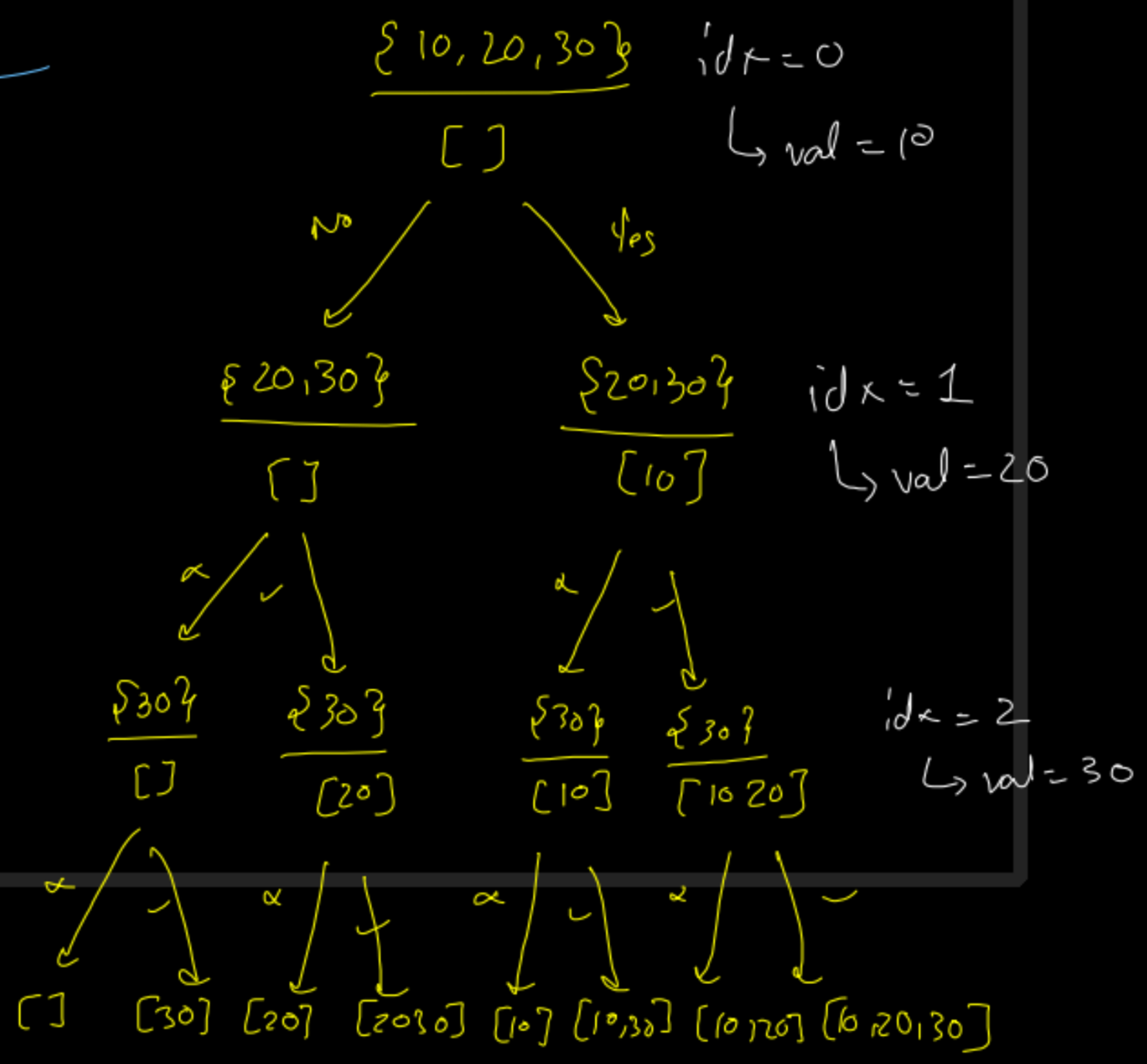
Strings → we don't need to deep copy because of interning.

```java
public List<List<Integer>> subsets(int[] nums,int idx) {

    if(idx == nums.length) {
        List<List<Integer>> res = new ArrayList<>();
        res.add(new ArrayList<>());
        return res;
    }

    List<List<Integer>> smallAns = subsets(nums,idx + 1);
    List<List<Integer>> res = new ArrayList<>();

    for(List<Integer> subset : smallAns) {
        res.add(subset);

        //deep copy the subset
        List<Integer> completeSubset = new ArrayList<>();
        completeSubset.add(nums[idx]);

        for(int val : subset) {
            completeSubset.add(val);
        }

        res.add(completeSubset);
    }

    return res;
}

public List<List<Integer>> subsets(int[] nums) {
    return subsets(nums,0);
}
```

{10, 20, 30}

Subsets (Print Method)

$\{\overset{0}{10}, \overset{1}{20}, \overset{2}{30}\}$



```java
List<List<Integer>> ans;

public void subsets(int[] nums,int idx,List<Integer> curr) {

    if(idx == nums.length) {
        ~~ans.add(curr);~~
        return;
    }

    //no
    subsets(nums,idx + 1,curr);

    //yes
    curr.add(nums[idx]);
    subsets(nums,idx + 1,curr);
}

public List<List<Integer>> subsets(int[] nums) {

    ans = new ArrayList<>();
    subsets(nums,0,new ArrayList<>());
    return ans;
}
```

copy = new ArL<>(curr);

0 1 2

{10, 20, 30}

• S(nums, 0, [30]) 4k

• S(nums, 1, [30]) 4k

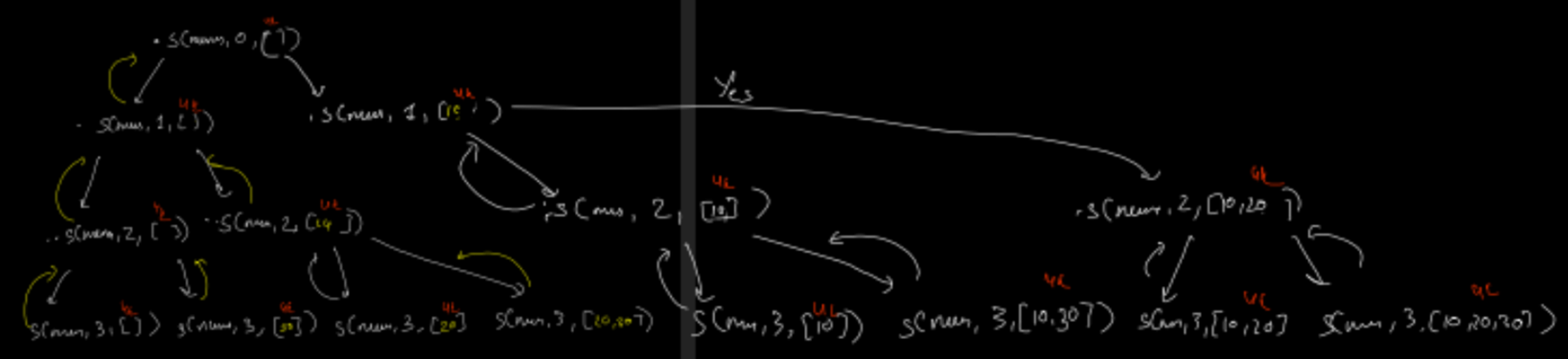
• S(nums, 2, [30]) 4k

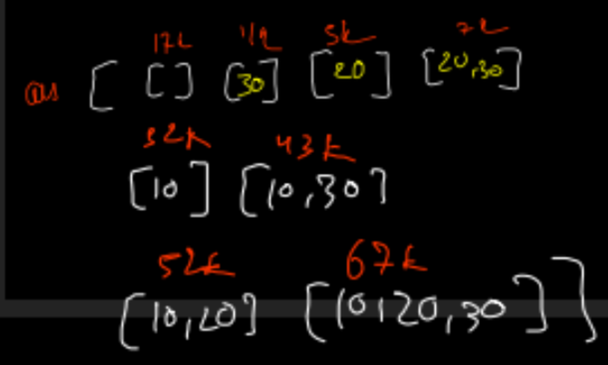
S(nums, 3, [ ]) 4k S(nums, 3, [30]) 4k

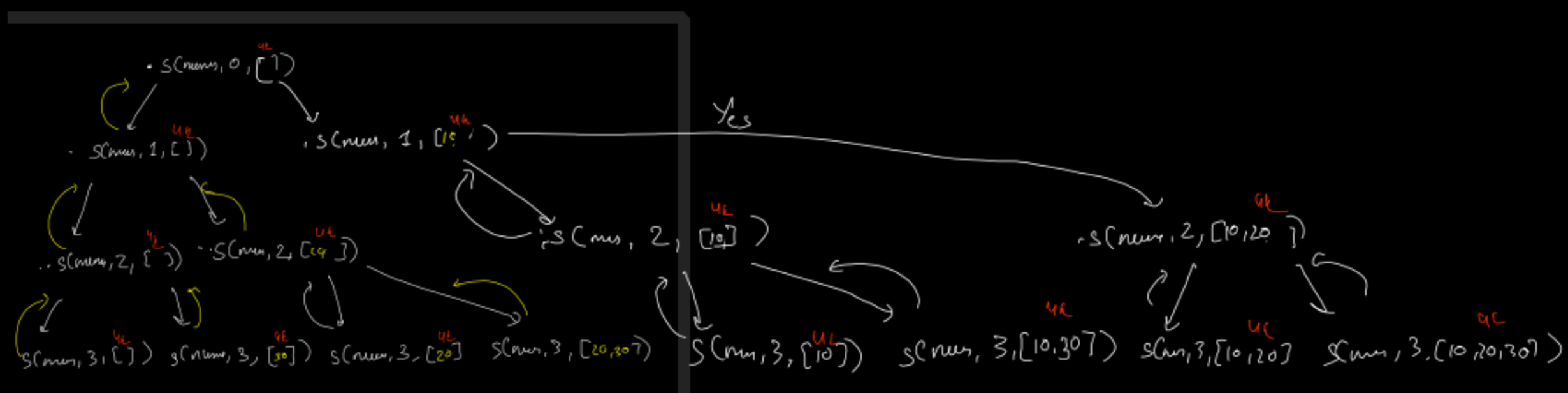
ans [[ ] [30]] 17k 11k

[[ ]]

```java
List<List<Integer>> ans;

public void subsets(int[] nums,int idx,List<Integer> curr) {

    if(idx == nums.length) {
        //deep copy
        List<Integer> copy = new ArrayList<>(curr);
        ans.add(copy);
        return;
    }

    //no
    subsets(nums,idx + 1,curr);

    //yes
    curr.add(nums[idx]);
    subsets(nums,idx + 1,curr);
    curr.remove(curr.size() -1); //backtracking
}
```







Lecture-7

Recursion & Backtracking

(Subsets & Maze Path Problems)

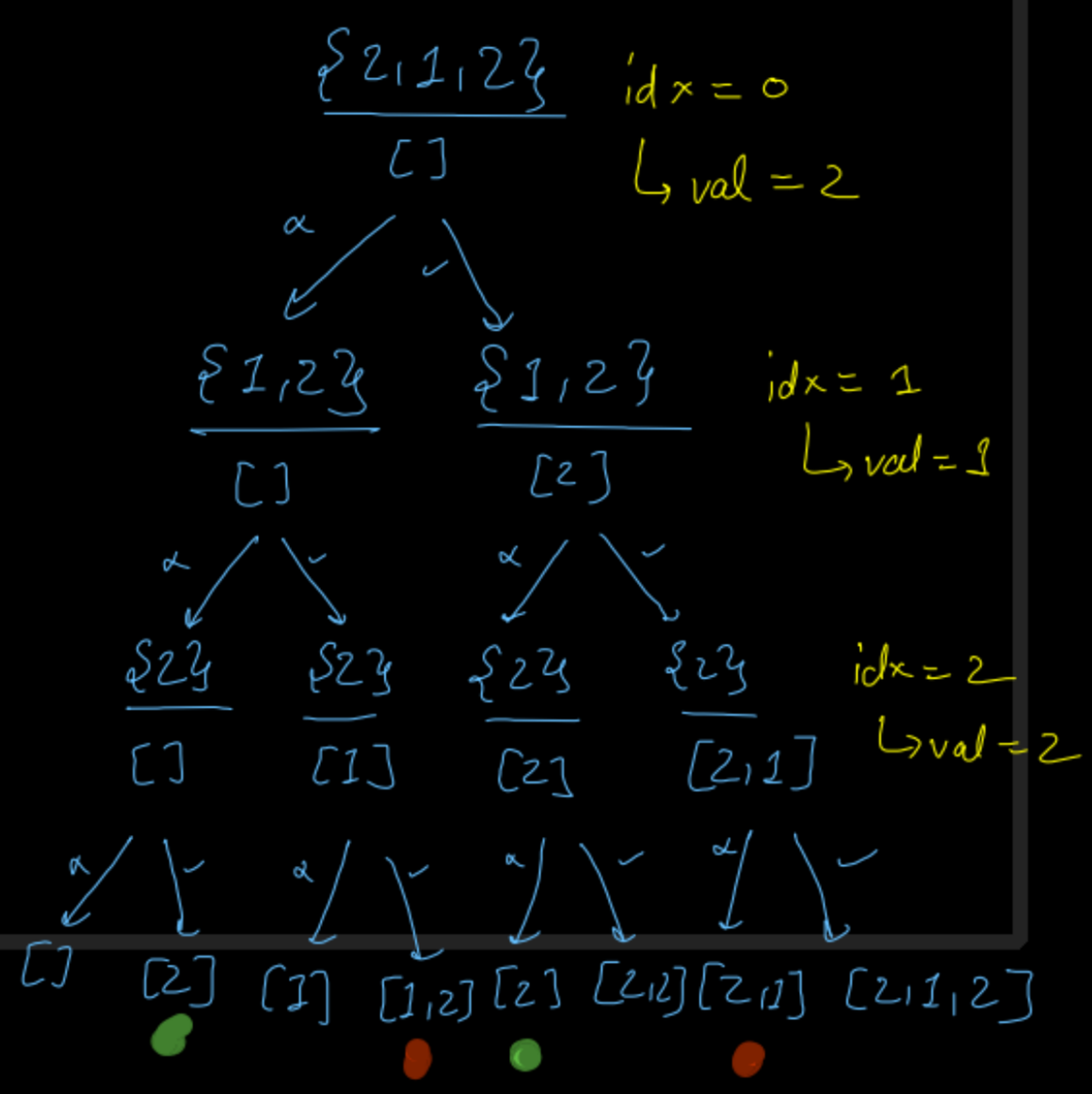
1. Unique Subsets (Subsets - II)
2. Print Stair Paths
3. Get Stair Paths

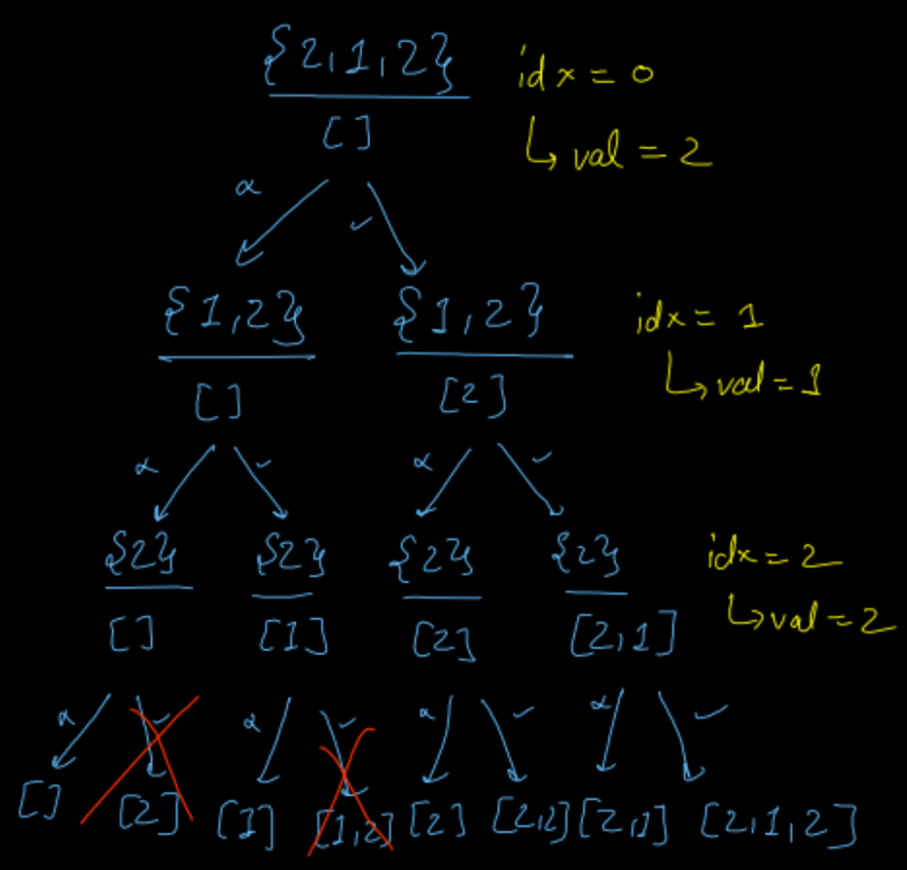
} Concept of Pruning.

4. Matchsticks to Square

Subsets-2

$\{\overset{0}{2}, \overset{1}{1}, \overset{2}{2}\}$





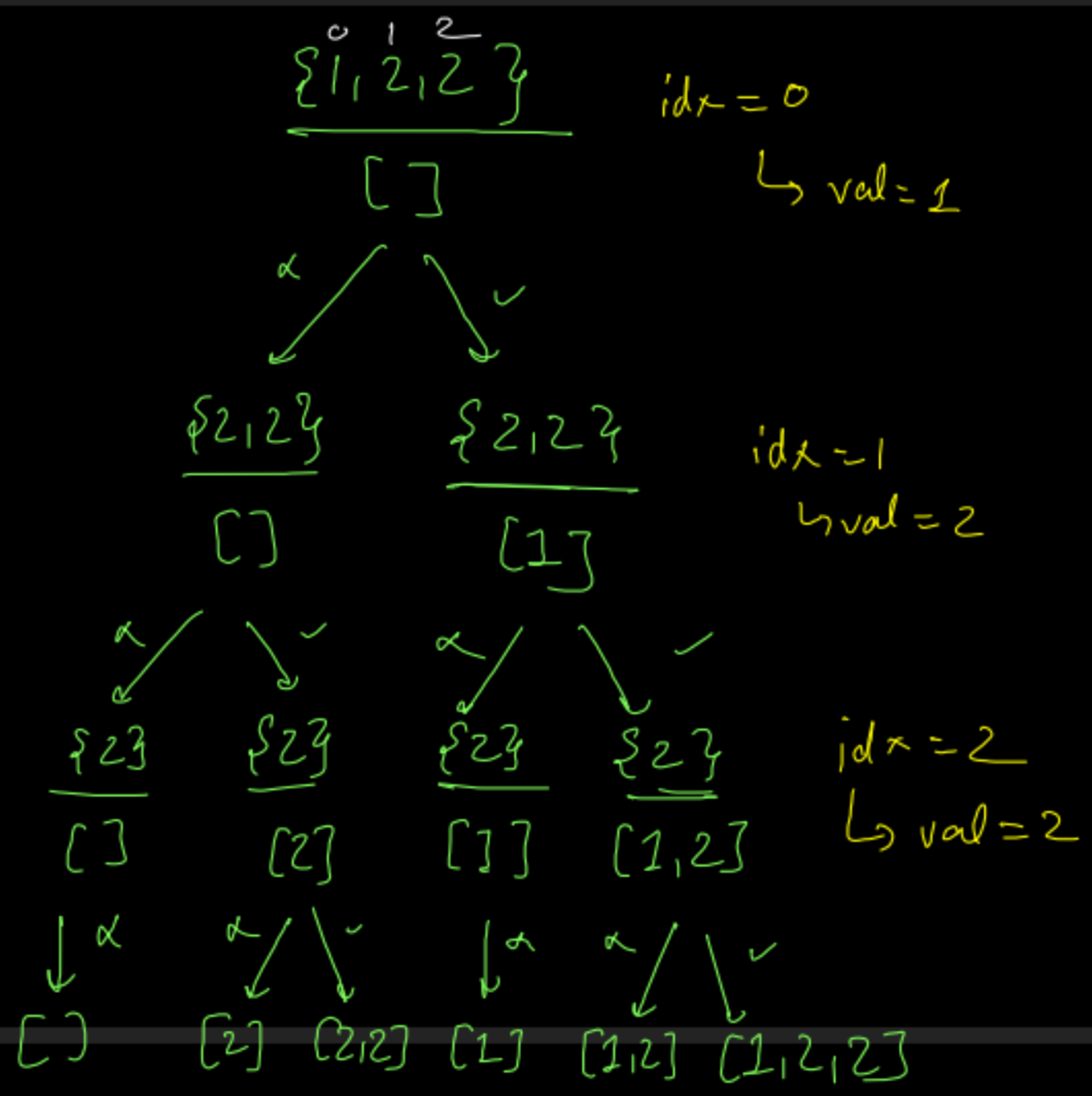
{2, 1, 2}

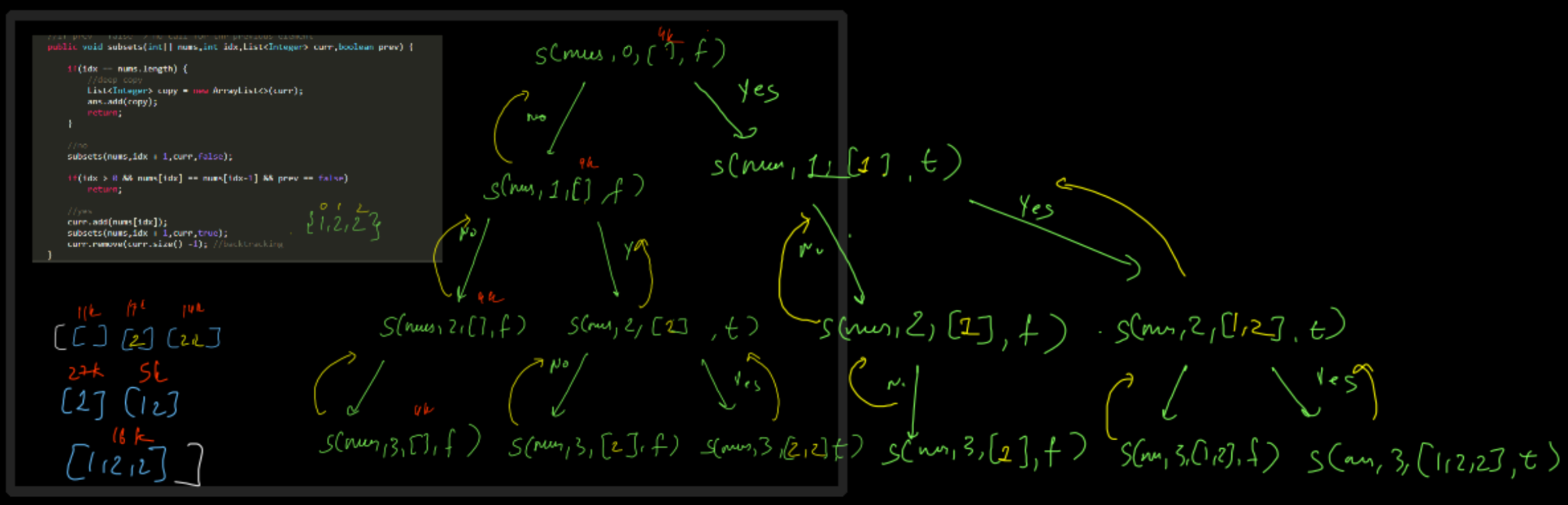
if you apply a 'no' call for an element, 'no' call should be applied if it repeats in future.

{2, 1, 3, 4, 2, 2, 8}

After sorting

$\{\overset{0}{1}, \overset{1}{2}, \overset{2}{2}\}$






```java
//if prev = false -> no call for the previous element
public void subsets(int[] nums,int idx,List<Integer> curr,boolean prev) {

    if(idx == nums.length) {
        //deep copy
        List<Integer> copy = new ArrayList<>(curr);
        ans.add(copy);
        return;
    }

    //no
    subsets(nums,idx + 1,curr,false);

    if(idx > 0 && nums[idx] == nums[idx-1] && prev == false)
        return;

    //yes
    curr.add(nums[idx]);
    subsets(nums,idx + 1,curr,true);
    curr.remove(curr.size() -1); //backtracking
}
```

Print Stair Paths

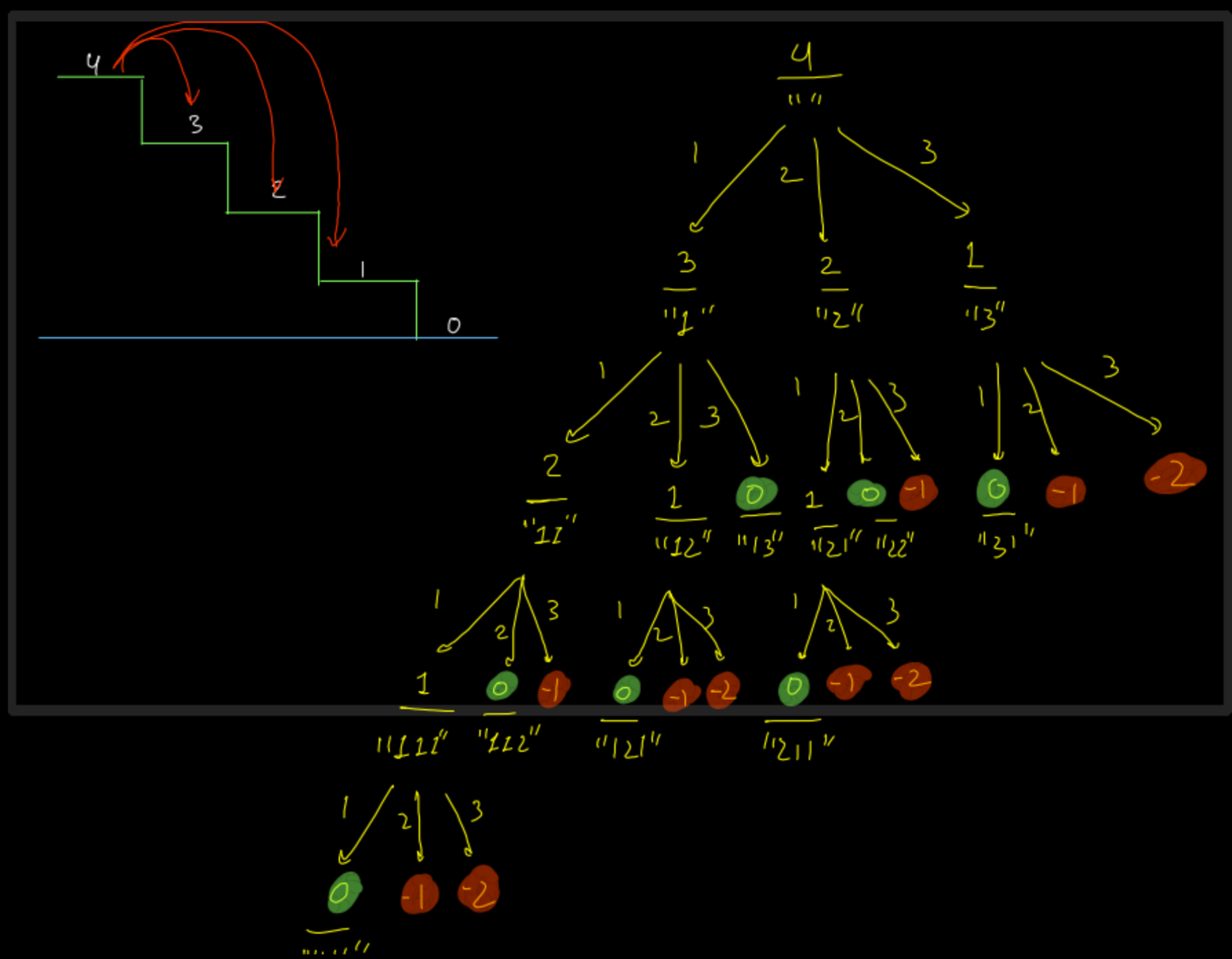
Easy

Prev Next

1. You are given a number n representing number of stairs in a staircase.
2. You are standing at the bottom of staircase. You are allowed to climb 1 step, 2 steps or 3 steps in one move.
3. Complete the body of printStairPaths function - without changing signature - to print the list of all paths that can be used to climb the staircase up.

Use sample input and output to take idea about output.

Note -> The online judge can't force you to write the function recursively but that is what the spirit of question is. Write recursive and not iterative logic. The purpose of the question is to aid learning recursion and not test you.



1 1 1 1

1 1 2

1 2 1

2 1 1

1 3

2 2

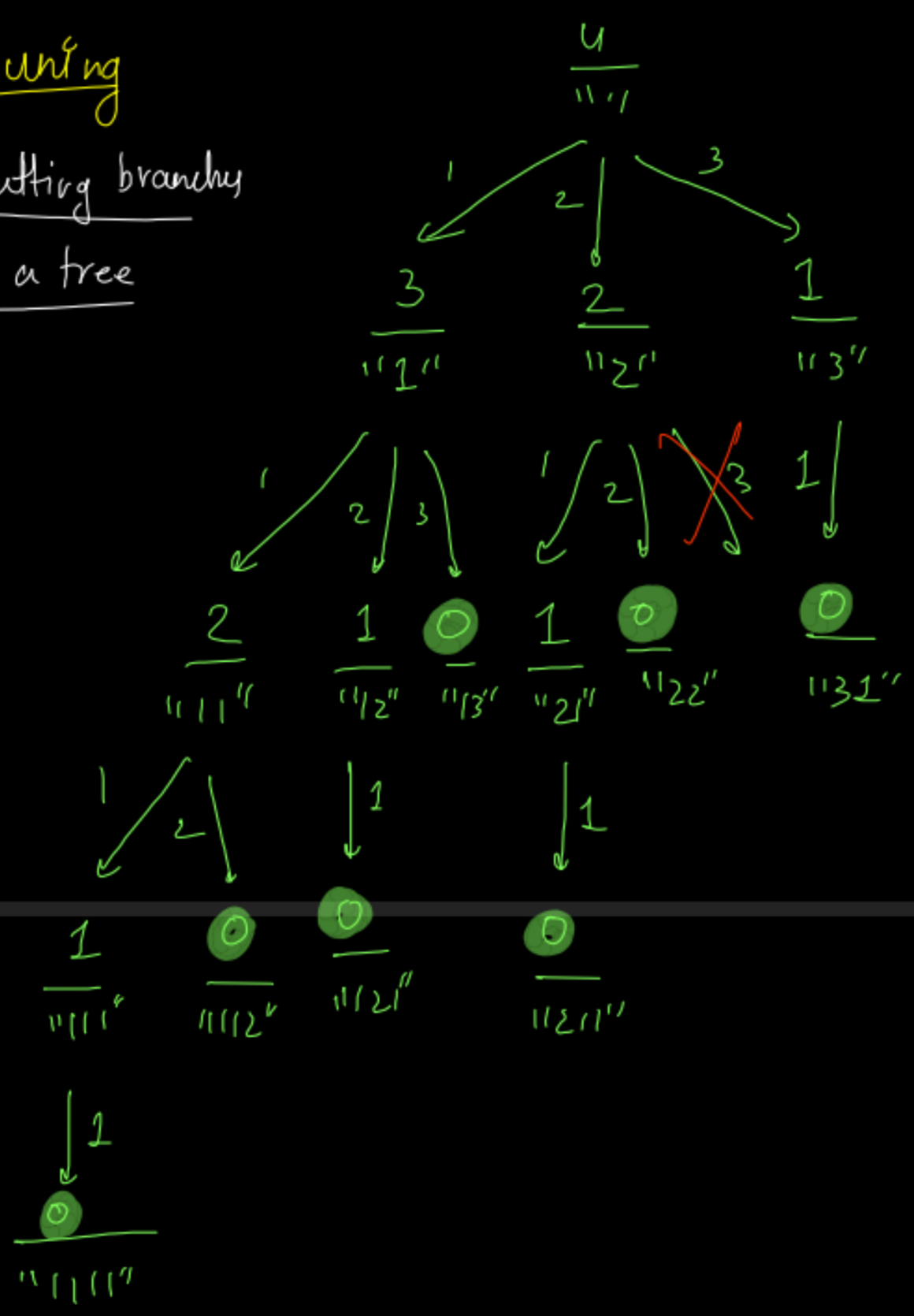
3 1

0.7 x

```java
public static void printStairPaths(int n, String path) {

    //negative base case -> we don't want this to happen
    if(n < 0) {
        return;
    }

    //positive base case -> we want this to happen
    if(n == 0) {
        System.out.println(path);
        return;
    }

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

Pruning

↳ cutting branches of a tree



```java
public static void printStairPaths(int n, String path) {

    //positive base case -> we want this to happen
    if(n == 0) {
        System.out.println(path);
        return;
    }

    //pruning
    if(n > 0) {
        printStairPaths(n-1,path + "1");
    }

    if(n > 1) {
        printStairPaths(n-2,path + "2");
    }

    if(n > 2) {
        printStairPaths(n-3,path + "3");
    }

}
```

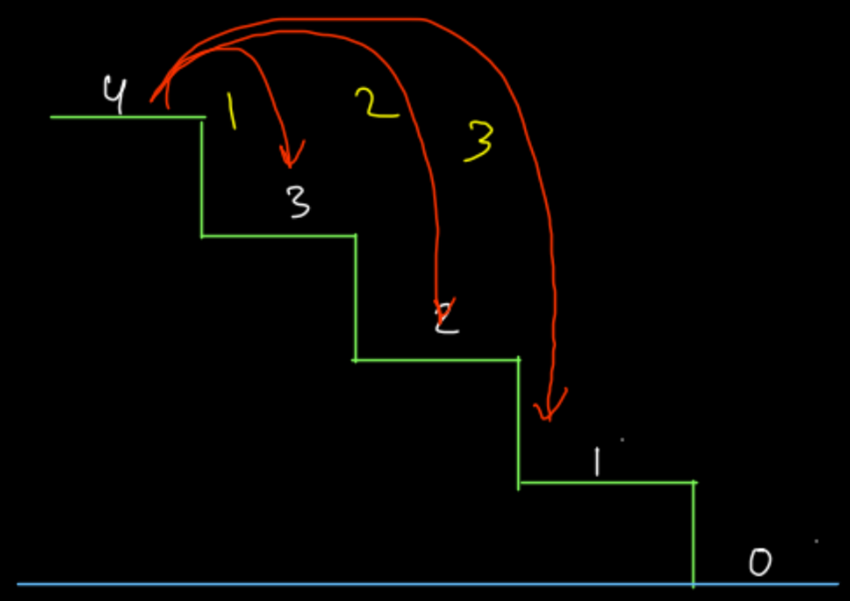
Lecture - 8

(Recursion & Backtracking)

→ Get Stair Paths (With -ve base case)

→ Get Stair Paths (With Pruning)

→ Matchstics to Square



Expectation: $s(n=4)$

↳ ["1111", "121", "13", "211", "22", "31", "112"]

faith: $3\,s(n-1) \rightarrow$ ["111", "12", "21", "3"]

$2\,s(n-2) \rightarrow$ ["11", "2"]

$1\,s(n-3) \rightarrow$ ["1"]

faith: $3s(n-1) \rightarrow$ ["1111", "112", "121", "13"]

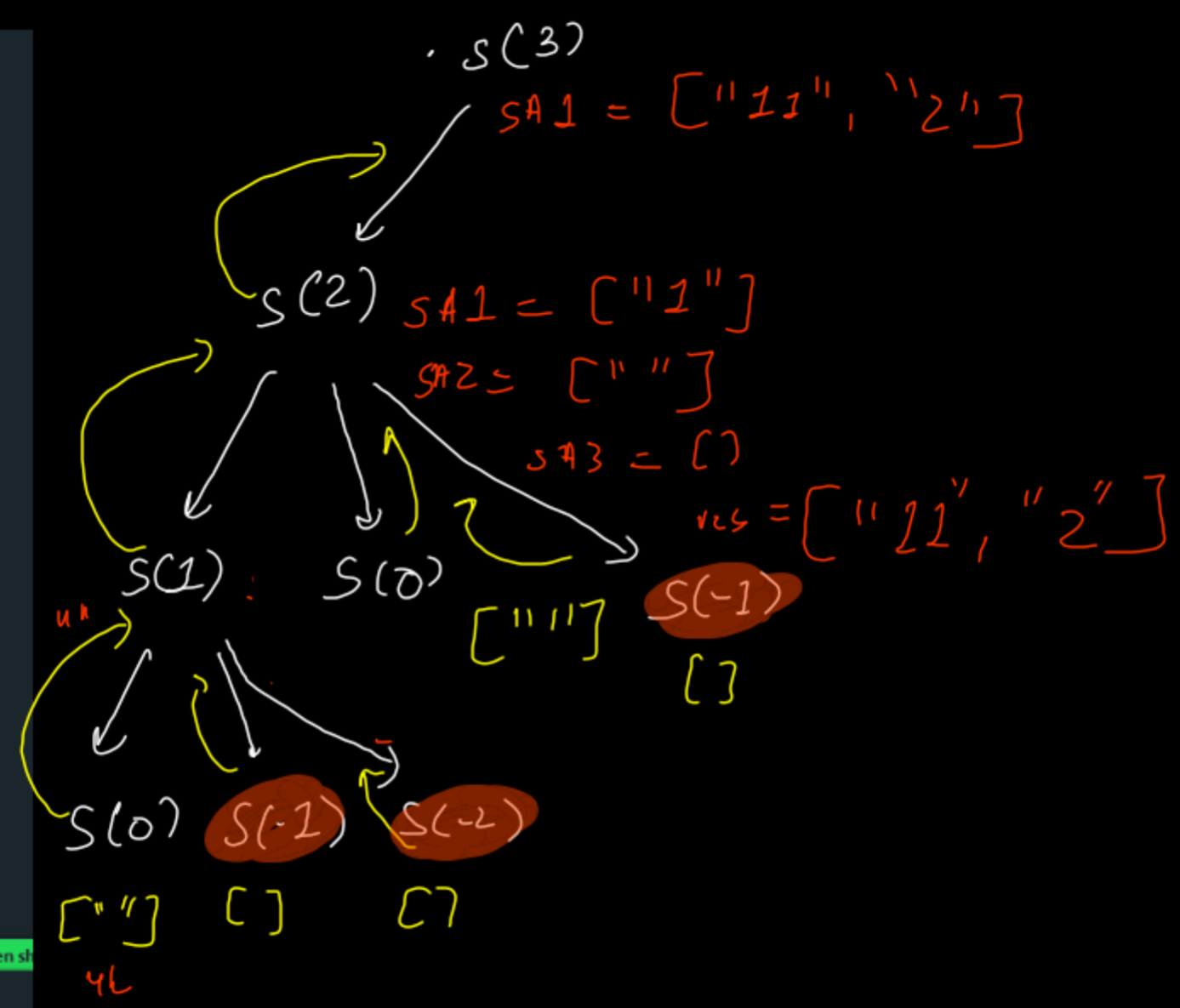
$2s(n-2) \rightarrow$ ["211", "22"]

$1s(n-3) \rightarrow$ ["31"]

and

["1111", "112", "121", "13", "211", "22", "31"]

```java
public static ArrayList<String> getStairPaths(int n) {

    //negative base case
    if(n < 0) {
        return new ArrayList<>();
    }

    //positive base
    if(n == 0) {
        ArrayList<String> res = new ArrayList<>();
        res.add("");
        return res;
    }

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

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

    for(String path : smallAns1) {
        res.add("1" + path);
    }

    for(String path : smallAns2) {
        res.add("2" + path);
    }

    for(String path : smallAns3) {
        res.add("3" + path);
    }

    return res;
}
```



```java
public static ArrayList<String> getStairPaths(int n) {


    //positive base
    if(n == 0) {
        ArrayList<String> res = new ArrayList<>();
        res.add("");
        return res;
    }

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

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

    if(n > 1) {
        ArrayList<String> smallAns2 = getStairPaths(n - 2);
        for(String path : smallAns2) {
            res.add("2" + path);
        }
    }

    if(n > 2) {
        ArrayList<String> smallAns3 = getStairPaths(n - 3);
        for(String path : smallAns3) {
            res.add("3" + path);
        }
    }


    return res;
}
```

You are screen sharing

With Pruning

473. Matchsticks to Square

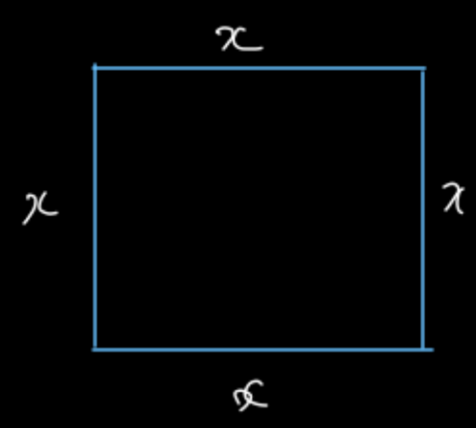
Medium 3246 255 Add to List Share

You are given an integer array `matchsticks` where `matchsticks[i]` is the length of the ith matchstick. You want to use **all the matchsticks** to make one square. You **should not break** any stick, but you can link them up, and each matchstick must be used **exactly one time**.

Return `true` if you can make this square and `false` otherwise.

Square

↳ 4 sides are equal



Perimeter = $4x$

{2, 2, 2, 6}

↳ P = 12

↳ divisible by 4

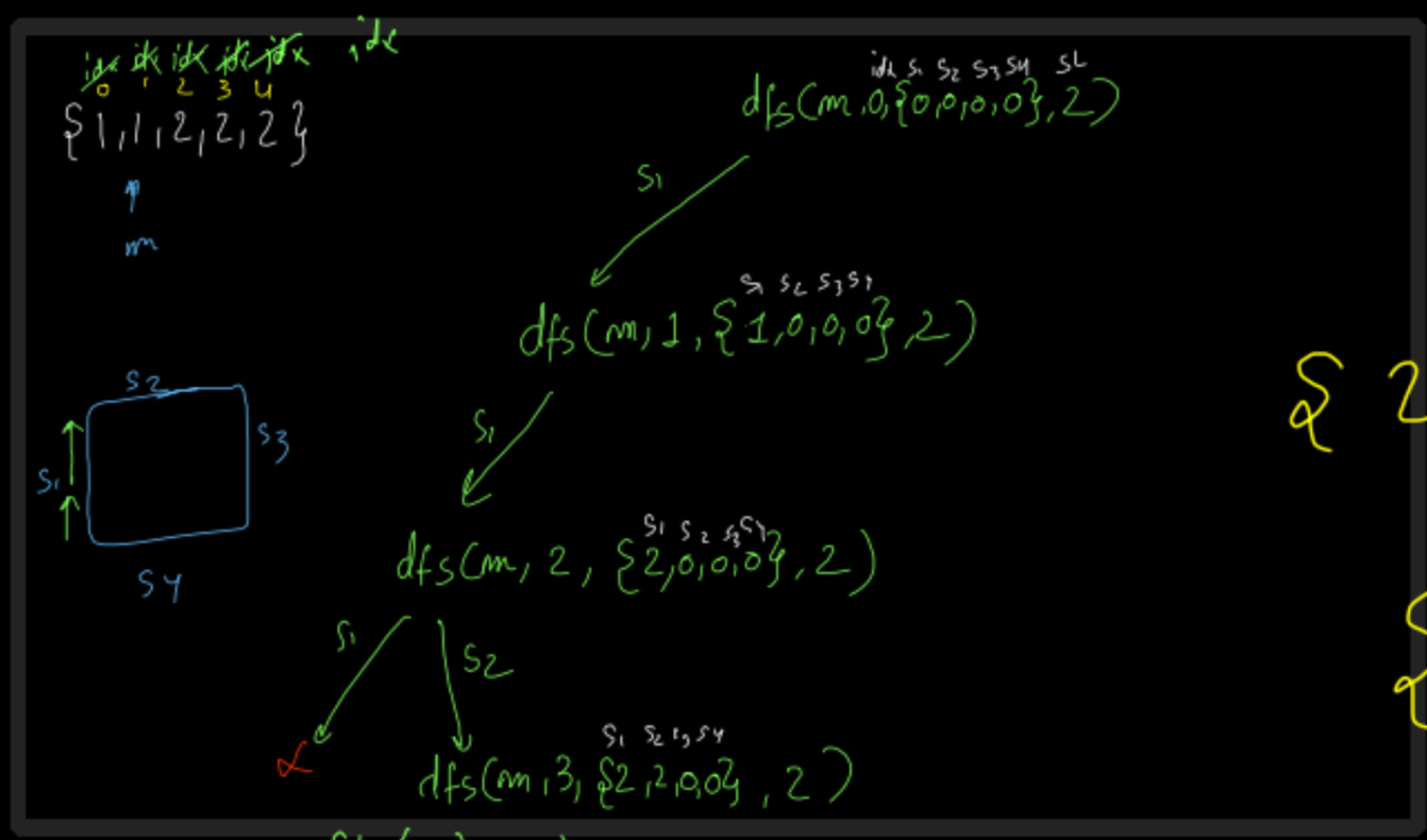
still cannot form a sq.

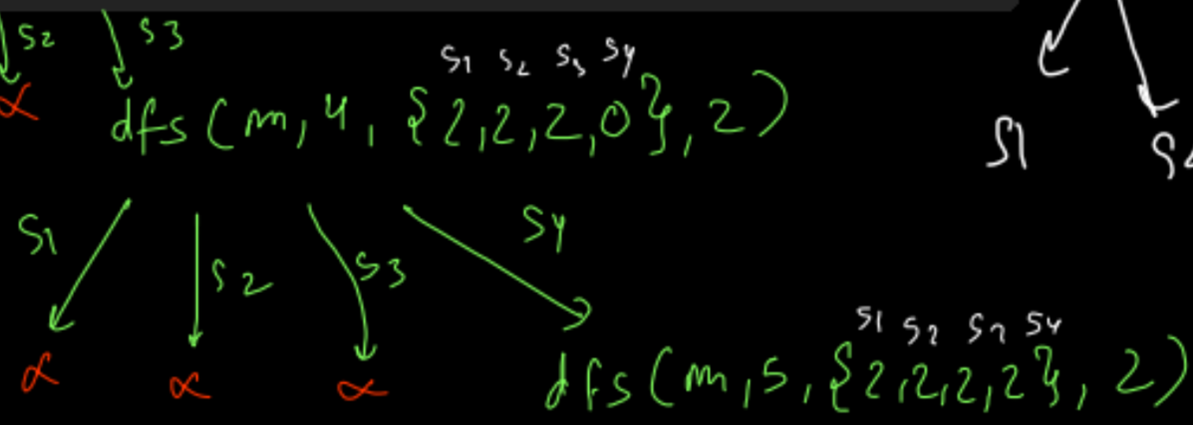
{1, 1, 2, 2, 2} → 1 + 1 + 2 + 2 + 2 = 8

If Perimeter is not divisible by 4, it is confirm that square cannot be formed. However, if it is divisible by 4, we can't guarantee anything

$\{2, 2, 2, 6\}$

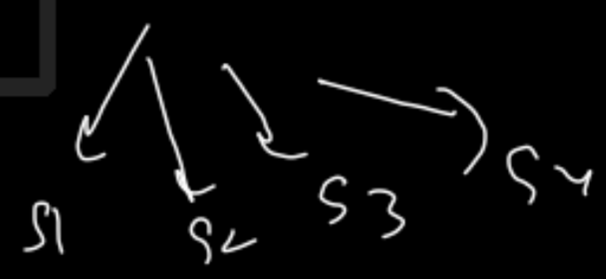
1, 1, 2, 3, 2





{2, 2, 2, 6}

{6, 2, 2, 2} → 3



```java
public boolean dfs(int[] matchsticks,int idx,int[] sides,int sideLength) {
    //base case
    if(idx == matchsticks.length) return true;

    for(int i=0;i<4;i++) {
        //pruning
        if(sides[i] + matchsticks[idx] <= sideLength) {
            //place the idxth matchstick on ith side
            sides[i] += matchsticks[idx];
            boolean canWePlaceRemSticks = dfs(matchsticks,idx + 1,sides,sideLength);
            if(canWePlaceRemSticks == true) return true;
            sides[i] -= matchsticks[idx];
        }
    }

    return false;
}
```

```java
public void reverse(int[] arr) {
    int lo = 0;
    int hi = arr.length -1;

    while(lo < hi) {
        int temp = arr[lo];
        arr[lo] = arr[hi];
        arr[hi] = temp;
        lo++;
        hi--;
    }
}
```

```java
public boolean makesquare(int[] matchsticks) {
    if(matchsticks == null || matchsticks.length == 0) return false;

    int peri = 0;
    for(int val : matchsticks) peri += val;

    if(peri % 4 != 0) return false;

    int sideLength = peri / 4;
    int[] sides = new int[4];

    //sort in descending order
    Arrays.sort(matchsticks);
    reverse(matchsticks);

    return dfs(matchsticks,0,sides,sideLength);
}
}
```

Lecture - 9

Recursion & Backtracking

Beautiful Arrangements
- Meth 1 (Generate All Perms)
- Meth 2

526. Beautiful Arrangement

Medium 2548 315 Add to List Share

Suppose you have `n` integers labeled `1` through `n`. A permutation of those `n` integers `perm` (**1-indexed**) is considered a **beautiful arrangement** if for every `i` (`1 <= i <= n`), **either** of the following is true:

- `perm[i]` is divisible by `i`.
- `i` is divisible by `perm[i]`.

Given an integer `n`, return *the **number** of the **beautiful arrangements** that you can construct.*

$N = 3$

1, 2, 3

1 2 3
{1, 2, 3}

1 2 3
{1, 3, 2}

1 2 3
{2, 1, 3}

1 2 3
{2, 3, 1}

1 2 3
{3, 1, 2}

1 2 3
{3, 2, 1}

perm[i] % i == 0

or

i % perm[i] == 0

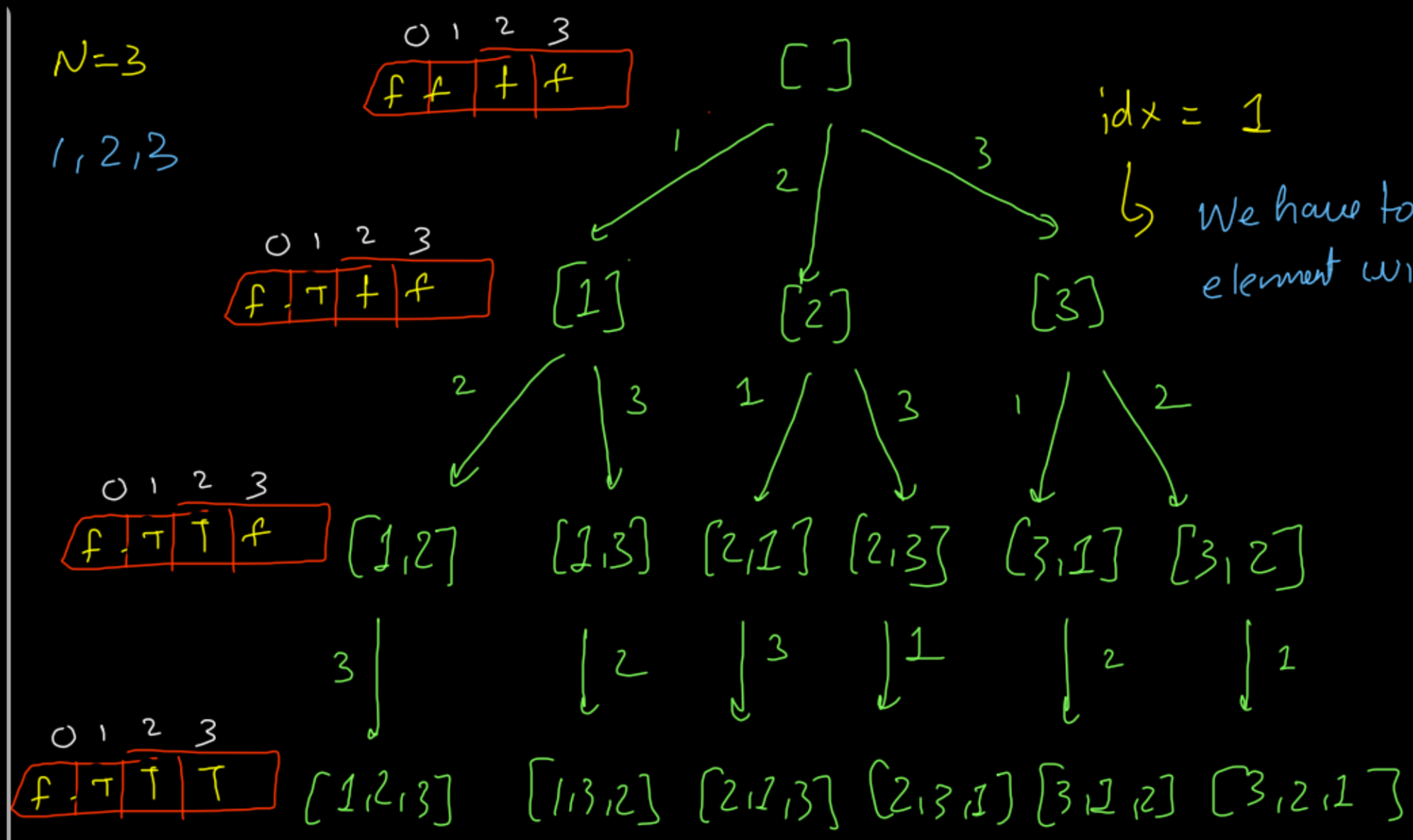
1 2 3
{2, 1, 3} ✓

1 2 3
{1, 2, 3} ✓

1 2 3
{1, 3, 2} ✗

N=3

1, 2, 3



idx = 1

↳ We have to decide which element will occupy this idx

N=3

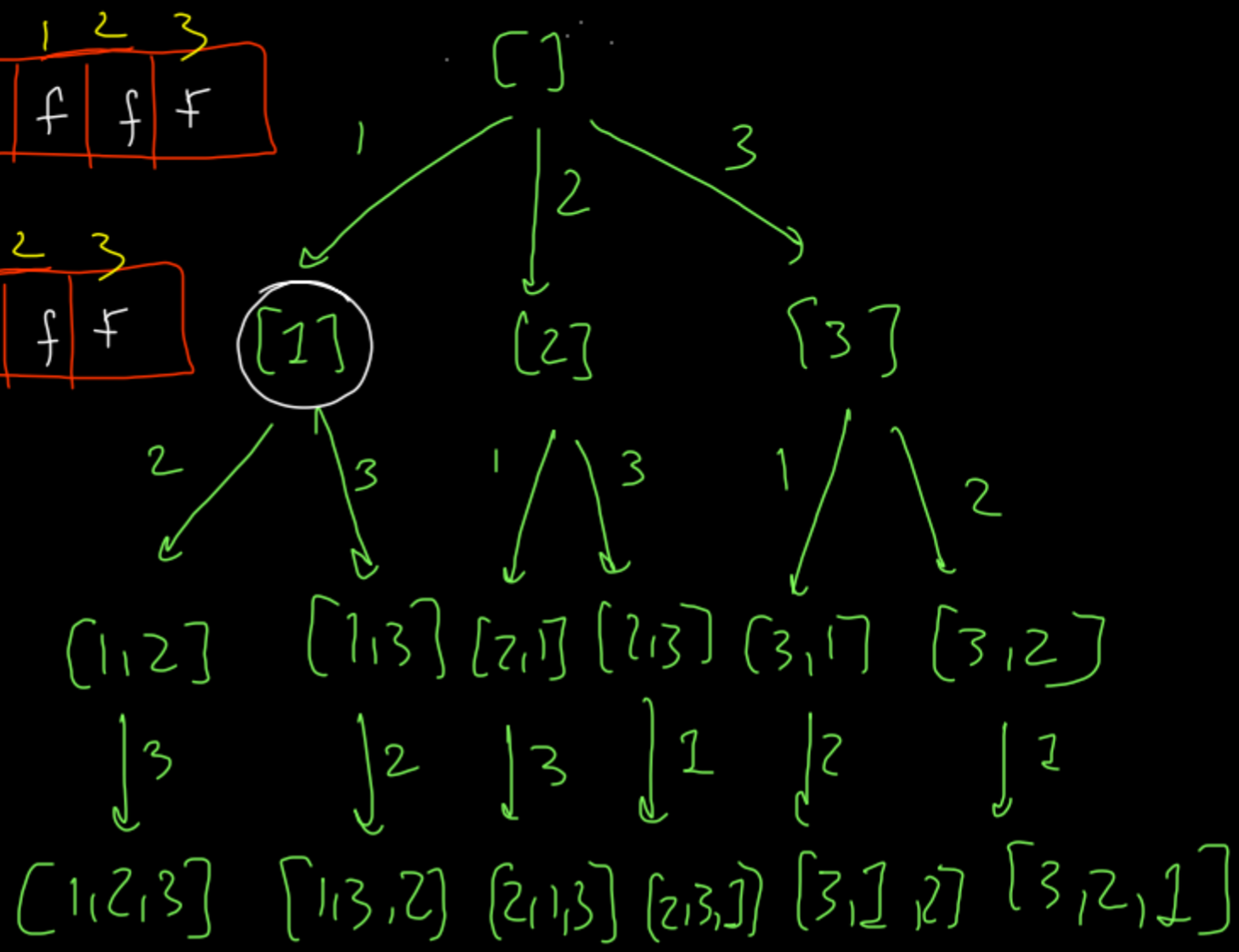
1,2,3

0	1	2	3
F	F	F	F

0	1	2	3
F	~~F~~	F	F
	T		

0	1	2	3
F	T	~~F~~	F
		T	

0	1	2	3
F	T	T	~~F~~
			T



1

2

3

i = ~~1~~ 2

0	1	2
3	2	1
1	2	3

(i+1)

```java
public void permutations(int n,List<Integer> curr,boolean[] vis) {

    if(curr.size() == n) {

        for(int i=0;i<n;i++) {
            if(curr.get(i) % (i+1) != 0 && (i+1) % curr.get(i) != 0)
                return;
        }

        count++;
        return;
    }

    for(int i=1;i<=n;i++) {
        if(vis[i] == false) {
            curr.add(i);
            vis[i] = true;
            permutations(n,curr,vis);
            //backtracking
            curr.remove(curr.size() -1);
            vis[i] = false;
        }
    }
}
```

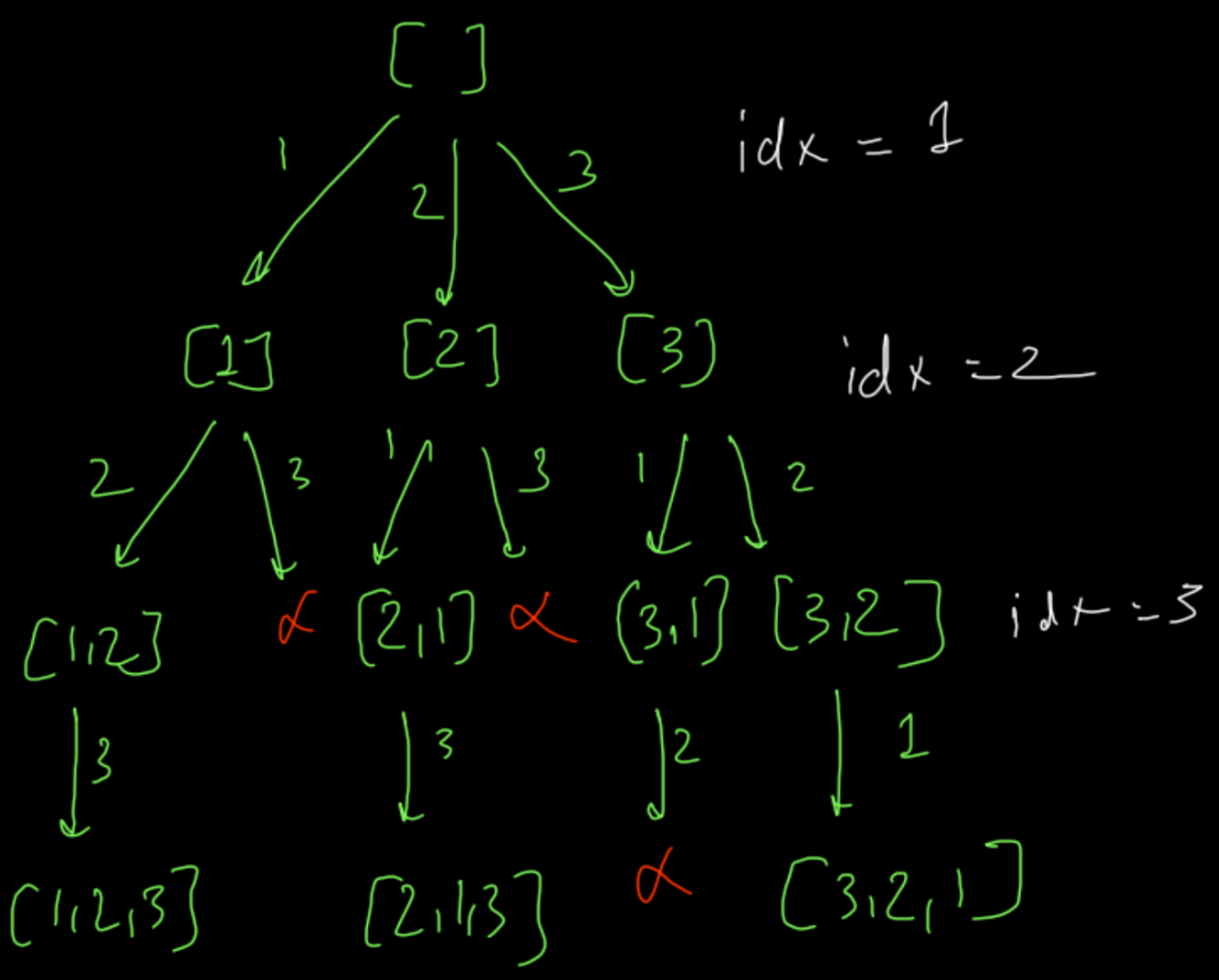
//TLE

```java
public int countArrangement(int n) {
    count = 0;
    permutations(n,new ArrayList<>(),new boolean[n+1]);
    return count;
}
```

$N=3$

$1, 2, 3$

$idx = 2$



```java
class Solution {

    int count;

    public void permutations(int n,int idx,boolean[] vis) {

        if(idx > n) {
            count++;
            return;
        }

        for(int num=1;num<=n;num++) {
            if(vis[num] == false && (num % idx == 0 || idx % num == 0)) {
                vis[num] = true;
                permutations(n,idx + 1,vis);
                vis[num] = false;
            }
        }
    }

    public int countArrangement(int n) {
        count = 0;
        permutations(n,1,new boolean[n+1]);
        return count;
    }
}
```