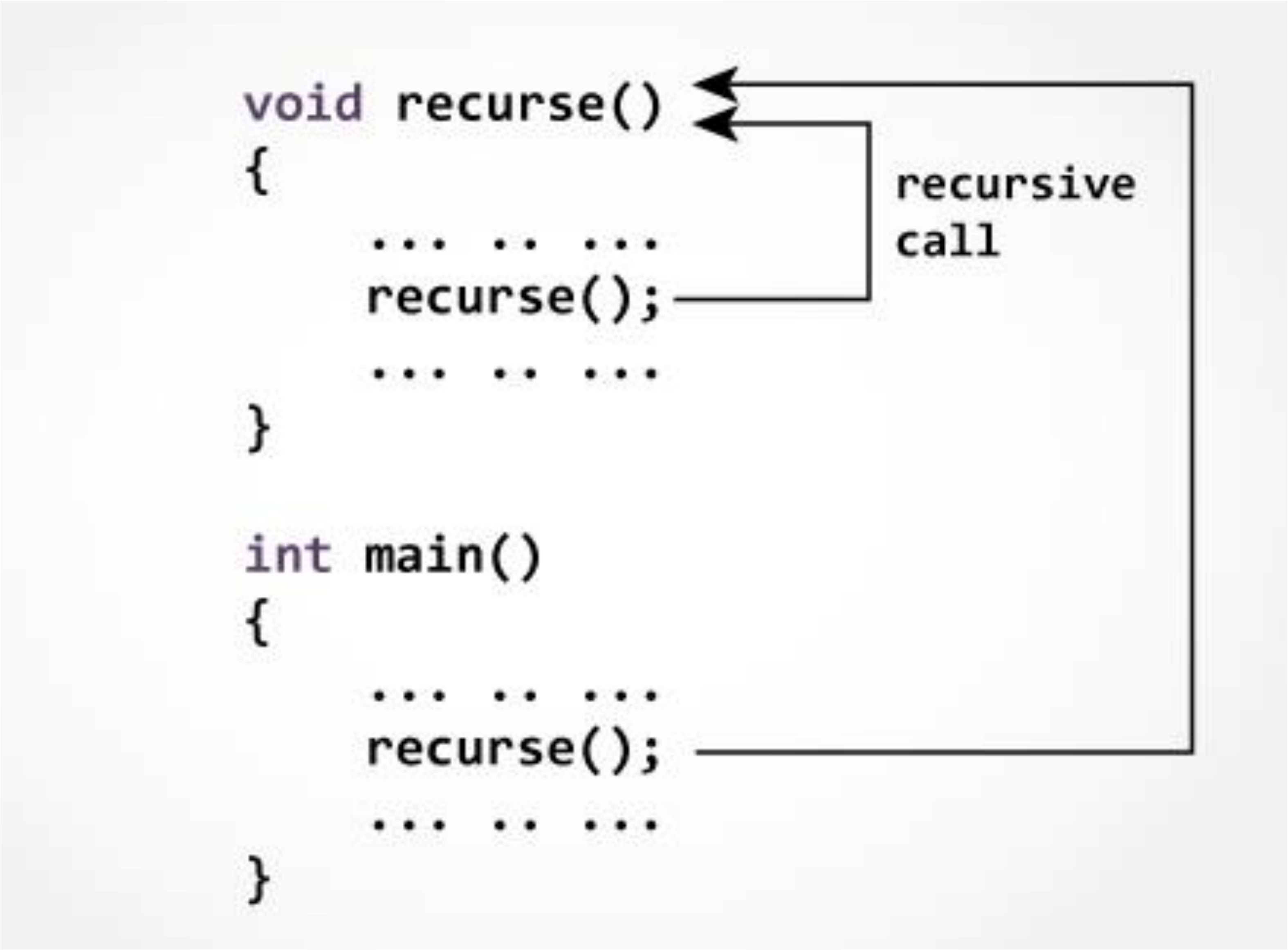
Topics

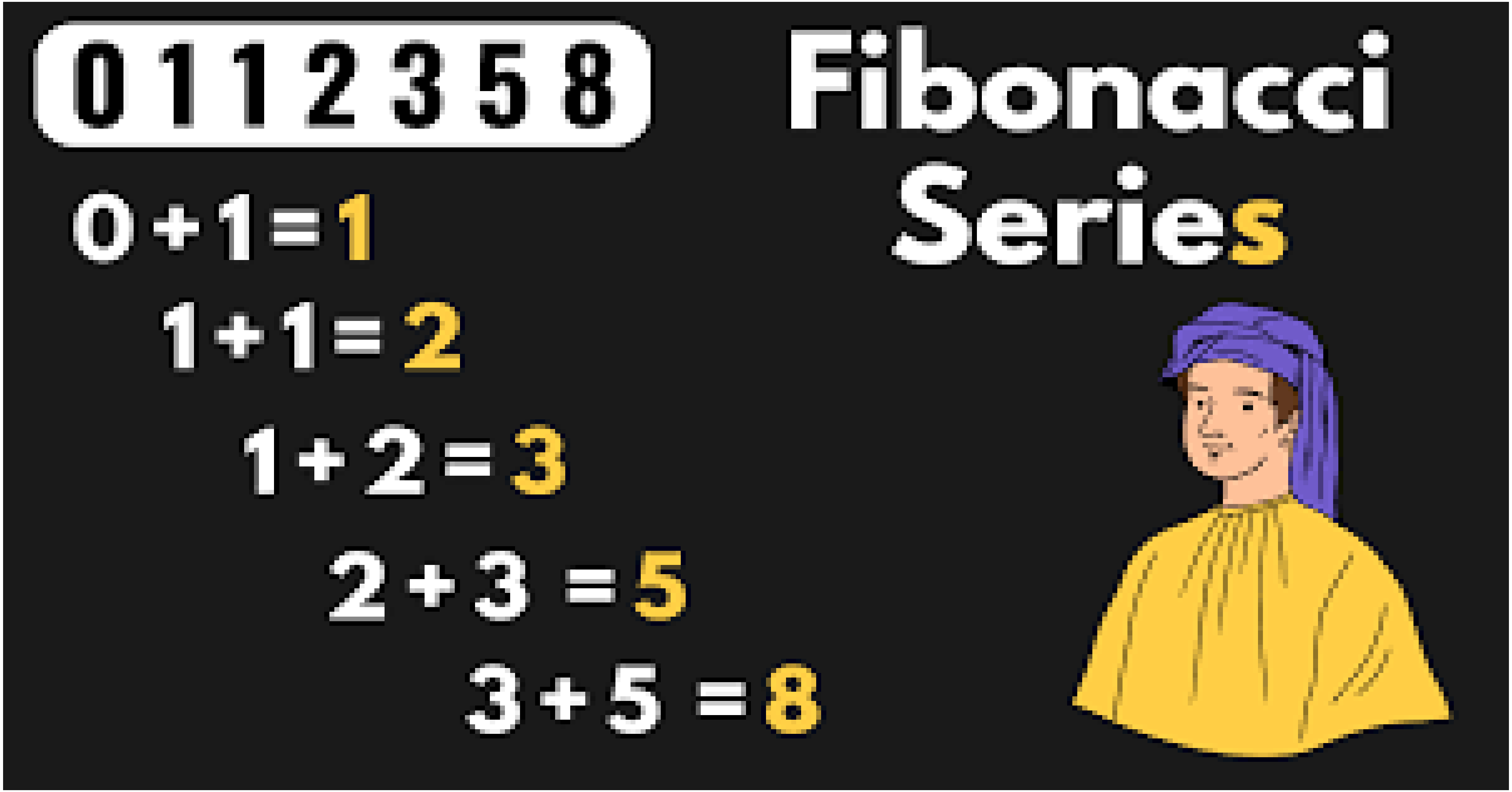
1. Recursive definitions and Processes
2. Writing Recursive Programs
3. Efficiency in Recursion
4. Towers of Hanoi problem.

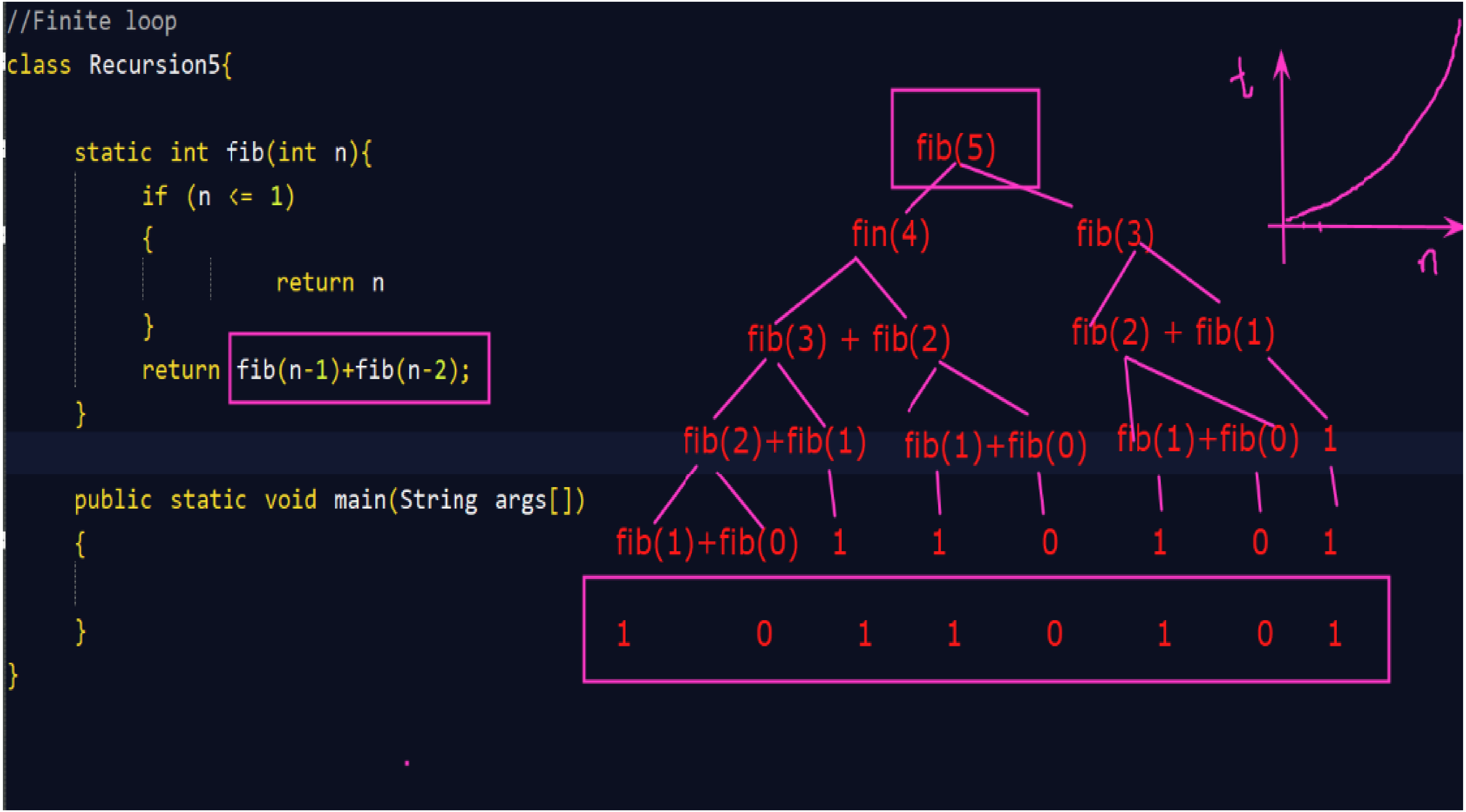
How does Recursion works?



```
void recurse()
{
    ... .. ...
    recurse();
    ... .. ...
}

int main()
{
    ... .. ...
    recurse();
    ... .. ...
}
```


0 1 1 2 3 5 8
Fibonacci Series
0+1=1
1+1=2
1+2=3
2+3 =5
3+5 =8

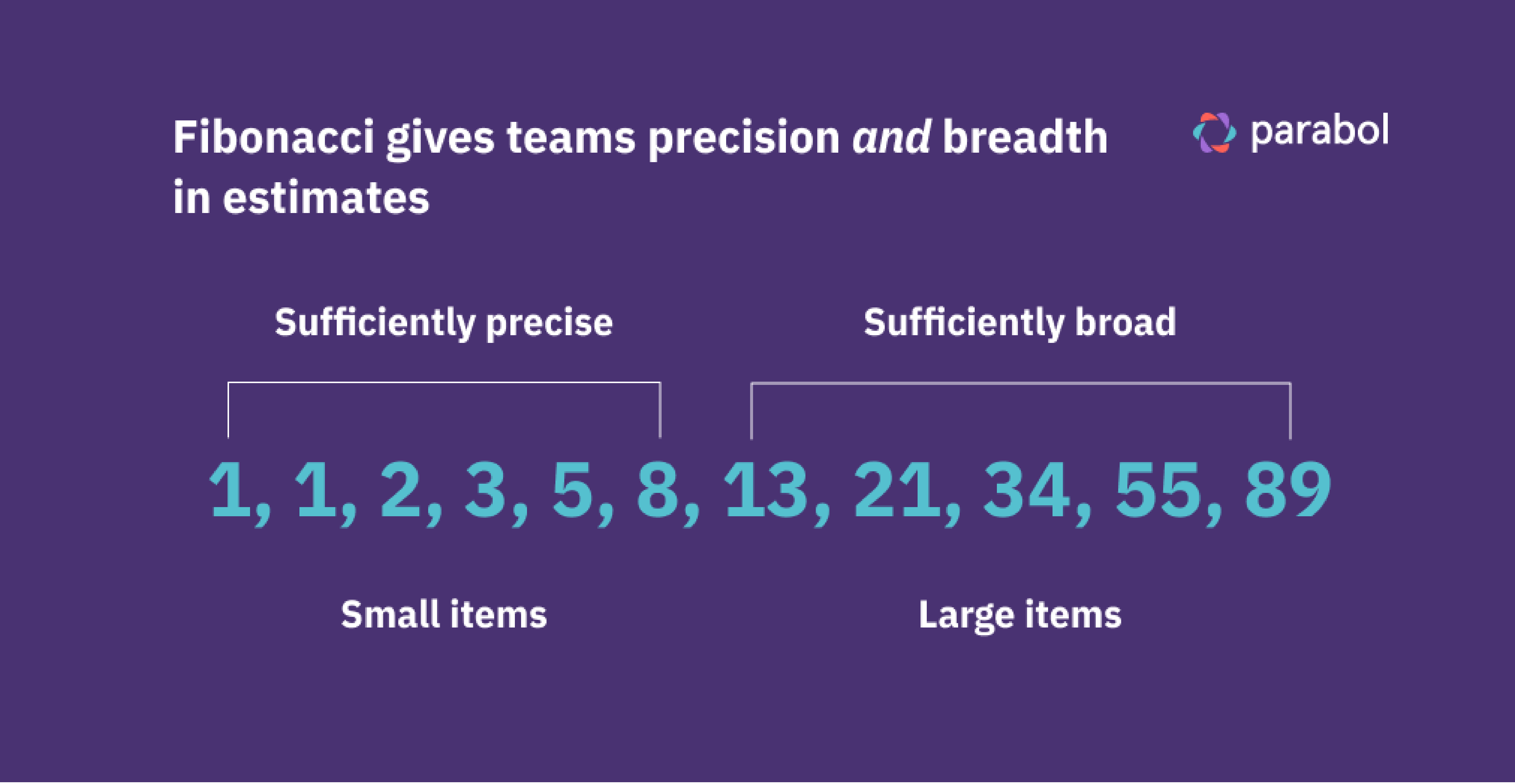


```
//Finite loop
class Recursion5{

    static int fib(int n){
        if (n <= 1)
        {
                return n
        }
        return fib(n-1)+fib(n-2);
    }

    public static void main(String args[])
    {

    }
}
```



Fibonacci gives teams precision *and* breadth in estimates

parabol

Sufficiently precise

Sufficiently broad

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89

Small items

Large items

Why Algorithms?

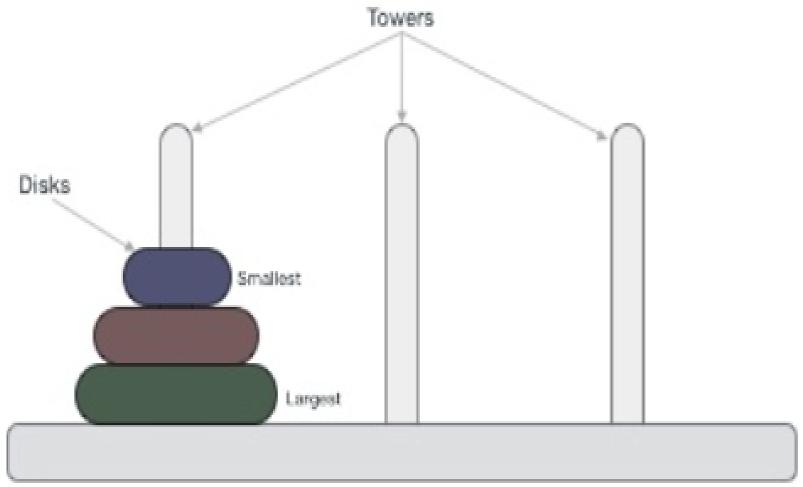
- Fibonacci numbers
  - Compute first N Fibonacci numbers using iteration.
  - … using recursion.
- Write the code.
- Try for N=5, 10, 20, 50, 100
- What do you see? Why does this happen?

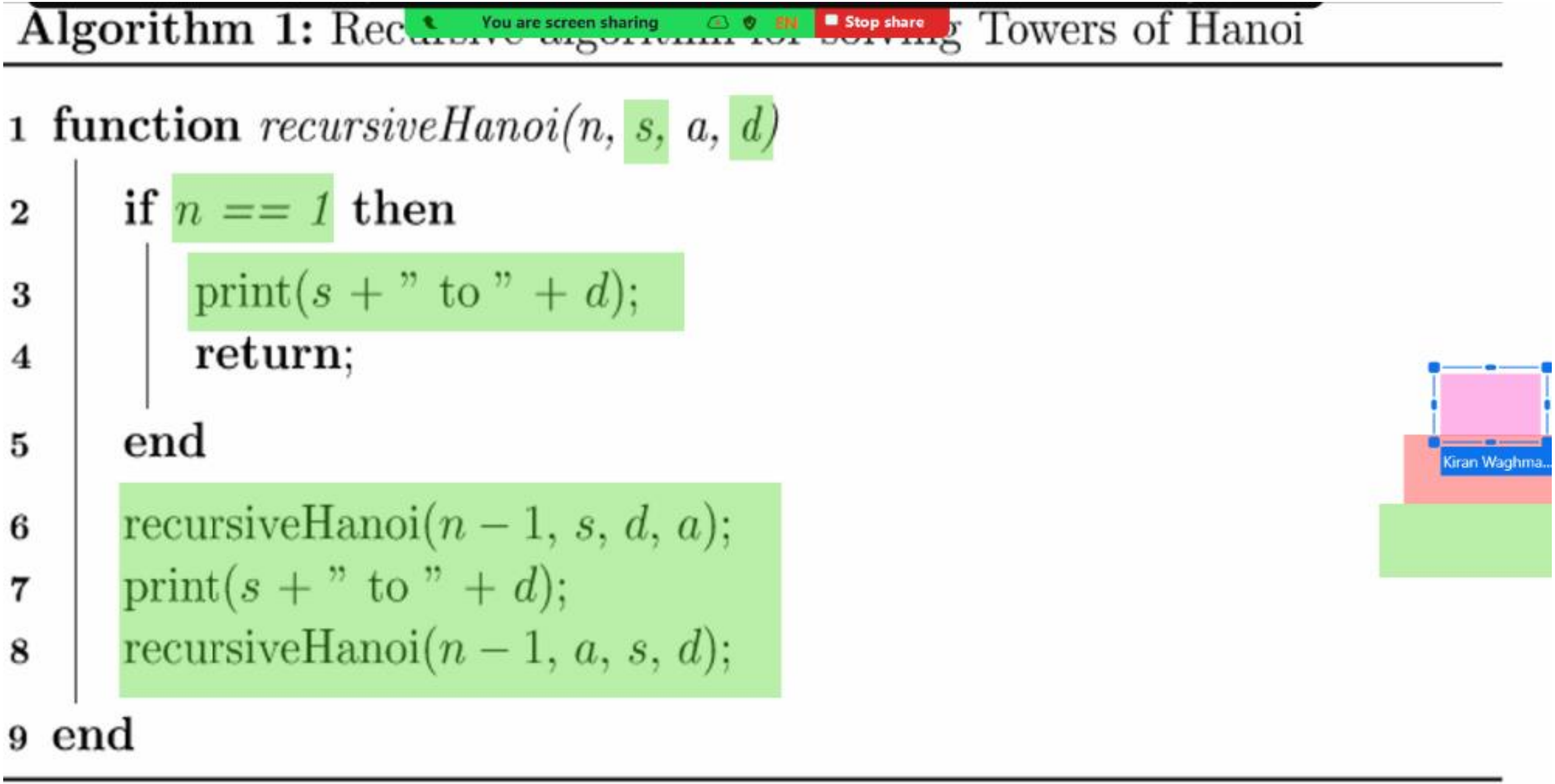
What is Tower of Hanoi?

- **A mathematical puzzle consisting of three towers and more than one ring is known as Tower of Hanoi.**

- **Tower of Hanoi**

- **The rings are of different sizes and are stacked in ascending order, i.e., the smaller one sits over the larger one. In some of the puzzles, the number of rings may increase, but the count of the tower remains the same.**





Algorithm 1: Recursive algorithm for solving Towers of Hanoi

```
function recursiveHanoi($n$, $s$, $a$, $d$)
    if $n == 1$ then
        print($s$ + " to " + $d$);
        return;
    end
    recursiveHanoi($n - 1$, $s$, $d$, $a$);
    print($s$ + " to " + $d$);
    recursiveHanoi($n - 1$, $a$, $s$, $d$);
end
```

What are the rules to be followed by Tower of Hanoi?

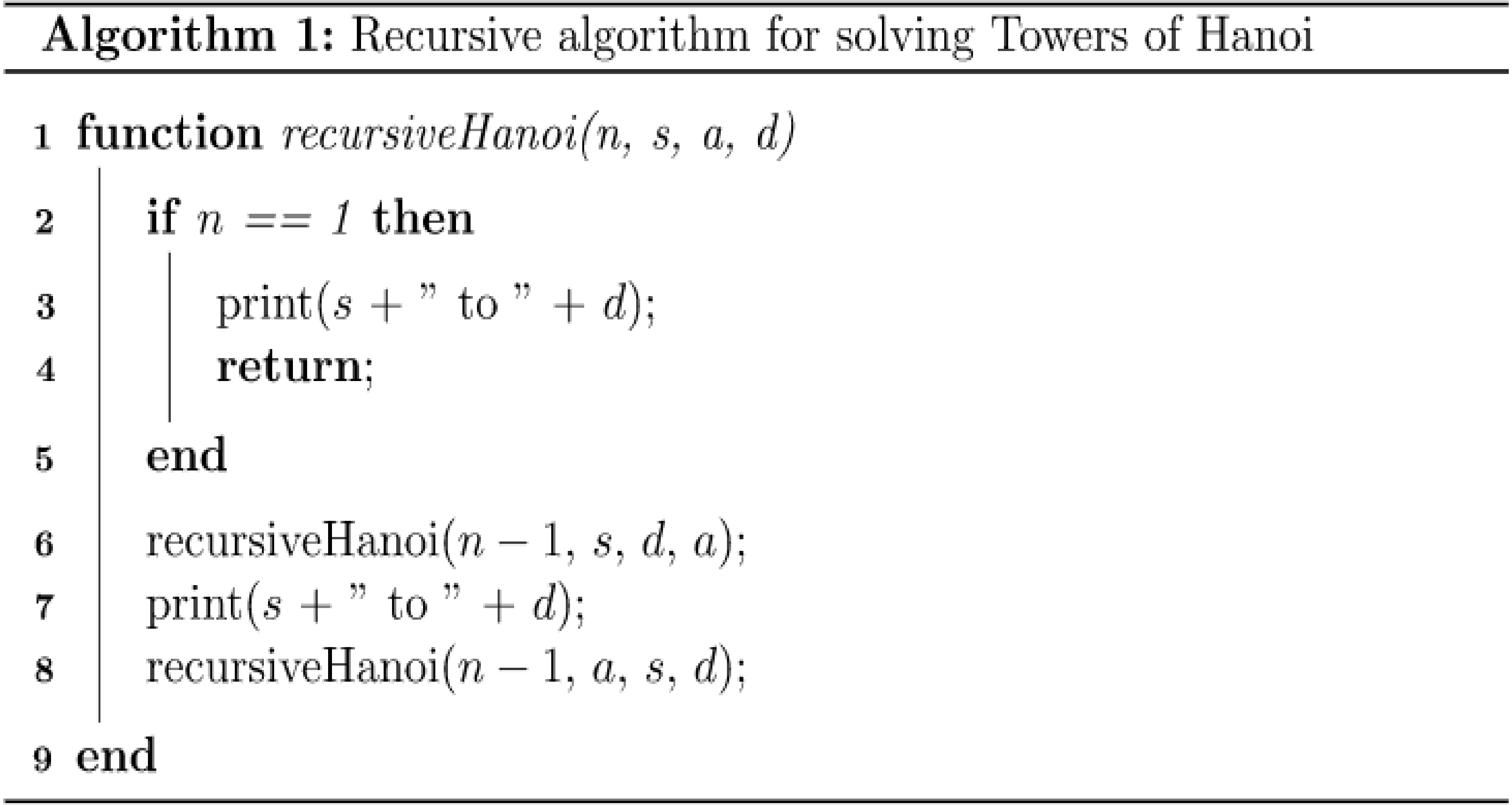
- **The Tower of Hanoi puzzle is solved by moving all the disks to another tower by not violating the sequence of the arrangements.**

The rules to be followed by the Tower of Hanoi are -

1. Only one disk can be moved among the towers at any given time.
2. Only the "top" disk can be removed.
3. No large disk can sit over a small disk.

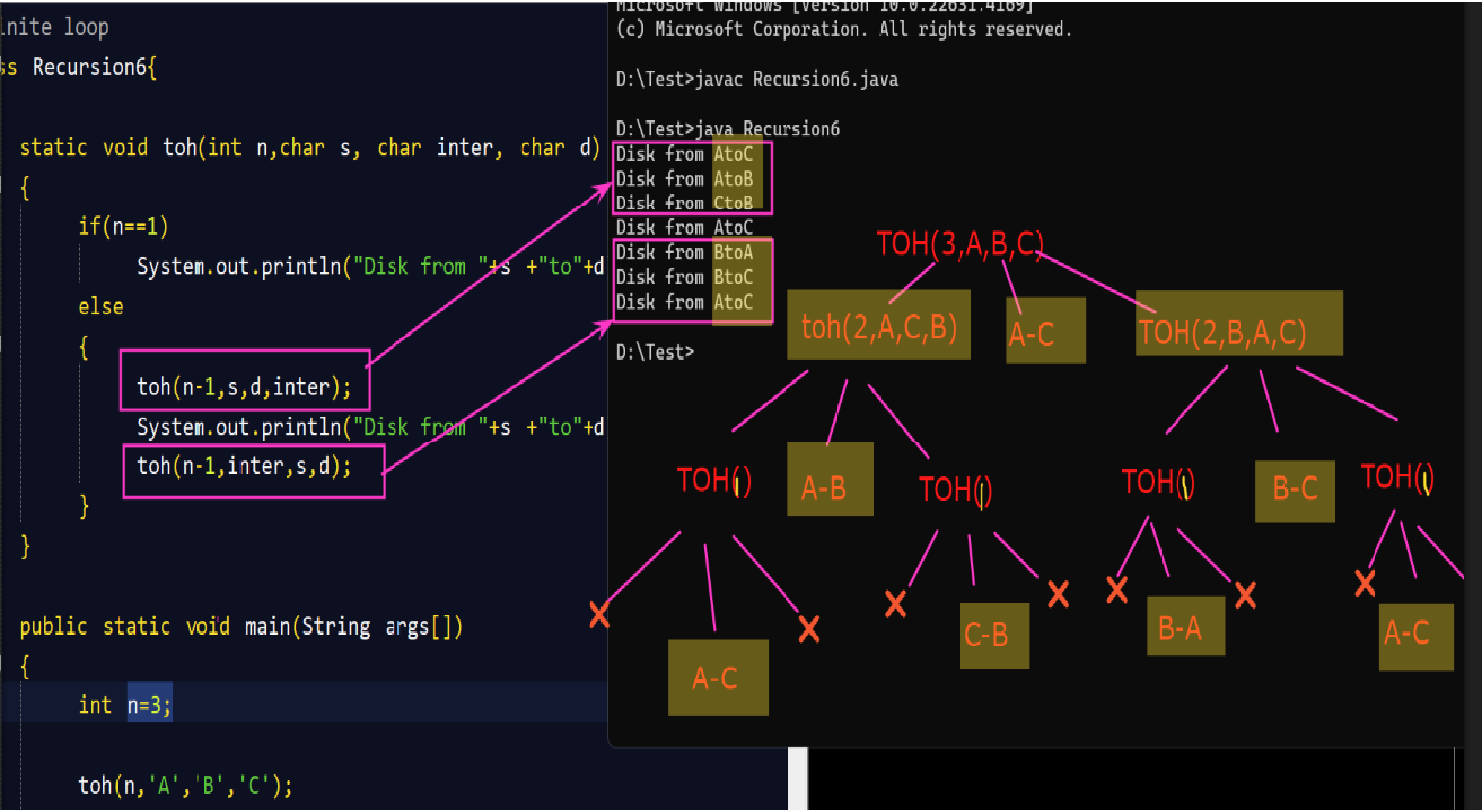
- **In pseudocode, this looks like the following.**
- **At the top level, we'll call MoveTower with**
  - disk=5, source=A, dest=B, and spare=C.
- **FUNCTION MoveTower(disk, source, dest, spare):**

```
IF disk == 0, THEN:
    move disk from source to dest
ELSE:
    MoveTower(disk - 1, source, spare, dest)   // Step 1 above
    move disk from source to dest             // Step 2 above
    MoveTower(disk - 1, spare, dest, source)   // Step 3 above
END IF
```



Algorithm 1: Recursive algorithm for solving Towers of Hanoi

```
function recursiveHanoi(n, s, a, d)
    if $n == 1$ then
        print($s$ + " to " + $d$);
        return;
    end
    recursiveHanoi($n - 1$, $s$, $d$, $a$);
    print($s$ + " to " + $d$);
    recursiveHanoi($n - 1$, $a$, $s$, $d$);
end
```



```
inite loop
ss Recursion6{

  static void toh(int n,char s, char inter, char d)
  {
      if(n==1)
          System.out.println("Disk from "+s +"to"+d
      else
      {
          toh(n-1,s,d,inter);
          System.out.println("Disk from "+s +"to"+d
          toh(n-1,inter,s,d);
      }
  }

  public static void main(String args[])
  {
      int n=3;

      toh(n,'A','B','C');
```

```
Microsoft Windows [Version 10.0.22631.4169]
(c) Microsoft Corporation. All rights reserved.

D:\Test>javac Recursion6.java

D:\Test>java Recursion6
Disk from AtoC
Disk from AtoB
Disk from CtoB
Disk from AtoC
Disk from BtoA
Disk from BtoC
Disk from AtoC

D:\Test>
```

Home Work

- Implement Tower of Hanoi Program
- No of Disk=3
- No of Disk=5
- No of Disk=n

Assignment 1

1. Print a series of numbers with recursive Java methods
2. Sum a series of numbers with Java recursion
3. Calculate a factorial in Java with recursion
4. Print the Fibonacci series with Java and recursion
5. A recursive Java palindrome checker

Outline of a Recursive Function

```
if (answer is known)
    provide the answer & exit
else
    call same function with
    a smaller version
    of the same problem
```

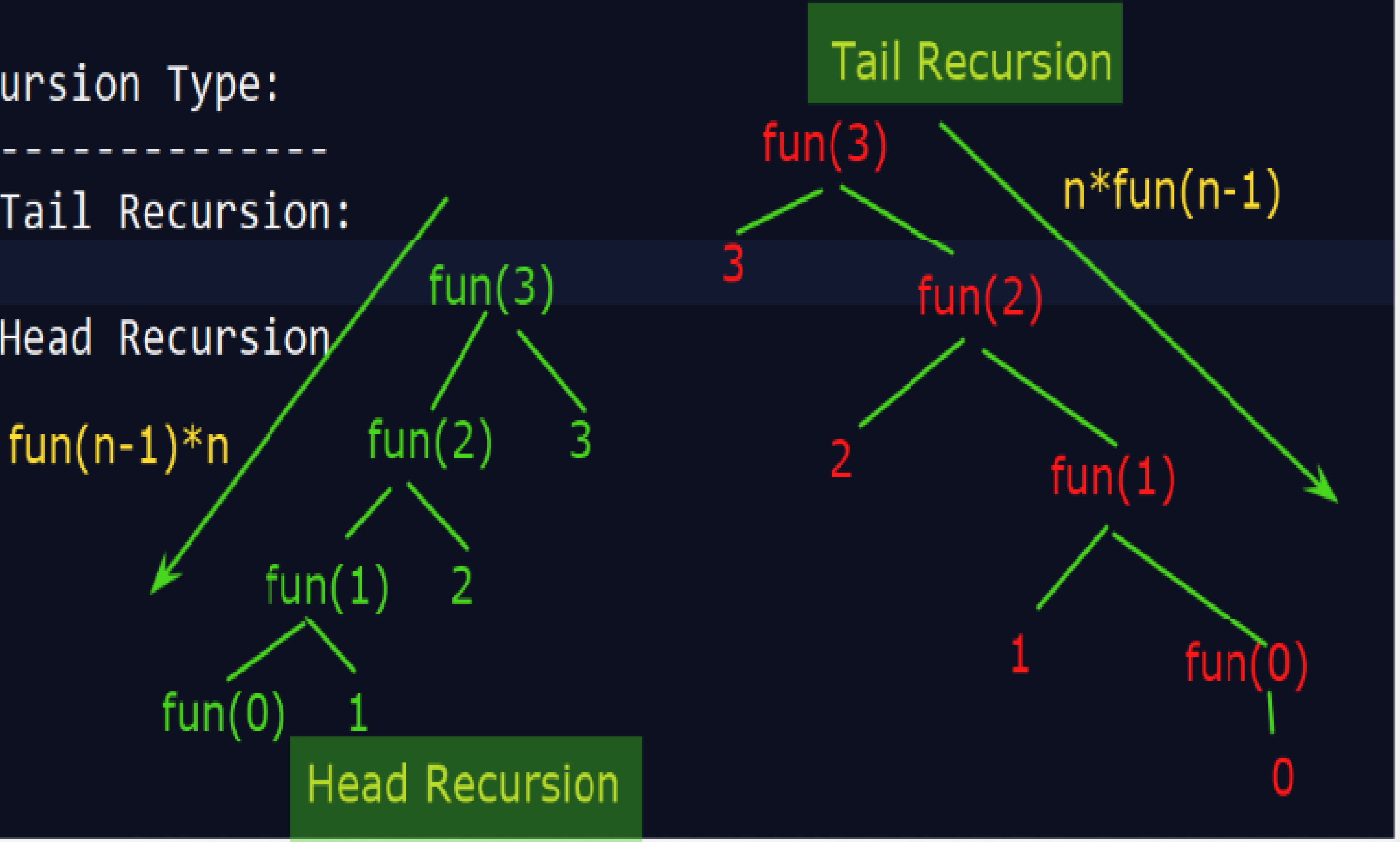
base case

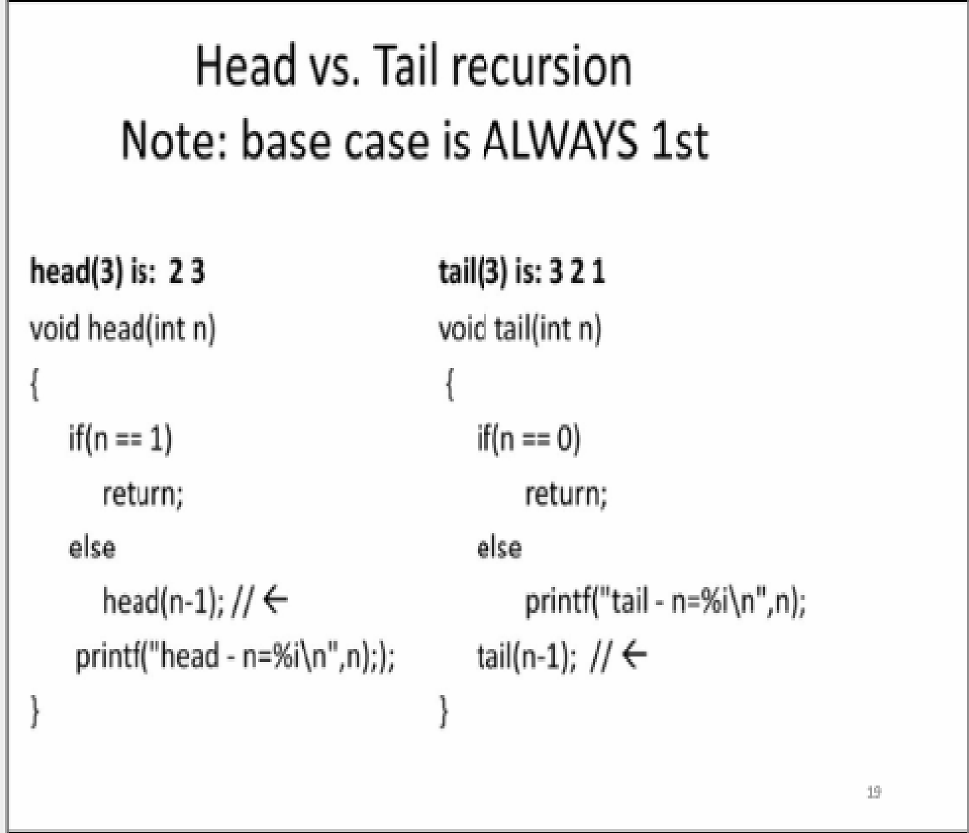
recursive case

```
xamples of Recursion:
   -Tower of Hanoi
   -Factorial
   -Fibonacci series
   -GCD
   -Printing all permutations of the given string
   -Generate all strings of n bits of binary number

Ecursion Type:
----------------
. Tail Recursion:

. Head Recursion
```





Head vs. Tail recursion

Note: base case is ALWAYS 1st

head(3) is: 2 3

```
void head(int n)
{
   if(n == 1)
      return;
   else
      head(n-1); // ←
   printf("head - n=%i\n",n););
}
```

tail(3) is: 3 2 1

```
void tail(int n)
{
   if(n == 0)
      return;
   else
      printf("tail - n=%i\n",n);
   tail(n-1);  // ←
}
```

Head vs. Tail recursion
Note: base case is ALWAYS 1st

head(3) is: 2 3

```
void head(int n)
{
    if(n == 1)
        return;
    else
        head(n-1); // ←
     printf("head - n=%i\n",n););
}
```

tail(3) is: 3 2 1

```
void tail(int n)
 {
    if(n == 0)
        return;
    else
        printf("tail - n=%i\n",n);
    tail(n-1);  // ←
}
```

Problem 1

Recursive program to find the Sum of the series 1 – 1/2 + 1/3 – 1/4 … 1/N
Given a positive integer N, the task is to find the sum of the series 1 – (1/2) + (1/3) – (1/4) +…. (1/N) using recursion.

Examples:

Input: N = 3
Output: 0.8333333333333333
Explanation:
1 – (1/2) + (1/3) = 0.8333333333333333

Input: N = 4
Output: 0.5833333333333333
Explanation:
1- (1/2) + (1/3) – (1/4) = 0.5833333333333333

Problem 2

Recursive Program to print multiplication table of a number
Given a number N, the task is to print its multiplication table using recursion.
Examples

Input: N = 5
Output:

```
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
```

Input: N = 8
Output:

```
8 * 1 = 8
8 * 2 = 16
8 * 3 = 24
8 * 4 = 32
8 * 5 = 40
8 * 6 = 48
8 * 7 = 56
8 * 8 = 64
8 * 9 = 72
8 * 10 = 80
```

Problem 3

Recursive program to print formula for GCD of n integers

Given a function gcd(a, b) to find GCD (Greatest Common Divisor) of two number. It is also known that GCD of three elements can be found by gcd(a, gcd(b, c)), similarly for four element it can find the GCD by gcd(a, gcd(b, gcd(c, d))). Given a positive integer n. The task is to print the formula to find the GCD of n integer using given gcd() function.
Examples:

Input : n = 3
Output : gcd(int, gcd(int, int))

Input : n = 5
Output : gcd(int, gcd(int, gcd(int, gcd(int, int))))

Problem 4

Java Program to Reverse a Sentence Using Recursion

A sentence is a sequence of characters separated by some delimiter. This sequence of characters starts at the 0th index and the last index is at len(string)-1. By reversing the string, we interchange the characters starting at 0th index and place them from the end. The first character becomes the last, the second becomes the second last, and so on.

Example:

Input : CDACMumbai
Output: iabmuMCADC

Input : Alice
Output: ecilA
Approach:

Check if the string is empty or not, return null if String is empty.
If the string is empty then return the null string.
Else return the concatenation of sub-string part of the string from index 1 to string length with the first character of a string. e.g. return substring(1)+str.charAt(0); which is for string “Mayur” return will be “ayur” + “M”.

Ackermann's function

$A(0, n) = n + 1$

$A(m, 1) = A(m+1, 0)$

$A(m+1, n+1) = A(m, A(m+1, n))$

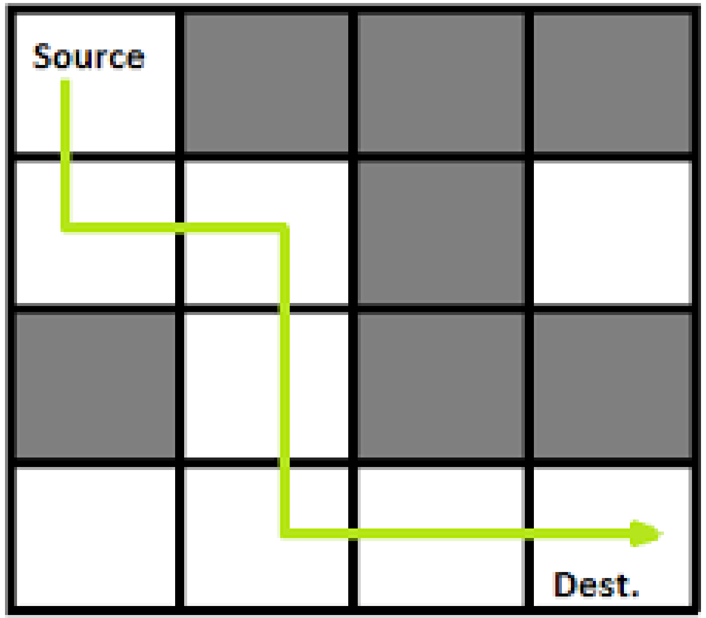
This function build a VERY deep stack very quickly

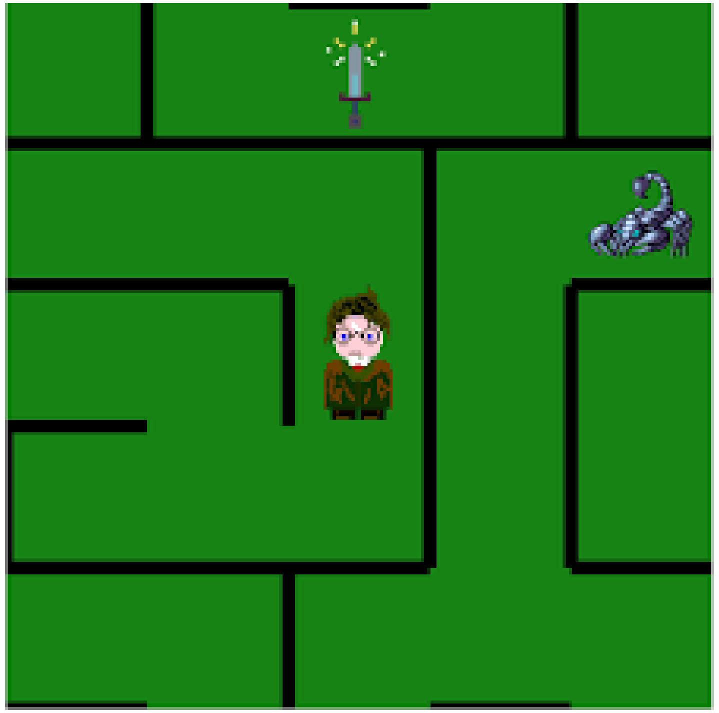
Day 1 : Questions

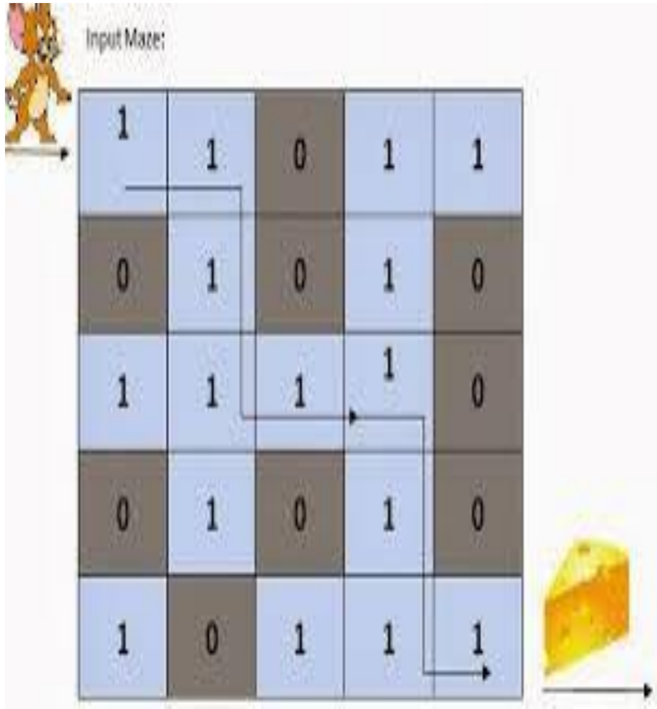
----

1. WHAT IS AN ALGORITHM?
2. WHY WE NEED TO DO ALGORITHM ANALYSIS?
3. WHAT ARE THE CRITERIA OF ALGORITHM ANALYSIS?
4.WHAT ARE ASYMPTOTIC NOTATIONS?
5. BRIEFLY EXPLAIN THE APPROACHES TO DEVELOP ALGORITHMS.
6. GIVE SOME EXAMPLES GREEDY ALGORITHMS.
7. WHAT ARE SOME EXAMPLES OF DIVIDE AND CONQUER ALGORITHMS?
8. WHICH PROBLEMS CAN BE SOLVED USING RECURSION?
9. HOW DOES RECURSION WORK IN JAVA?
10. WHAT IS TOWER OF HANOI?
11.WHY IS RECURSION USED?
12.WHAT ARE THE ADVANTAGES O AND DISADVANTAGES OF RECURSION?
13.DIFFERENTIATE BETWEEN RECURSION AND ITERATION.
14.WHAT IS HEAD AND TAIL RECURSION?
15.DISCUSS APPLICATIONS OF RECURSION.

Backtracking

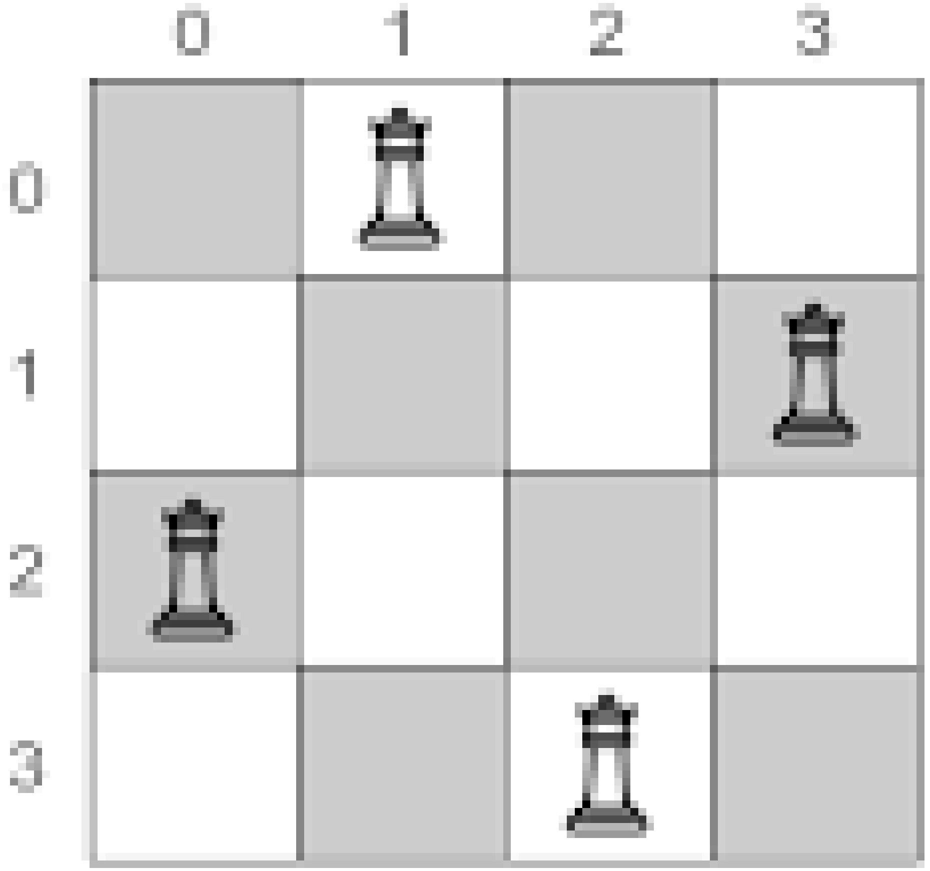




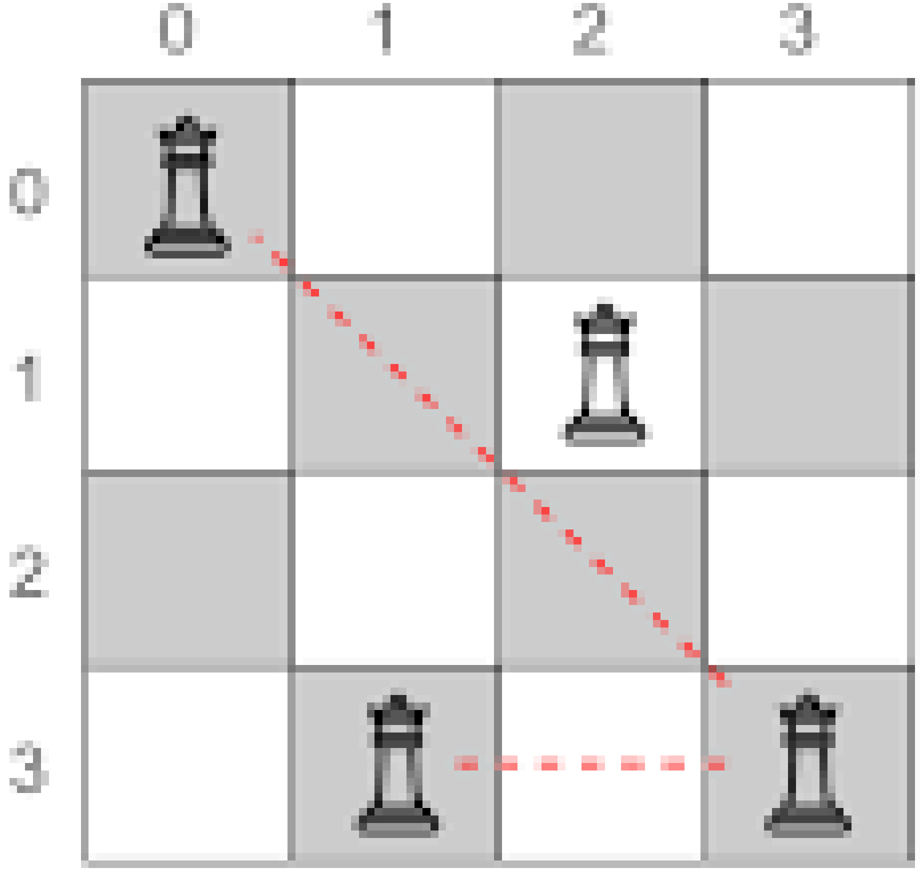


Input Maze:

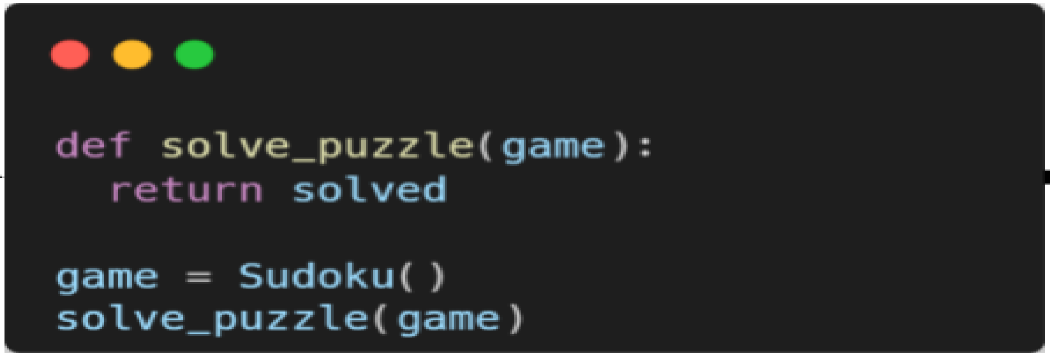
1	1	0	1	1
0	1	0	1	0
1	1	1	1	0
0	1	0	1	0
1	0	1	1	1



Valid queen positions



Invalid queen positions



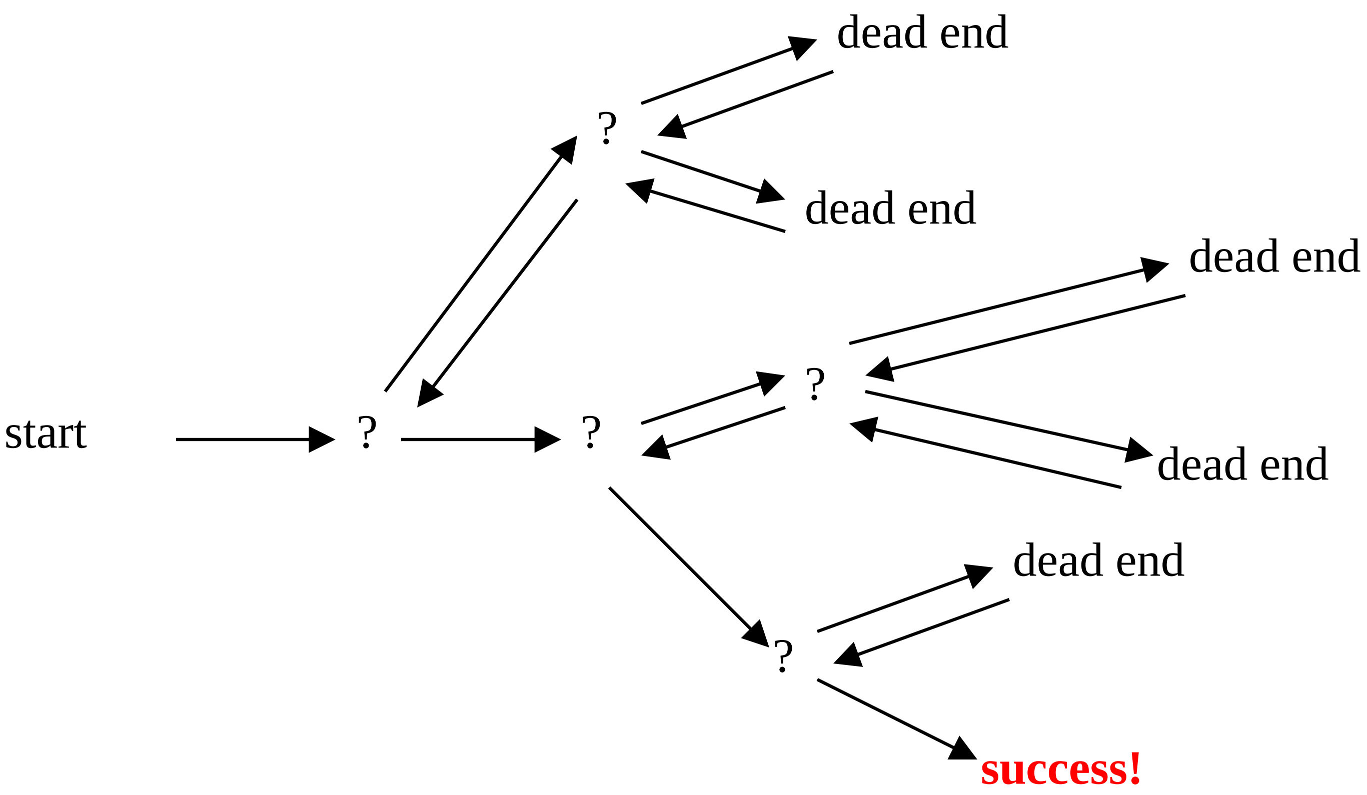
```
def solve_puzzle(game):
  return solved

game = Sudoku()
solve_puzzle(game)
```

		3			2	6	1	4
		2	6	4	1			8
	1	6		3	5		7	
			3				9	7
6	5				7		3	1
	3	7		5	4			
	7	9			6	2		
			5	8	3	7		
8	4			2			6	

5	8	3	9	7	2	6	1	4
7	9	2	6	4	1	3	5	8
4	1	6	8	3	5	9	7	2
1	2	4	3	6	8	5	9	7
6	5	8	2	9	7	4	3	1
9	3	7	1	5	4	8	2	6
3	7	9	4	1	6	2	8	5
2	6	1	5	8	3	7	4	9
8	4	5	7	2	9	1	6	3

Backtracking (animation)

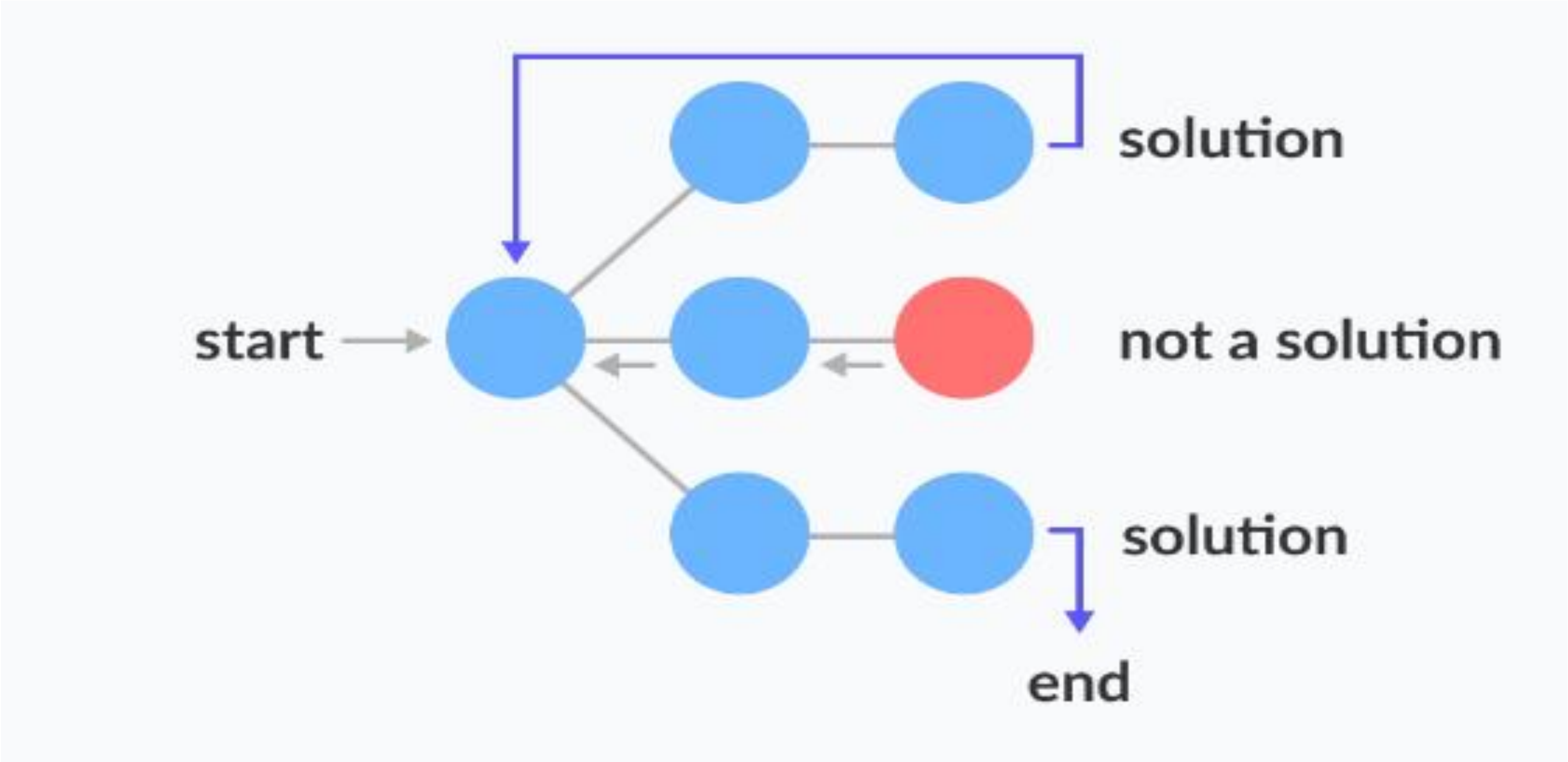


Backtracking

- Backtracking is a **problem-solving technique.**
- It involves systematically **exploring different paths** to find a solution.
- When faced with **multiple choices, backtracking tries each option.**
- It backtracks when it reaches **a dead end**.
- It's akin to navigating through a complex maze.
- **Wrong turns lead to retracing steps until the correct path** is found.
- Backtracking **enables the exploration of various possibilities**.
- It's a **powerful tool for tackling** challenging problems.

State Space Tree

- A space state tree is a tree representing **all the possible states (solution or nonsolution) of the problem** from the root as an initial state to the leaf as a terminal state.

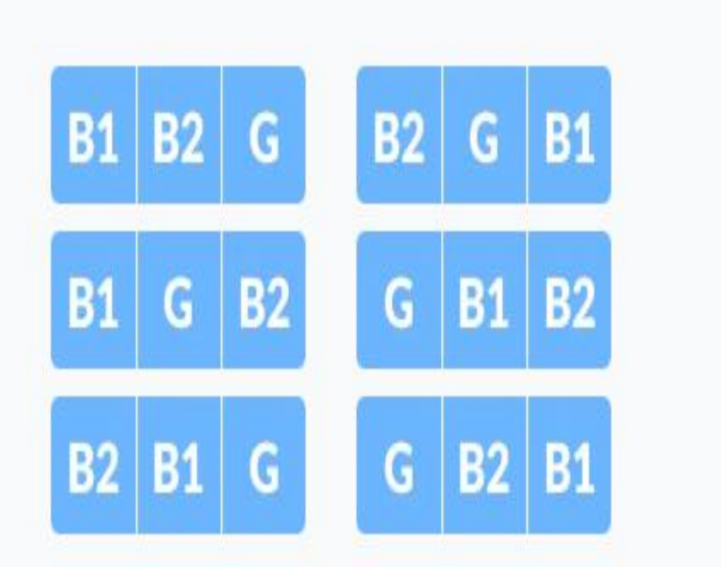


- **Backtracking Algorithm**

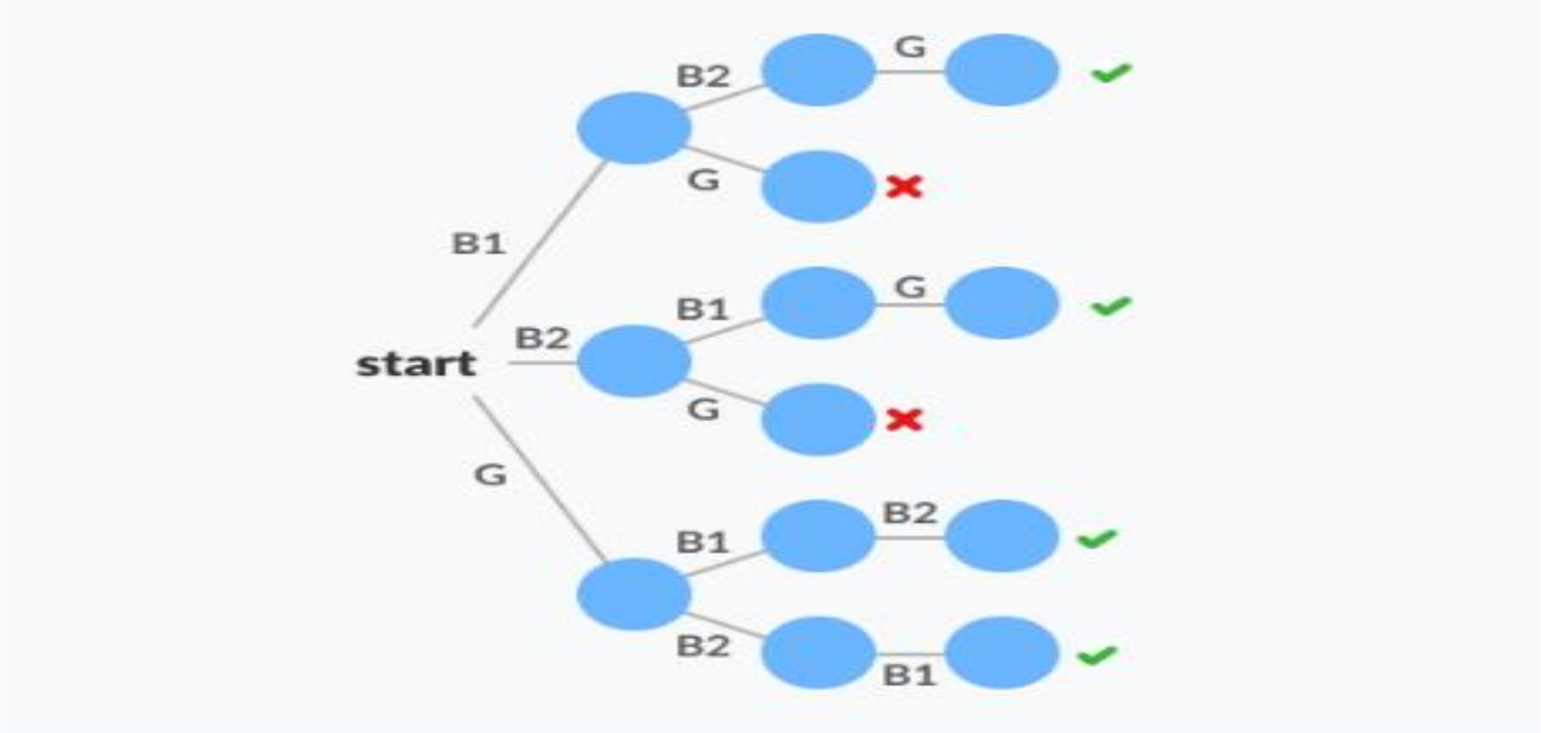
```
Backtrack(x)
    if x is not a solution
        return false
    if x is a new solution
        add to list of solutions
    backtrack(expand x)
```

Example Backtracking Approach

- Problem: You want to find all the possible ways of arranging 2 boys and 1 girl on 3 benches. Constraint: Girl should not be on the middle bench.



B1	B2	G	B2	G	B1
B1	G	B2	G	B1	B2
B2	B1	G	G	B2	B1

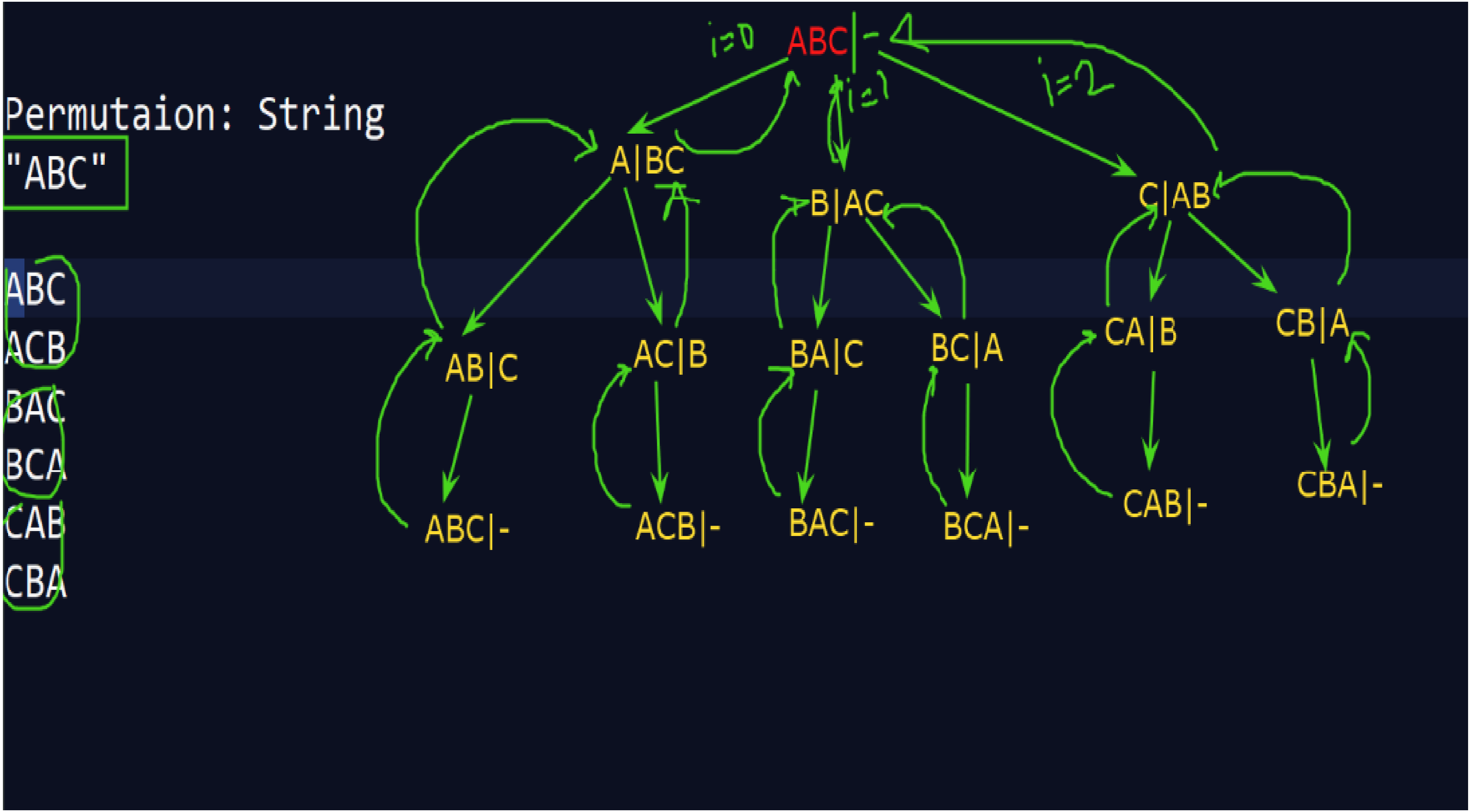


State tree with all the solutions

- **Backtracking Algorithm Applications**
- To find all Hamiltonian Paths present in a graph.
- To solve the N Queen problem.
- Maze solving problem.
- The Knight's tour problem.

Complexity Analysis of Backtracking

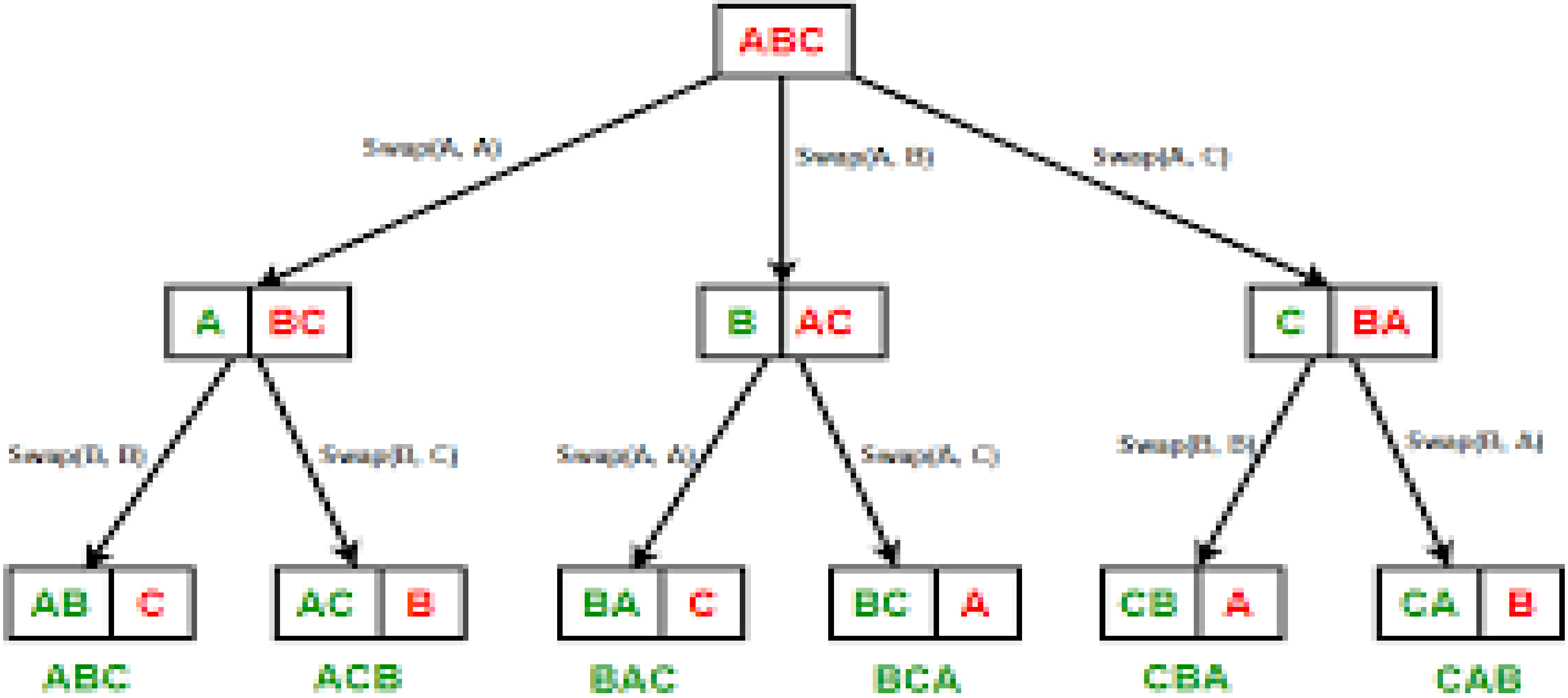
- Since backtracking algorithm is purely brute force therefore in terms of time complexity, it performs very poorly. Generally backtracking can be seen having below mentioned time complexities:
- Exponential (O(K^N))
- Factorial (O(N!))



Permutaion: String

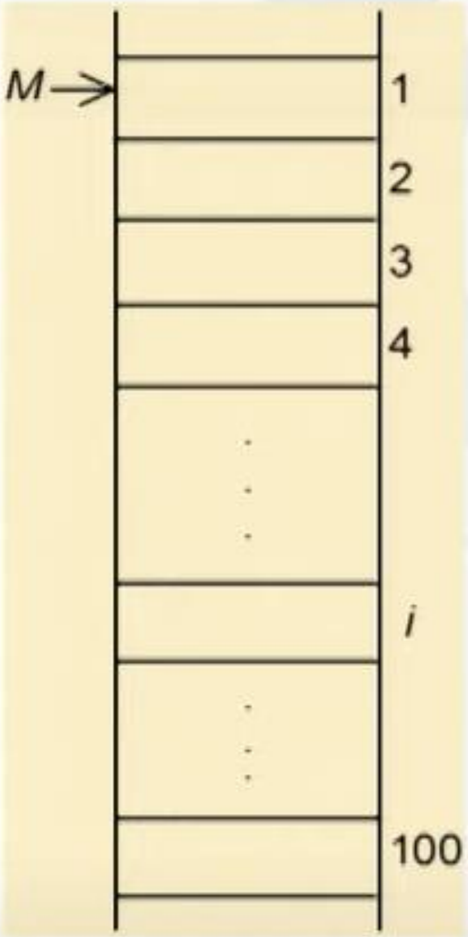
"ABC"

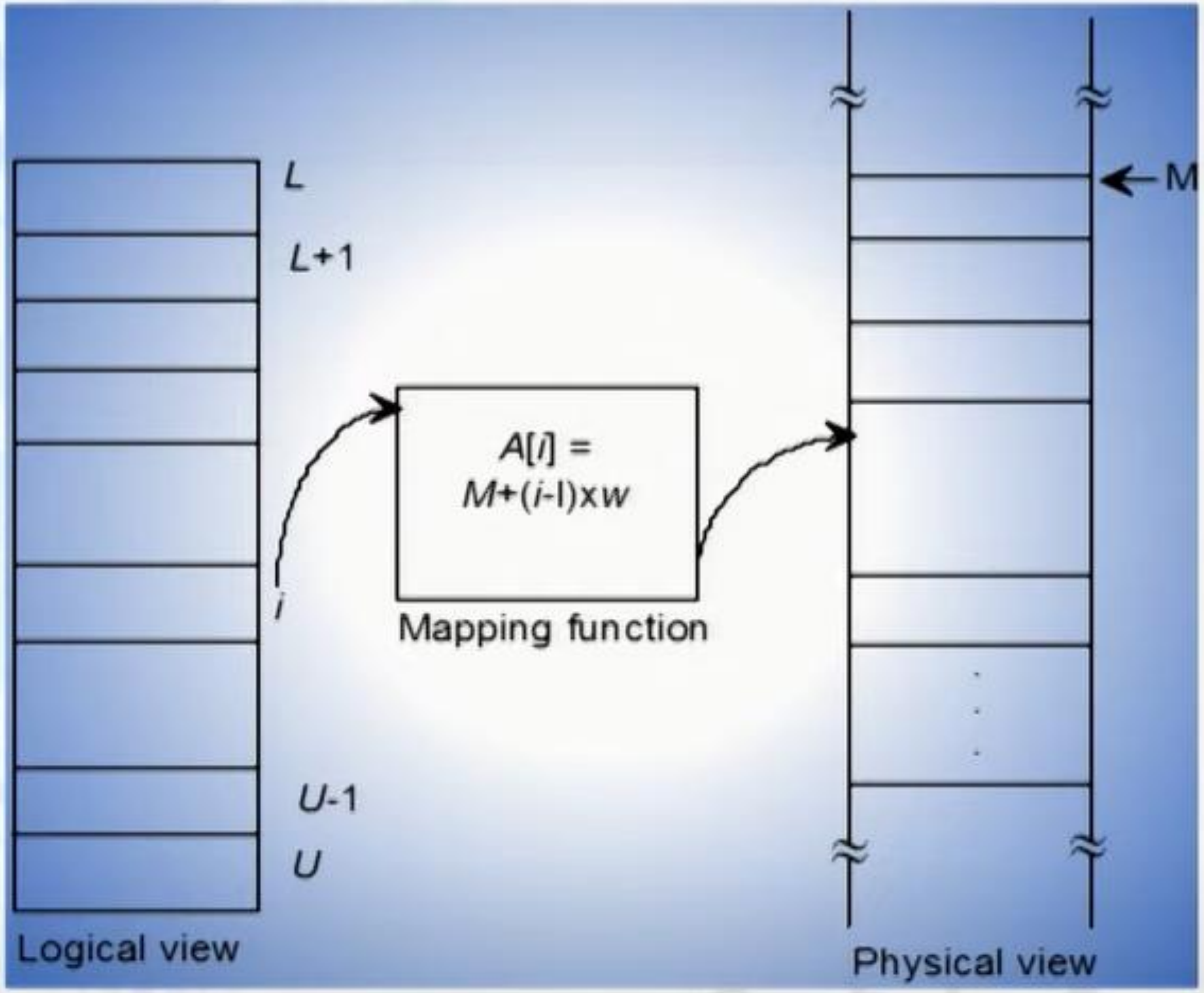
ABC
ACB
BAC
BCA
CAB
CBA



Recursion Tree for string "ABC"

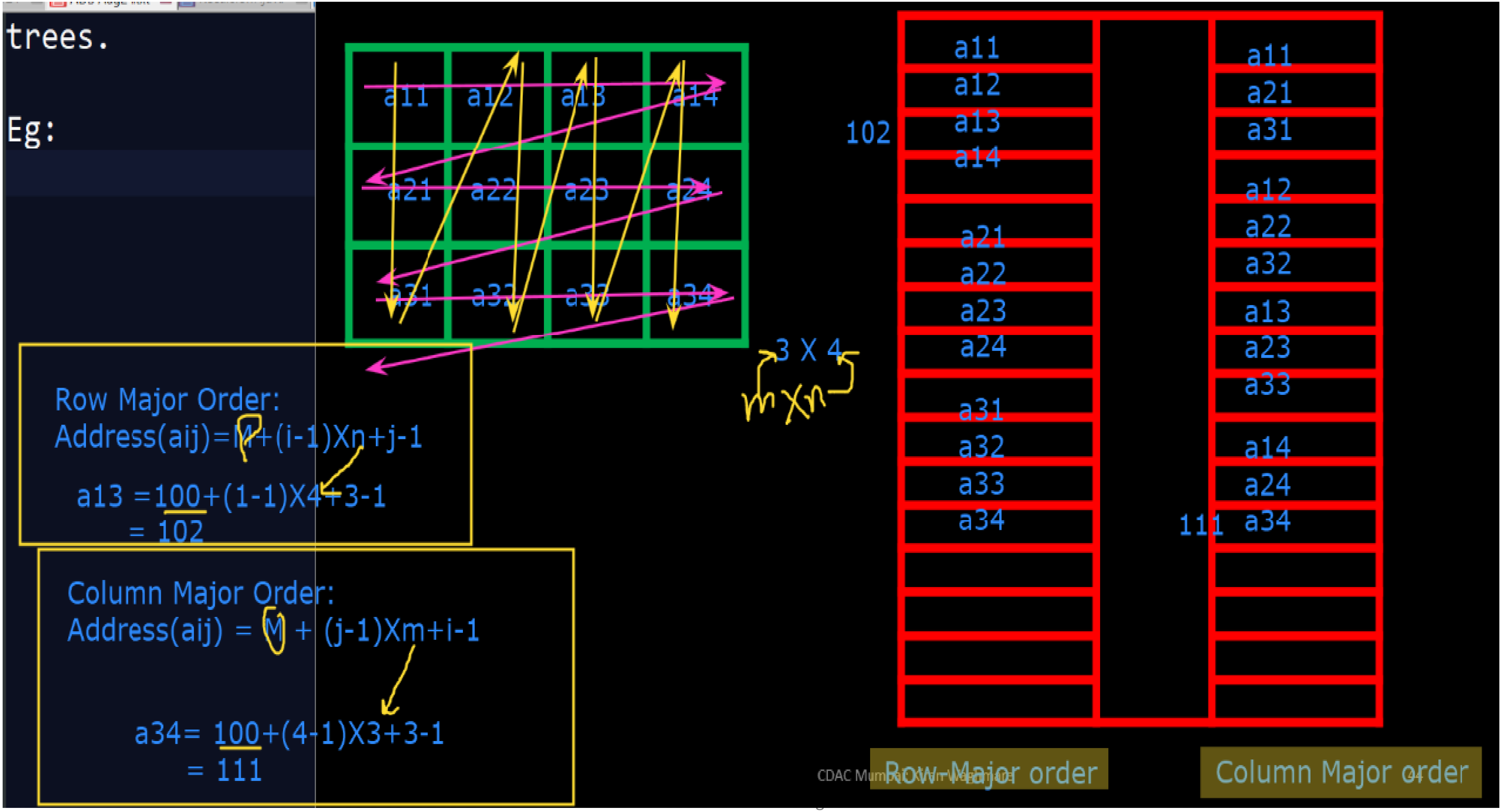
Concept of array





$$\text{Address }(\text{A}[i]) = M + (i - L) \times w$$

$$\text{Size }(\text{A}) = U - L + 1$$



trees.

Eg:

a11	a12	a13	a14
a21	a22	a23	a24
a31	a32	a33	a34

3 X 4

m X n

Row Major Order:
Address(aij)=M+(i-1)Xn+j-1

a13 =100+(1-1)X4+3-1
= 102

Column Major Order:
Address(aij) = M + (j-1)Xm+i-1

a34= 100+(4-1)X3+3-1
= 111

102

a11
a12
a13
a14

a21
a22
a23
a24

a31
a32
a33
a34

Row-Major order

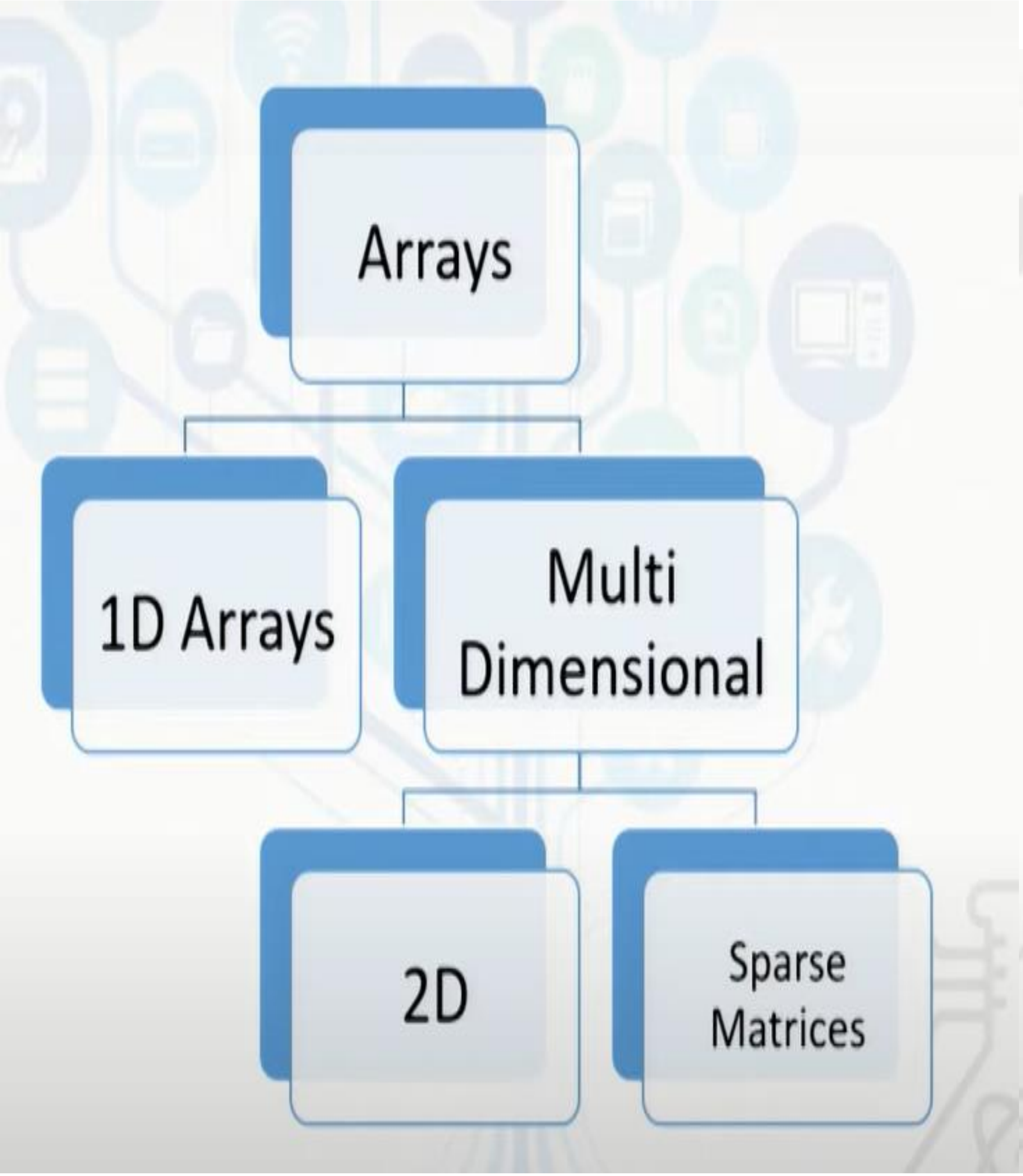
a11
a21
a31

a12
a22
a32

a13
a23
a33

a14
a24
111 a34

Column Major order



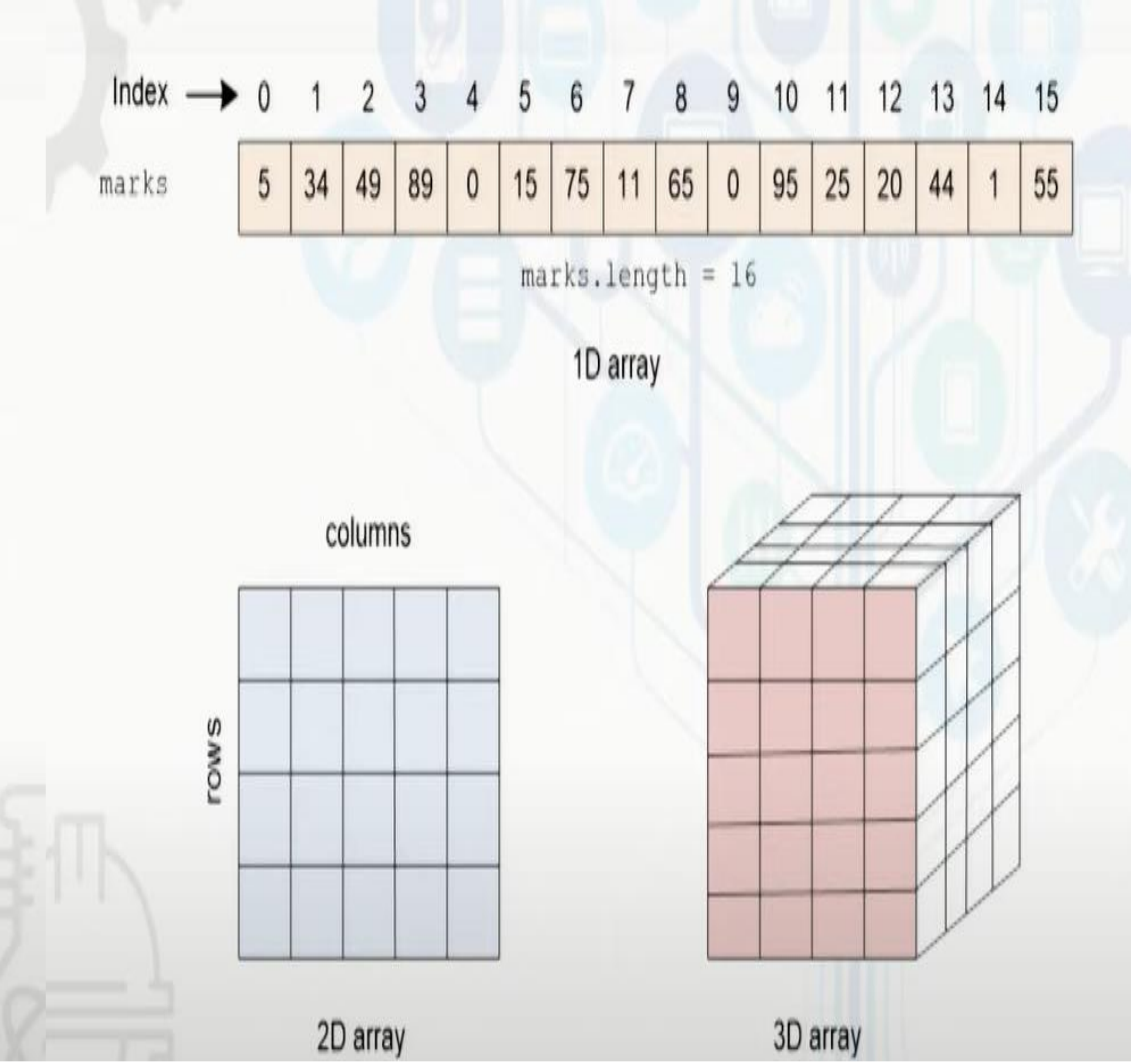
Arrays

1D Arrays

Multi Dimensional

2D

Sparse Matrices



Index →	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
marks	5	34	49	89	0	15	75	11	65	0	95	25	20	44	1	55

`marks.length = 16`

1D array

columns

rows

2D array

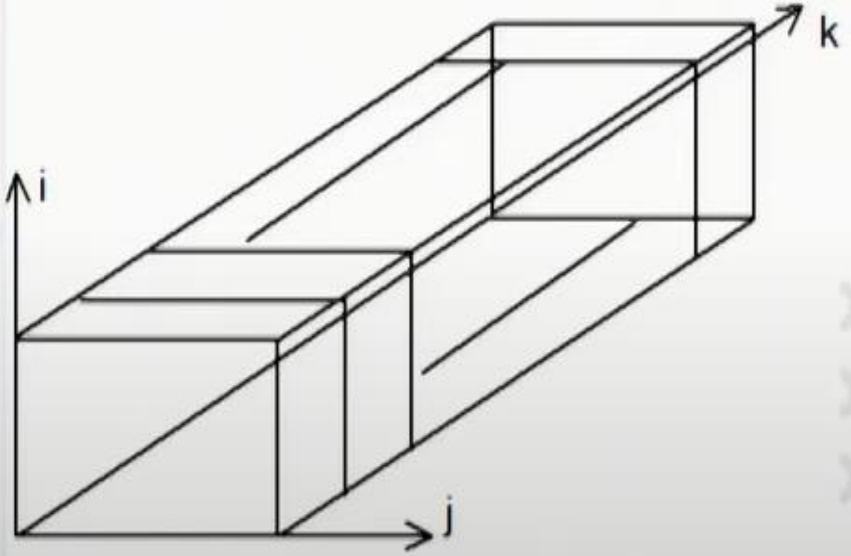
3D array

- More than one indexing to specify a location

2D: row, column

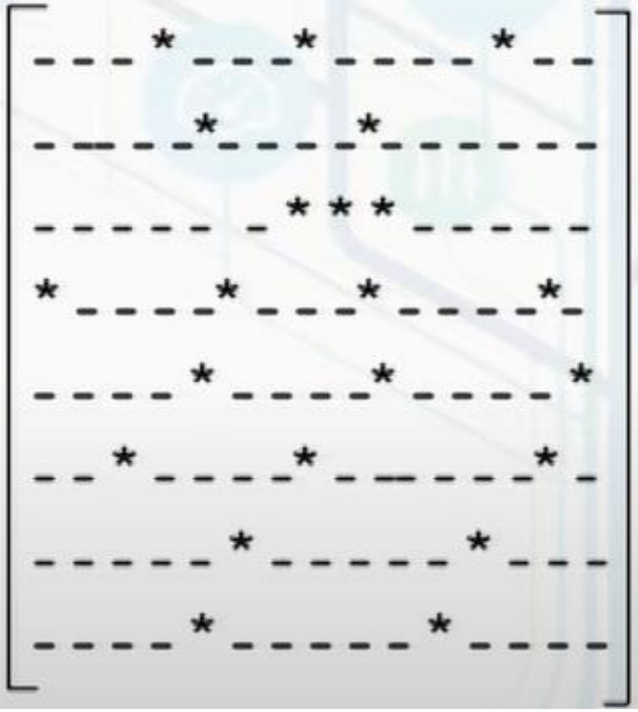
$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & a_{24} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ a_{m1} & a_{m2} & a_{m3} & a_{m4} & \cdots & a_{mn} \end{bmatrix}_{m \times n}$$

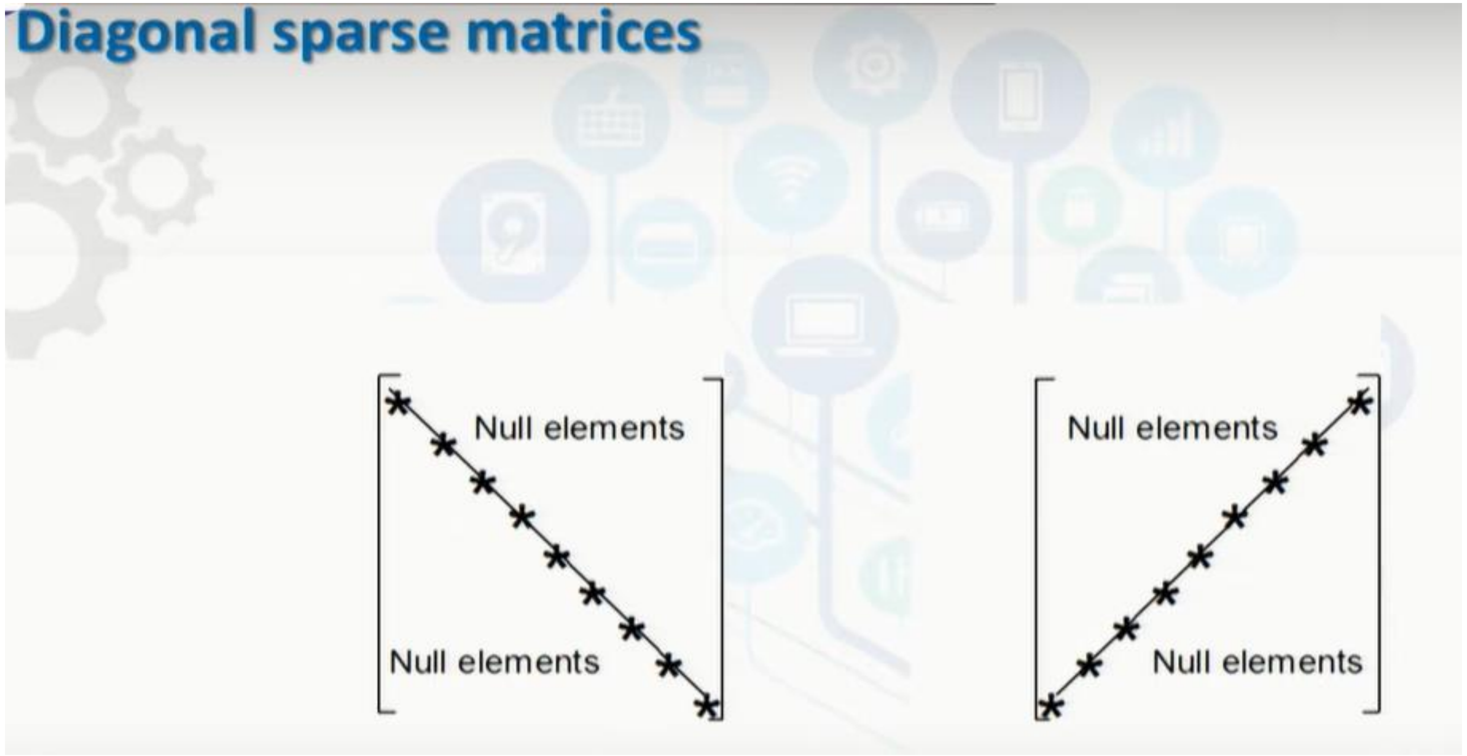
3D: row, column, height, etc.



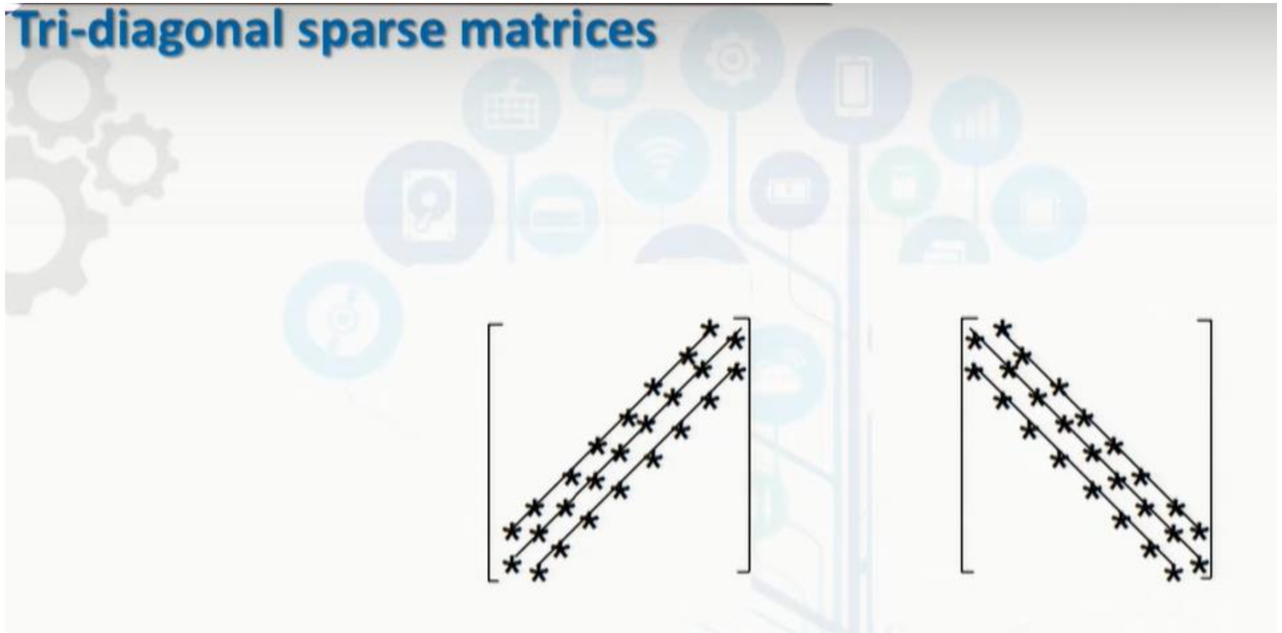
Sparse matrix

A *sparse* matrix is a two-dimensional array having the value of majority elements as null

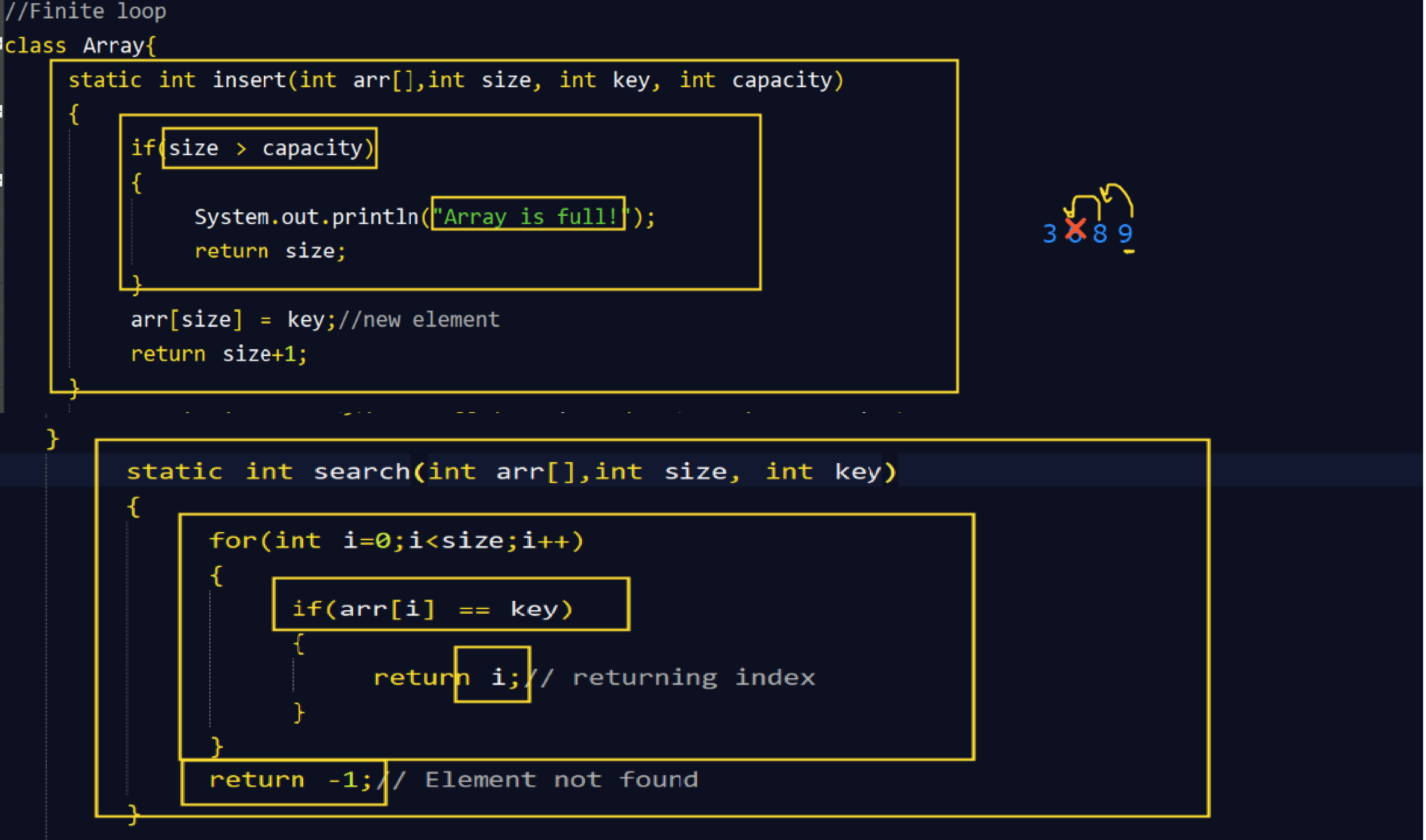




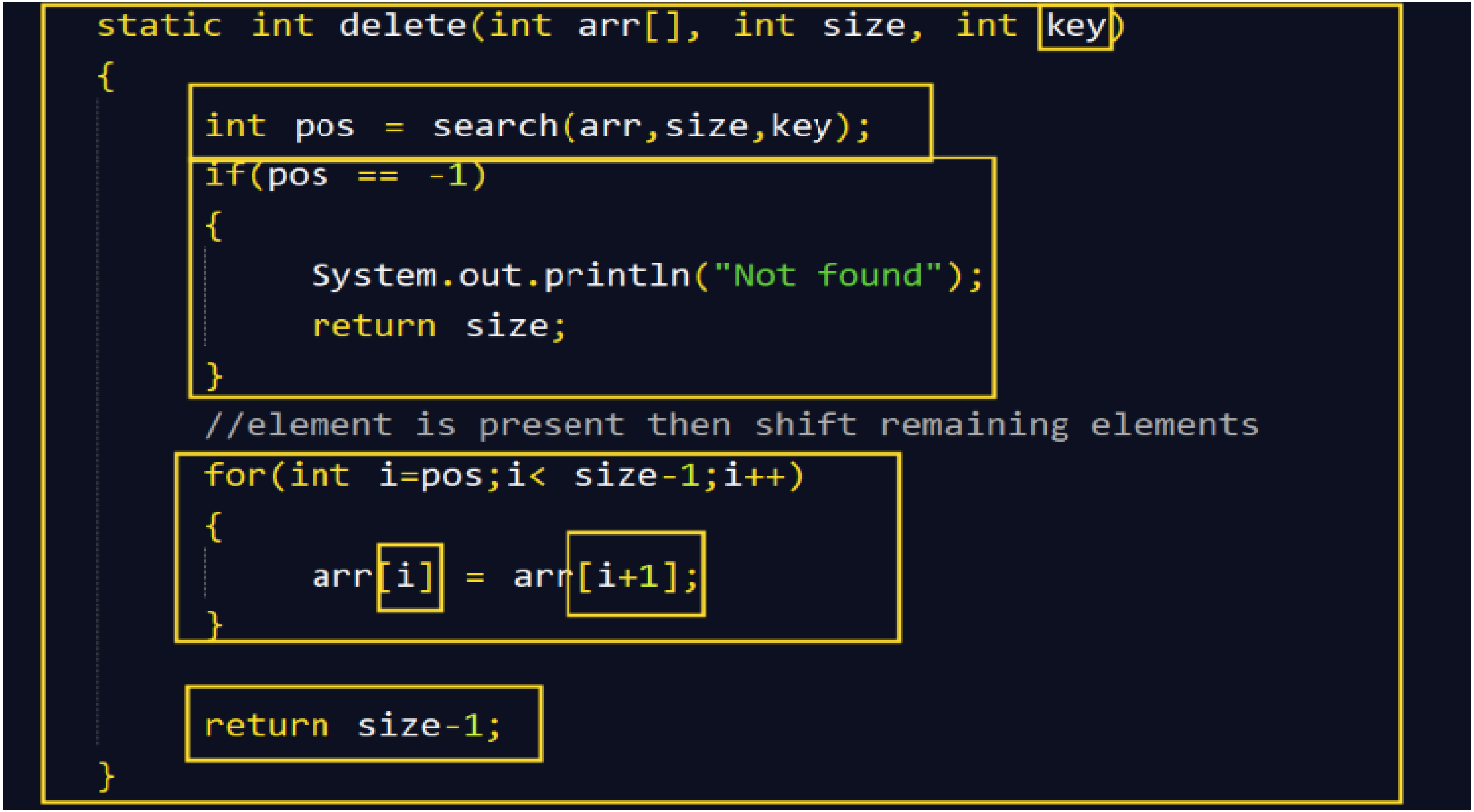
Diagonal sparse matrices



Tri-diagonal sparse matrices



```
//Finite loop
class Array{
    static int insert(int arr[],int size, int key, int capacity)
    {
        if(size > capacity)
        {
            System.out.println("Array is full!");
            return size;
        }
        arr[size] = key;//new element
        return size+1;
    }
}
    static int search(int arr[],int size, int key)
    {
        for(int i=0;i<size;i++)
        {
            if(arr[i] == key)
            {
                return i;// returning index
            }
        }
        return -1;// Element not found
    }
```



```
static int delete(int arr[], int size, int key)
{
    int pos = search(arr,size,key);
    if(pos == -1)
    {
        System.out.println("Not found");
        return size;
    }
    //element is present then shift remaining elements
    for(int i=pos;i< size-1;i++)
    {
        arr[i] = arr[i+1];
    }

    return size-1;
}
```

Problem statement: Find duplicates in an array

- Given an array a1[] of size N which contains elements from 0 to N-1, you need to find all the elements occurring more than once in the given array.

- **Example 1:**
  - Input:
    - N = 4
    - a[] = {0,3,1,2}
  - Output: -1
  - Explanation: N=4 and all elements from 0 to (N-1 = 3) are present in the given array. Therefore output is -1.

- **Example 2:**
  - Input:
    - N = 5
    - a[] = {2,3,1,2,3}
  - Output: 2 3
  - Explanation: 2 and 3 occur more than once in the given array.

Problem statement: Removing punctuations from a given string

- **Given a string, remove the punctuation from the string if the given character is a punctuation character, as classified by the current C locale. The default C locale classifies these characters as punctuation:**
    - ! " # $ % & ' ( ) * + , - . / : ; ? @ [\ ] ^ _ ` { | } ~

- **Example 1:**
    - Input : %welcome' to @cdacmumbai?<s
    - Output : welcome to cdacmumbai

- **Example 2:**
    - Input : Hello!!!, he said ---and went**.
    - Output : Hello he said and went

Problem statement: Program to find the initials of a name.

- Given a string name, we have to find the initials of the name
- Examples 1:
  - Input : Kabhi Haa Kabhi Naa
  - Output : K H K N
    - We take the first letter of all
    - words and print in capital letter.
- Example 2:
  - Input : Mahatma Gandhi
  - Output : M G
- Example 3:
  - Input : Shah Rukh Khan
  - Output : S R K

  - Example 4: your own name

Problem Statement : Find the Missing Number

You are given a list of n-1 integers and these integers are in the range of 1 to n. There are no duplicates in the list. One of the integers is missing in the list. Write an efficient code to find the missing integer.

Example:

Input: arr[] = {1, 2, 4, 6, 3, 7, 8}
Output: 5
Explanation: The missing number from 1 to 8 is 5

Input: arr[] = {1, 2, 3, 5}
Output: 4
Explanation: The missing number from 1 to 5 is 4