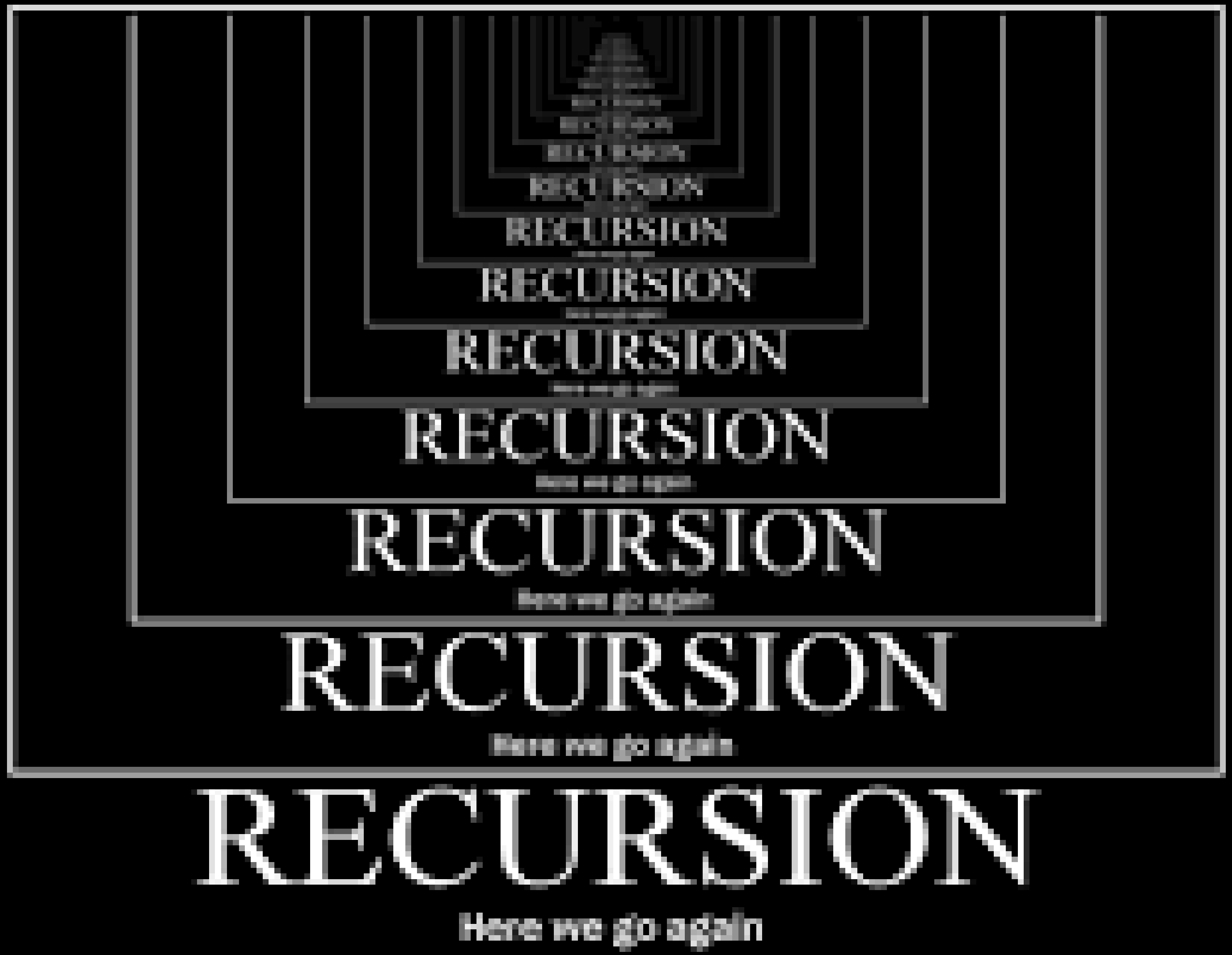
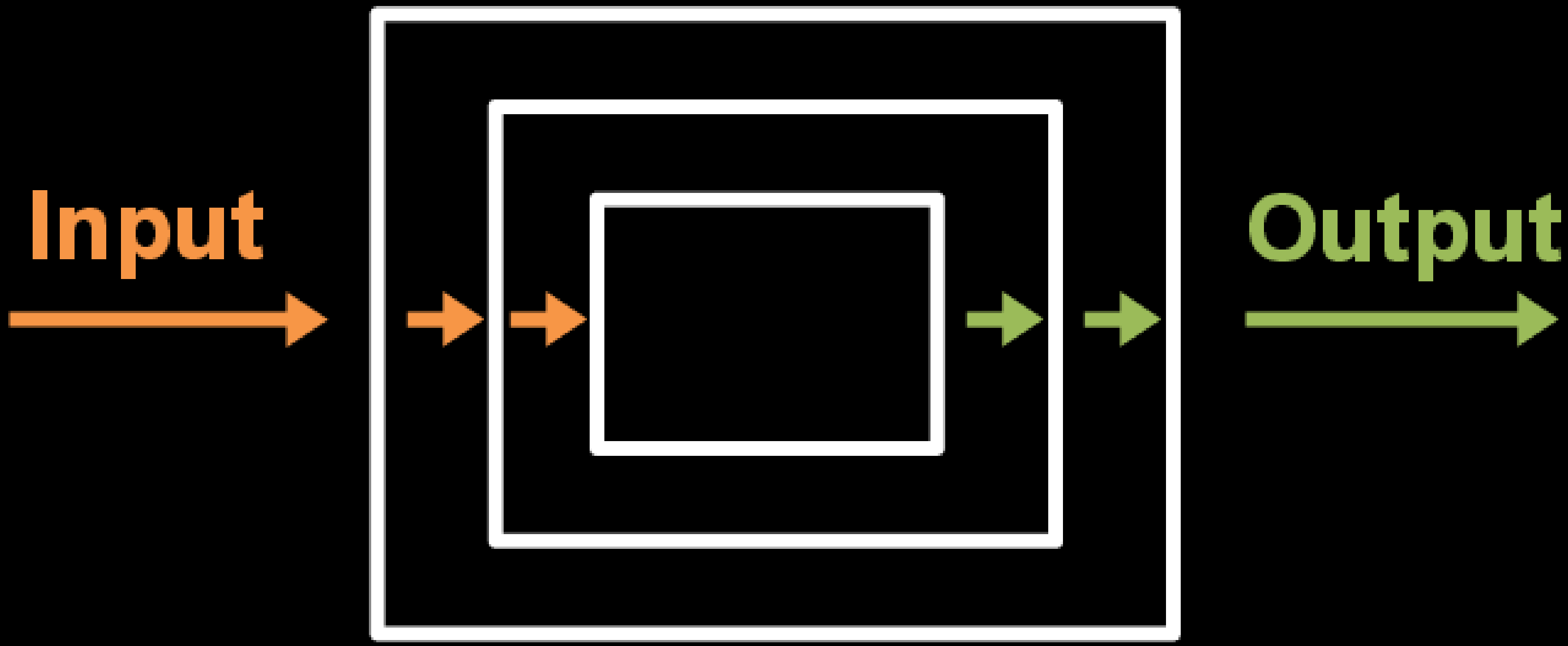
Topics

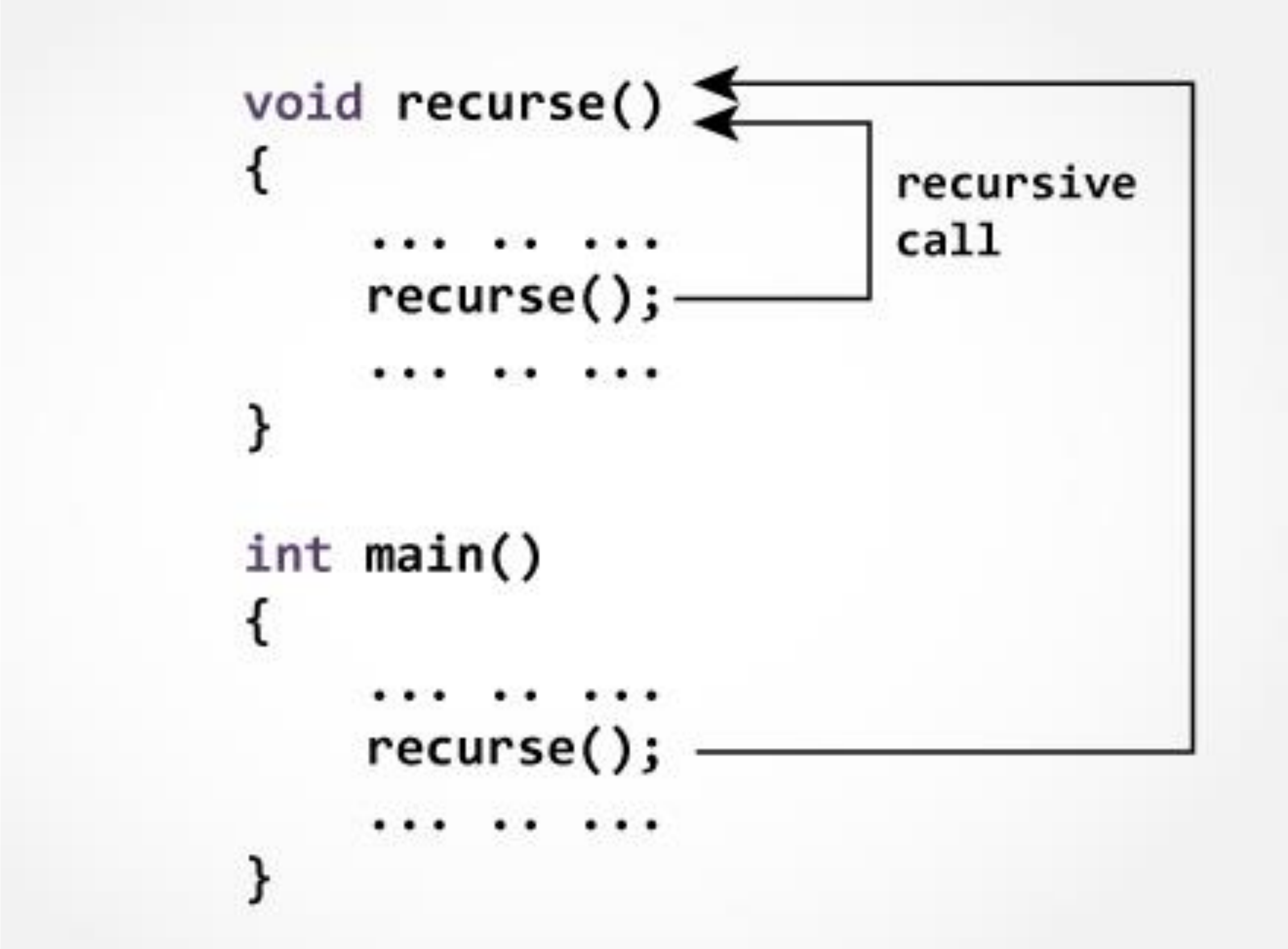
1. Recursive definitions and Processes
2. Writing Recursive Programs
3. Efficiency in Recursion
4. Towers of Hanoi problem.


RECURSION
Here we go again
RECURSION
Here we go again
RECURSION
Here we go again
RECURSION
Here we go again
RECURSION
Here we go again
RECURSION
RECURSION

Recursion



How does Recursion works?

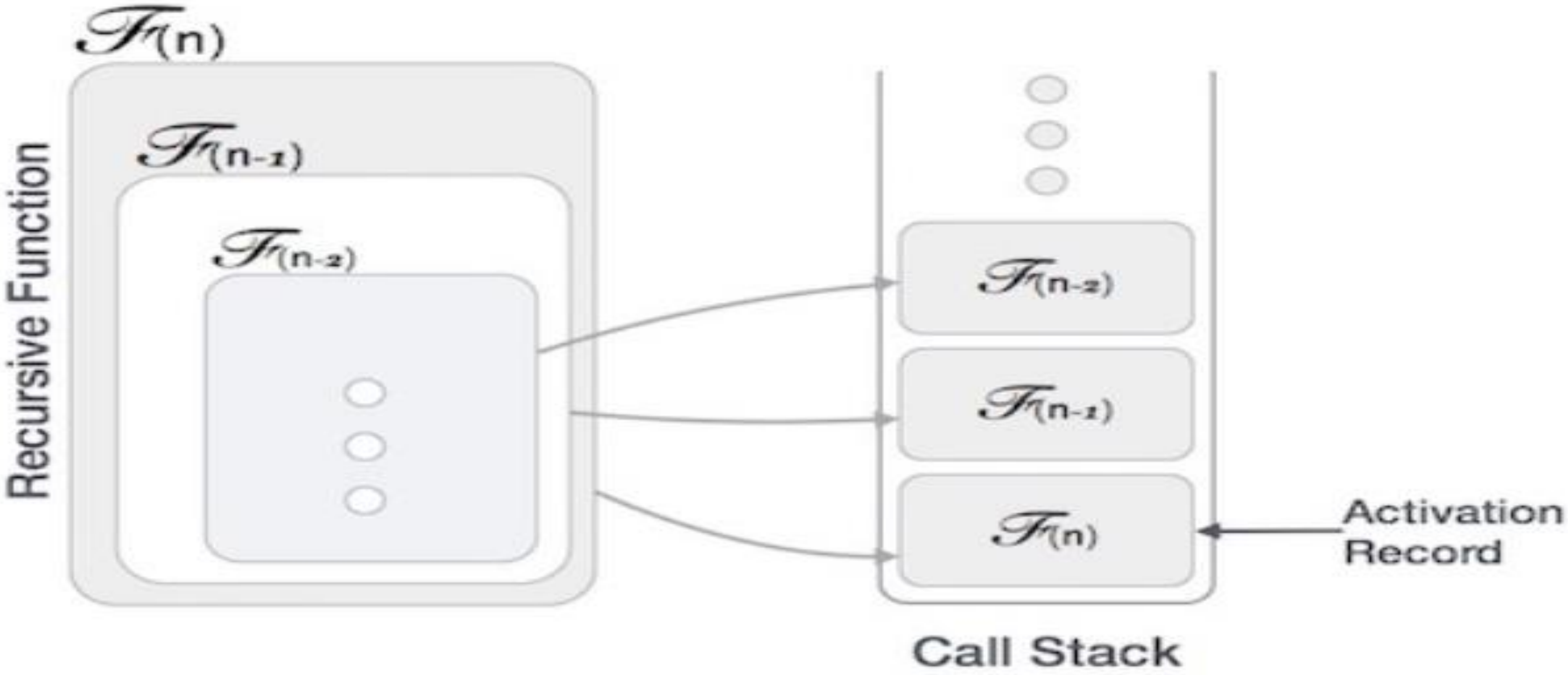


```
void recurse()
{
    ... .. ...
    recurse();
    ... .. ...
}

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

Recursion

- Any function which calls itself directly or indirectly is called Recursion and the corresponding function is called as recursive function.
- A recursive method solves a problem by calling a copy of itself to work on a smaller problem.
- It is important to ensure that the recursion terminates.
- Each time the function call itself with a slightly simple version of the original problem.
- Using recursion, certain problems can be solved quite easily.
- E.g: Tower of Hanoi (TOH), Tree traversals, DFS of Graph etc.,

How Data Structure Recursive function is implemented?

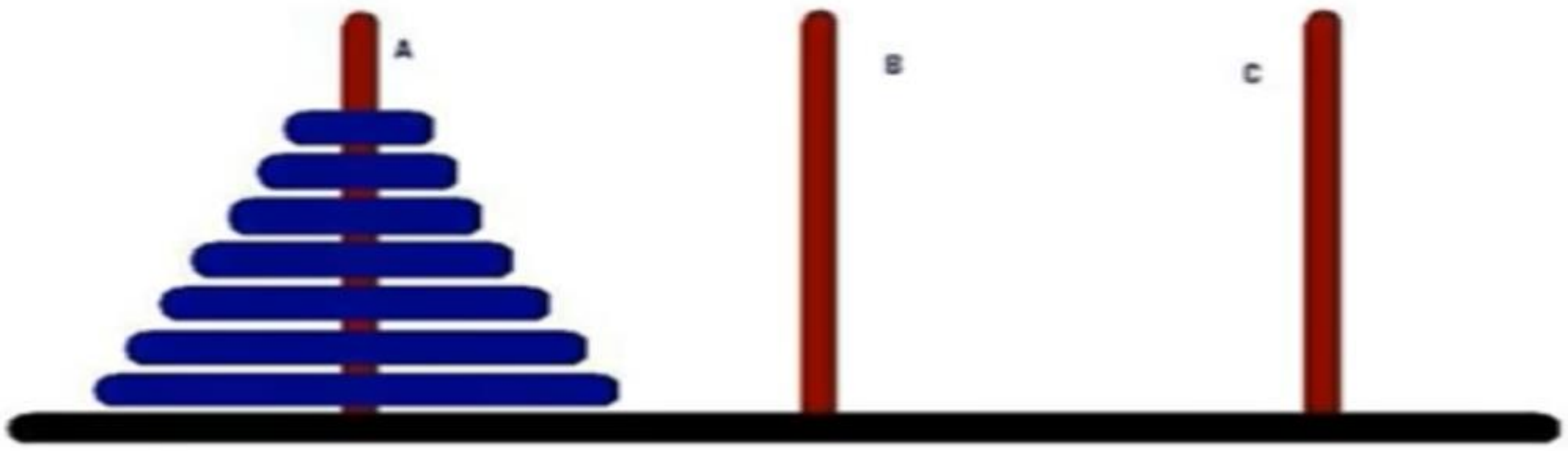


Application

- Fibonacci series
- Factorial of a number
- Merge sort, Quick sort
- Binary search
- Tree Traversal
- Graph Traversals (DFS & BFS)
- Dynamic Programming
- Divide & Conquer Algorithm
- Tower of Hanoi
- Backtracking Algorithms
- Greatest Common Divisor

Tower of Hanoi

- It is a mathematical puzzle.
- Inventor: French mathematician, Edouard Lucas in 1883.
- Objective of the puzzle is to move the entire stack to another rod.



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

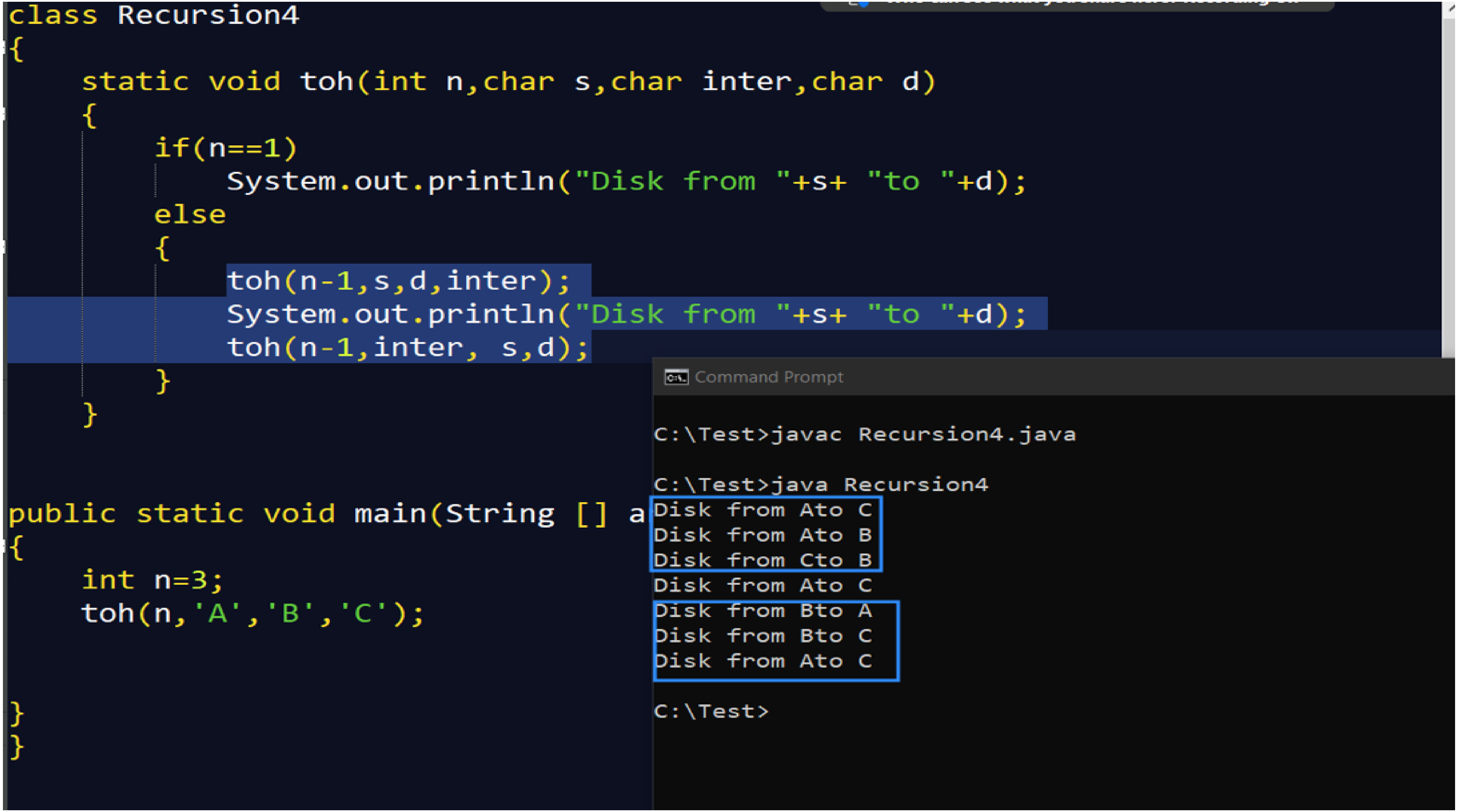
- **The Tower of Hanoi puzzle is solved by moving all the disks to another tower by not violating the sequence of the arrangements.**

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.

Home Work

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



```
class Recursion4
{
    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 [] a
{
    int n=3;
    toh(n,'A','B','C');


}
}
```

```
C:\Test>javac Recursion4.java

C:\Test>java Recursion4
Disk from Ato C
Disk from Ato B
Disk from Cto B
Disk from Ato C
Disk from Bto A
Disk from Bto C
Disk from Ato C

C:\Test>
```

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?

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

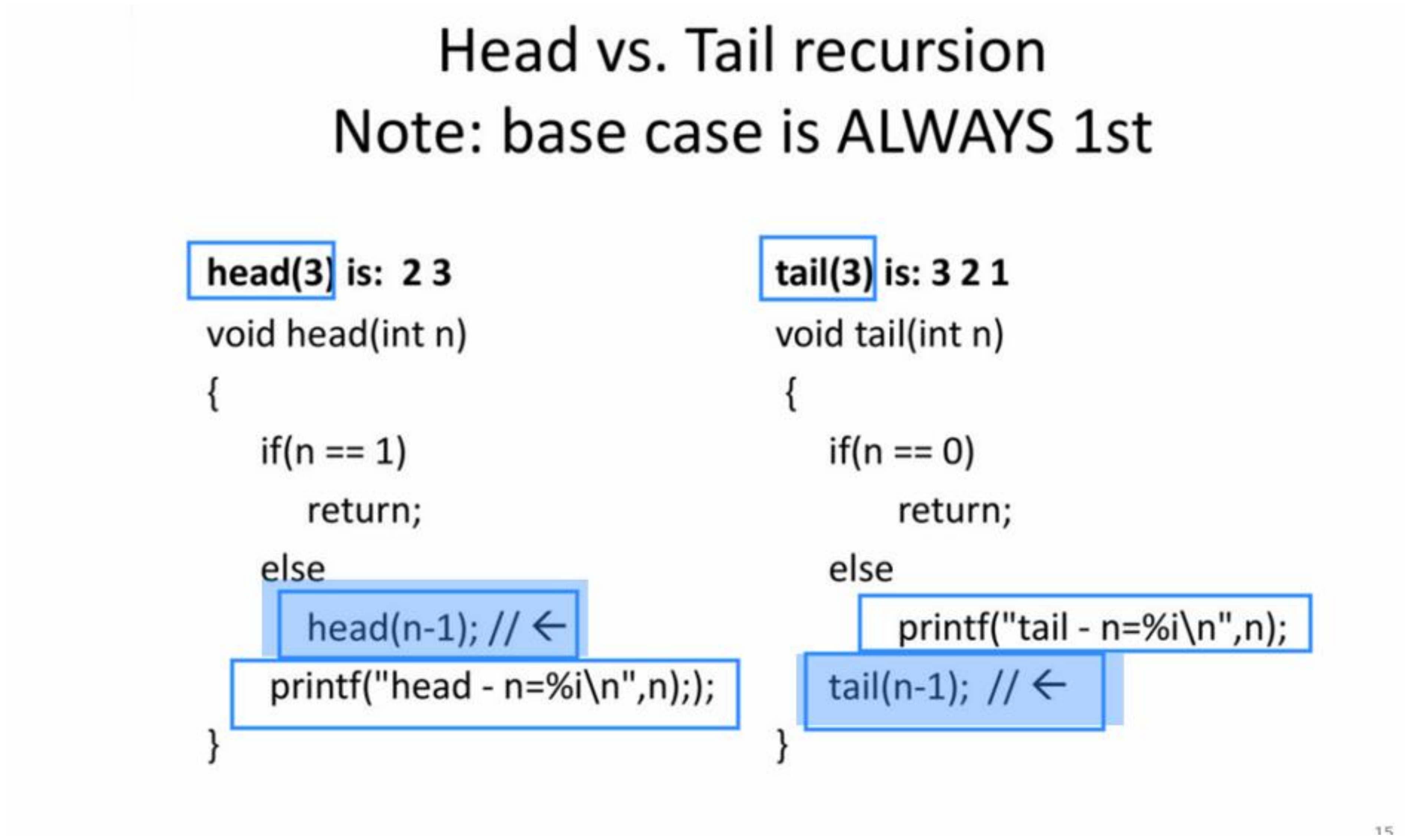
```
if (answer is known)
    provide the answer & exit
else
    call same function with
    a smaller version
    of the same problem
```

base case

recursive case

Type of Recursion

- Tail Recursion
- Head Recursion



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

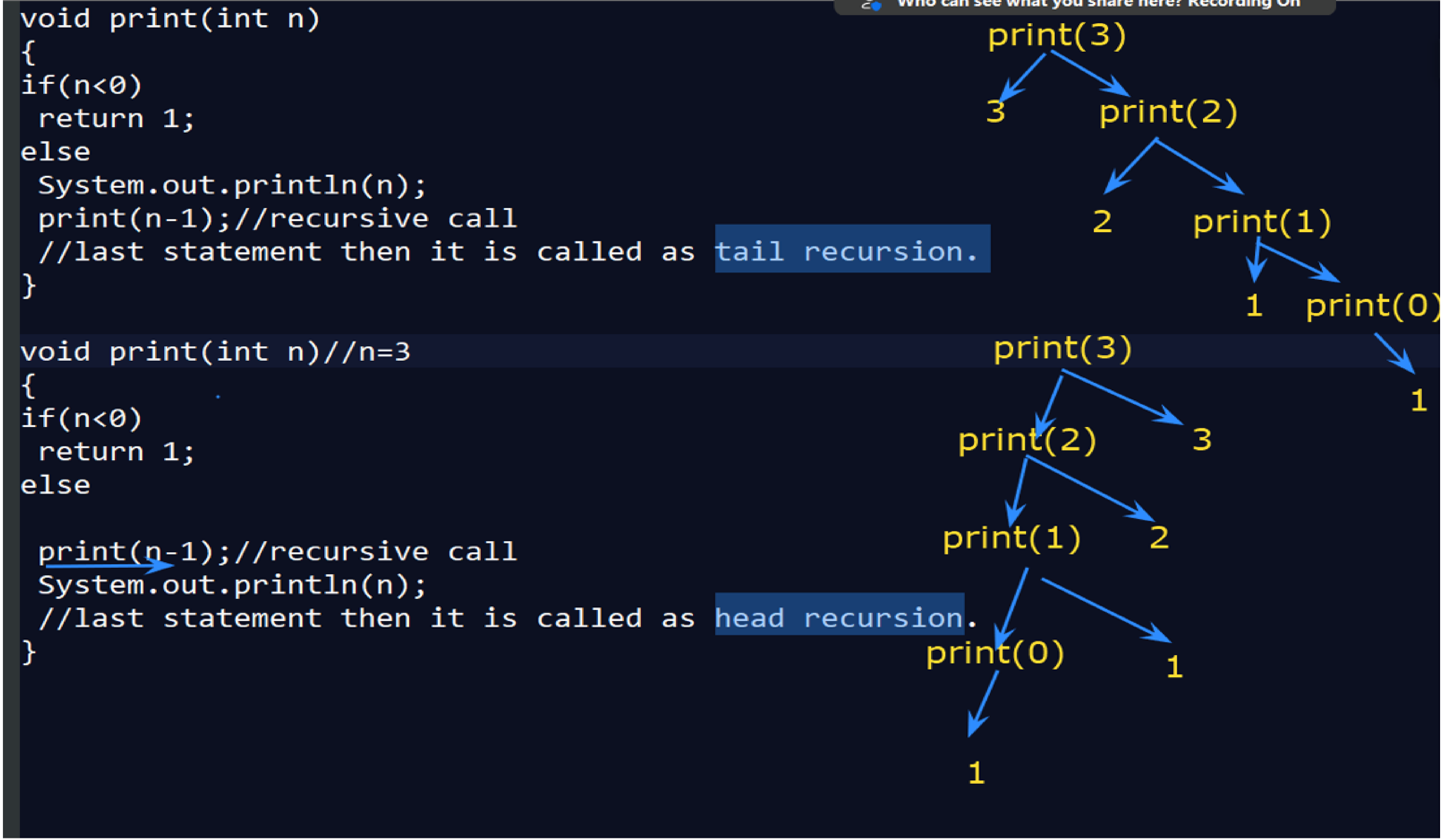
head(3) is: 2 3

```
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);  // ←
}
```

15



```
void print(int n)
{
if(n<0)
 return 1;
else
 System.out.println(n);
 print(n-1);//recursive call
 //last statement then it is called as tail recursion.
}

void print(int n)//n=3
{
if(n<0)
 return 1;
else

 print(n-1);//recursive call
 System.out.println(n);
 //last statement then it is called as head recursion.
}
```

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:

Base condition

```
if(n==1)
return "int"
```

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

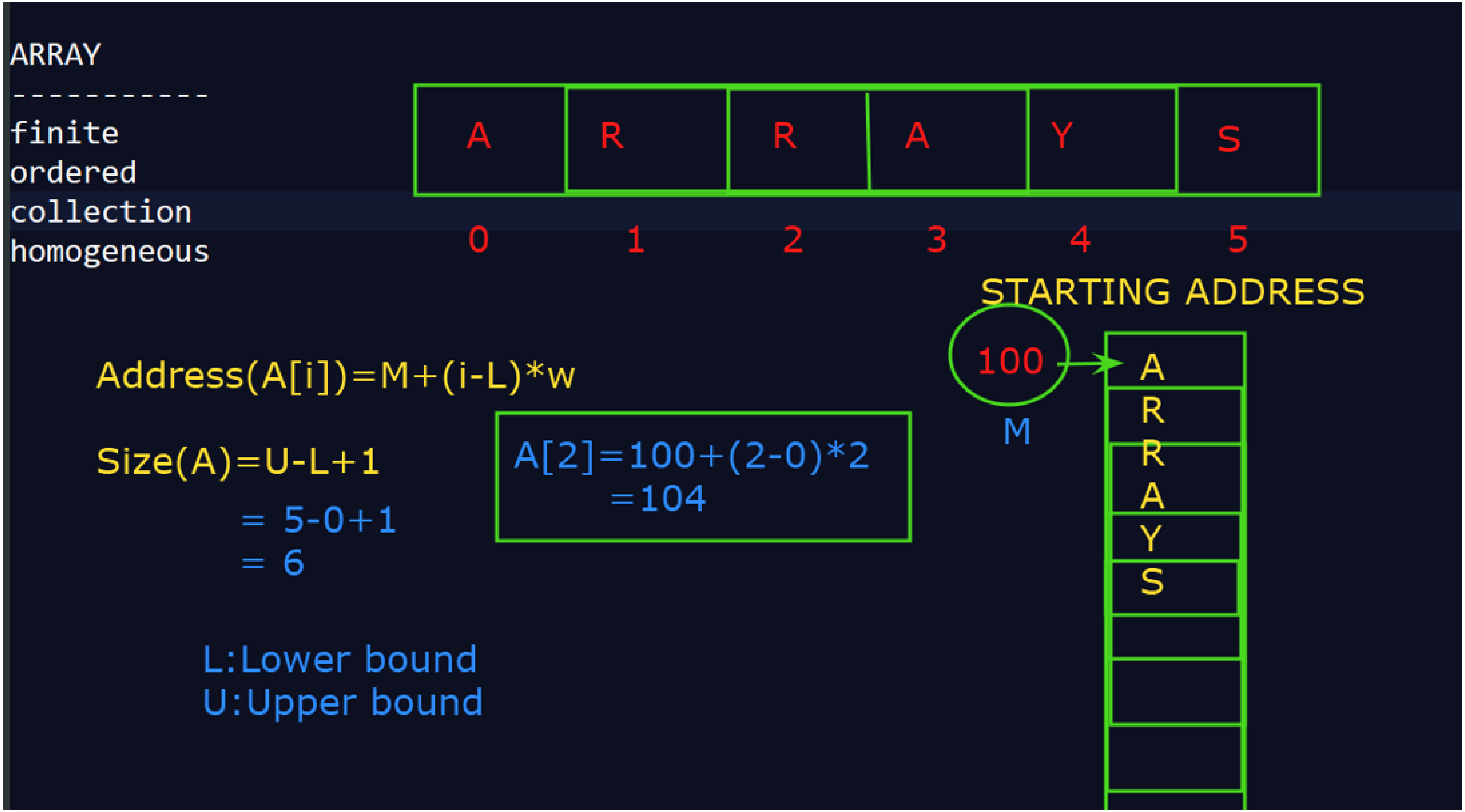
Input : n = 5
Output : gcd(int, gcd(int, gcd(int, gcd(int, int))))

4,6
4=4,2,1
6=6,3,2,1
GCD(4,6)=2

```
GCD(a,b)
if(a>b)
GCD(a%b,b)
else
GCD(a,b%a)
```

Algorithms & Data Structure

Kiran Waghmare



```
ARRAY
-----------
finite
ordered
collection
homogeneous
```

A	R	R	A	Y	S
0	1	2	3	4	5

STARTING ADDRESS

100 → A R R A Y S

M

Address(A[i])=M+(i-L)*w

Size(A)=U-L+1

= 5-0+1

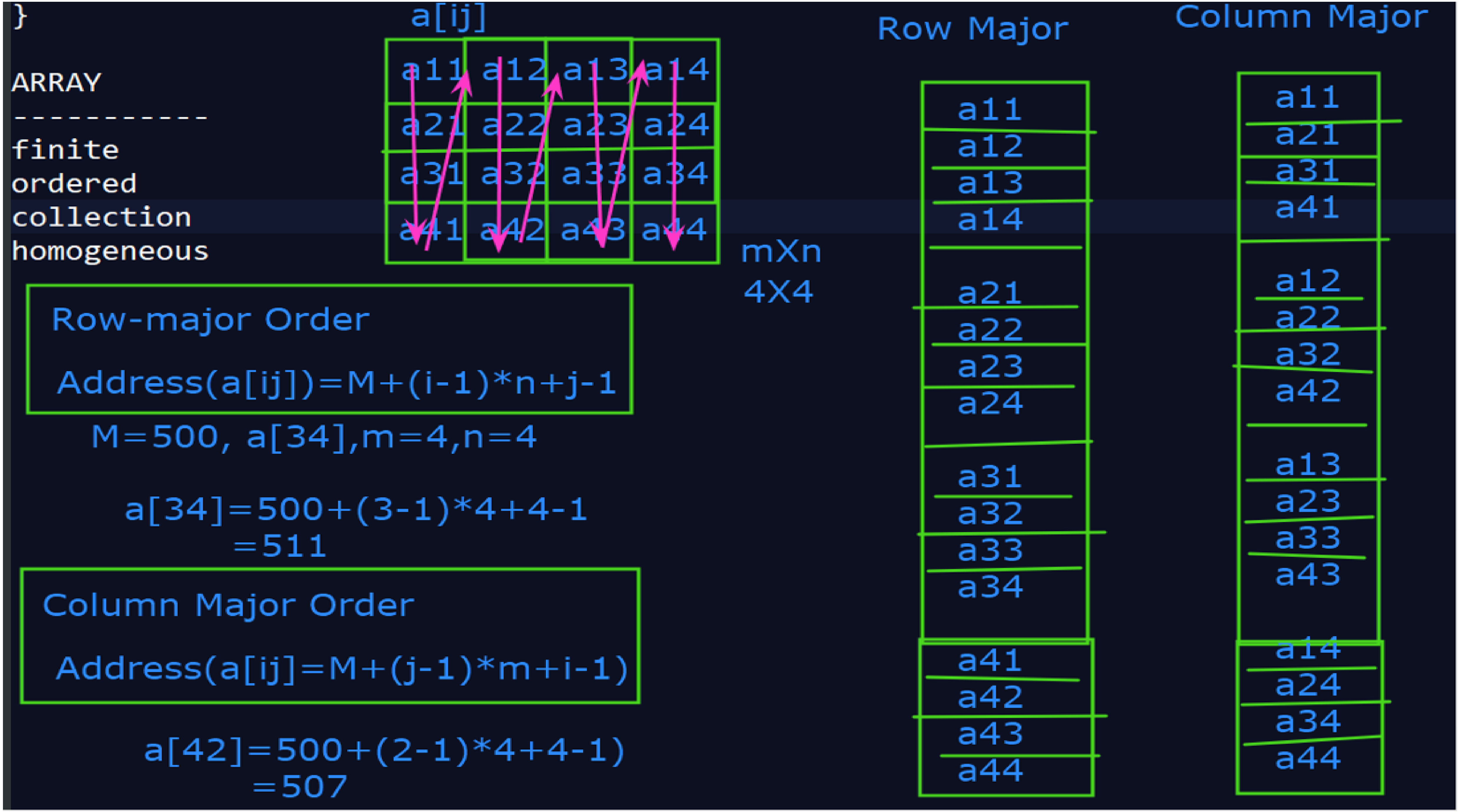
= 6

A[2]=100+(2-0)*2

=104

L:Lower bound

U:Upper bound



}

ARRAY

----

finite
ordered
collection
homogeneous

a[ij]

a11	a12	a13	a14
a21	a22	a23	a24
a31	a32	a33	a34
a41	a42	a43	a44

mXn
4X4

Row-major Order

Address(a[ij])=M+(i-1)*n+j-1

M=500, a[34],m=4,n=4

a[34]=500+(3-1)*4+4-1
=511

Column Major Order

Address(a[ij]=M+(j-1)*m+i-1)

a[42]=500+(2-1)*4+4-1)
=507

Row Major

a11
a12
a13
a14

a21
a22
a23
a24

a31
a32
a33
a34

a41
a42
a43
a44

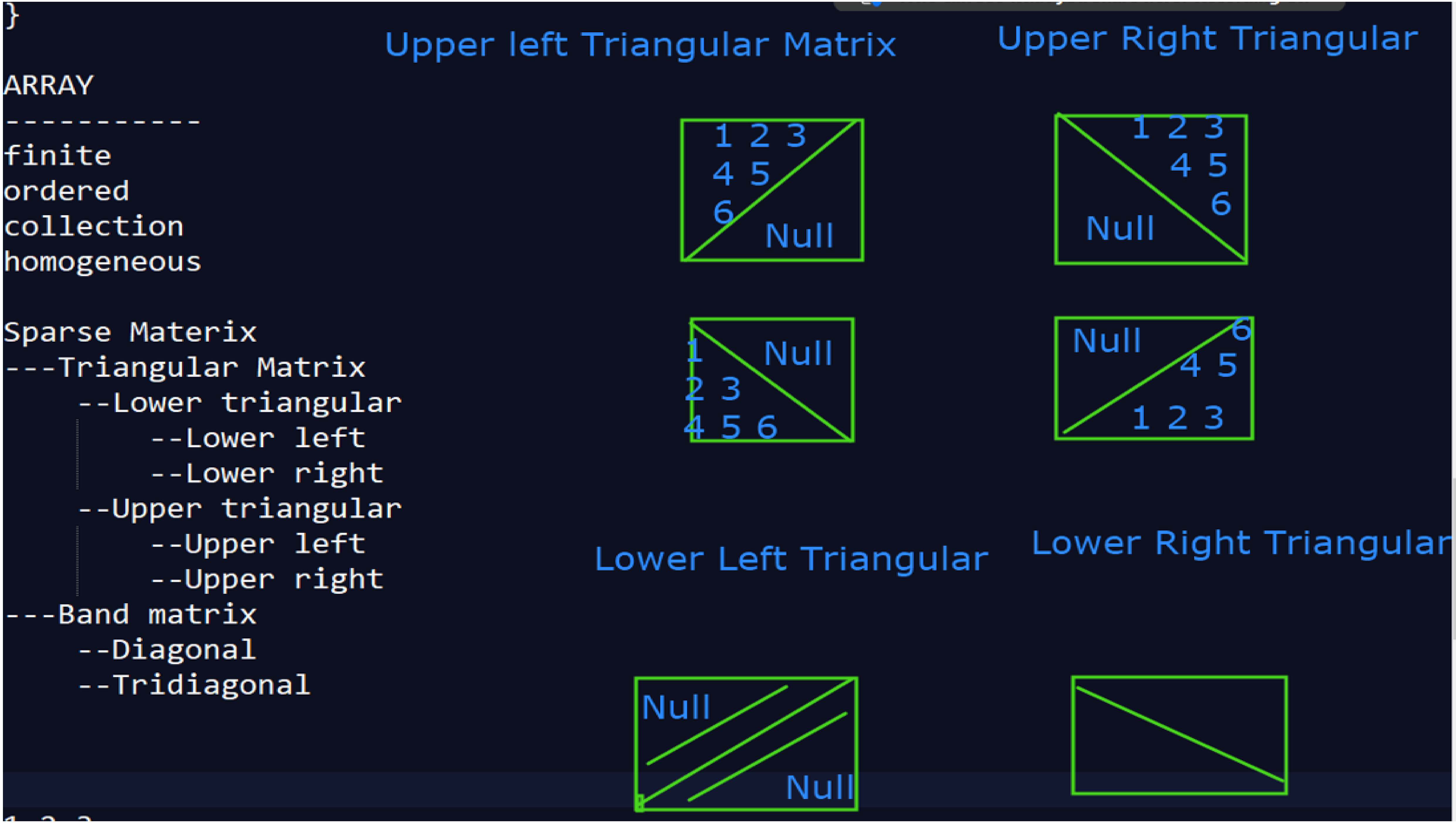
Column Major

a11
a21
a31
a41

a12
a22
a32
a42

a13
a23
a33
a43

a14
a24
a34
a44



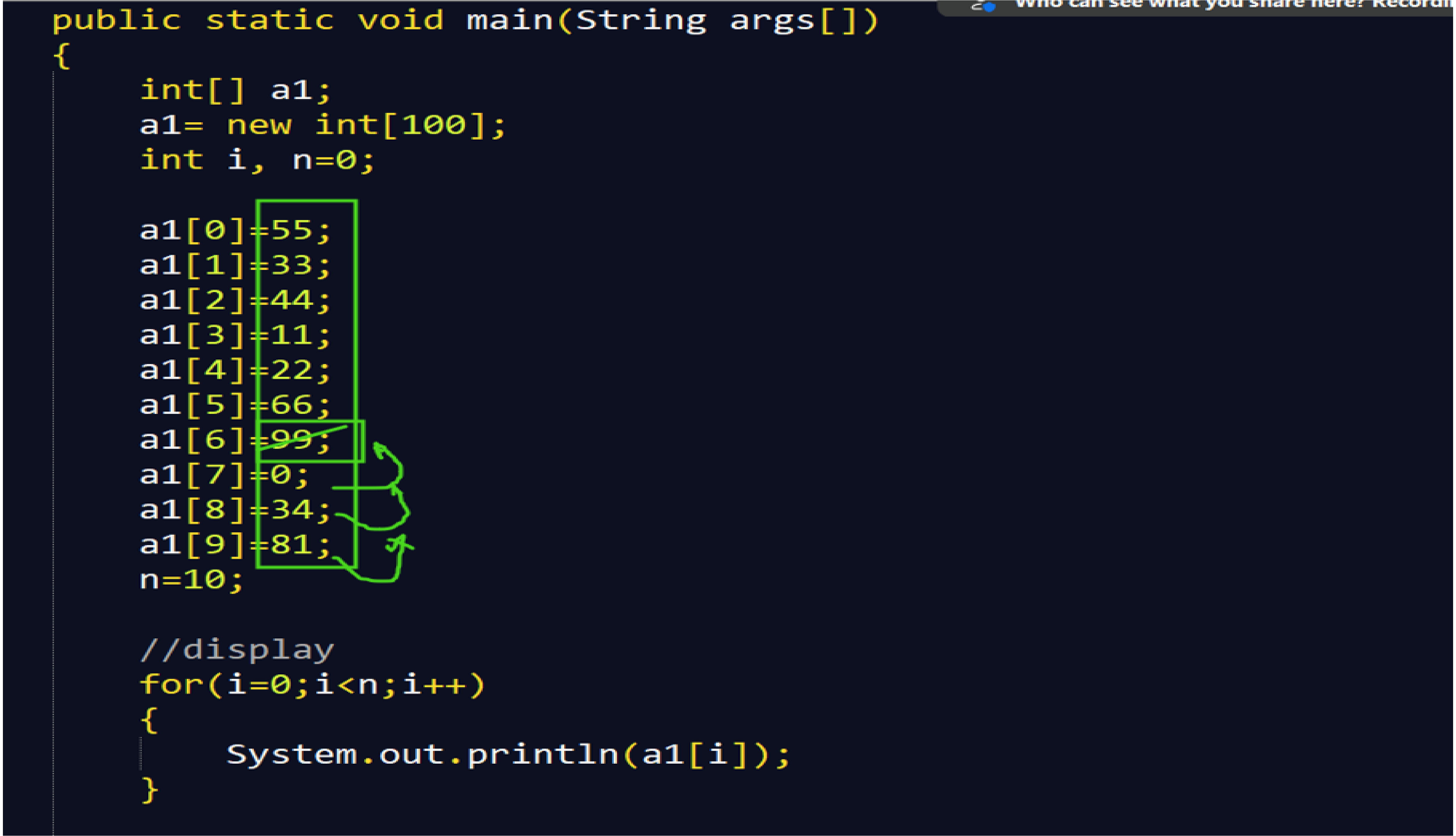
```
}

ARRAY
-----------
finite
ordered
collection
homogeneous

Sparse Materix
---Triangular Matrix
    --Lower triangular
        --Lower left
        --Lower right
    --Upper triangular
        --Upper left
        --Upper right
---Band matrix
    --Diagonal
    --Tridiagonal
```

Upper left Triangular Matrix

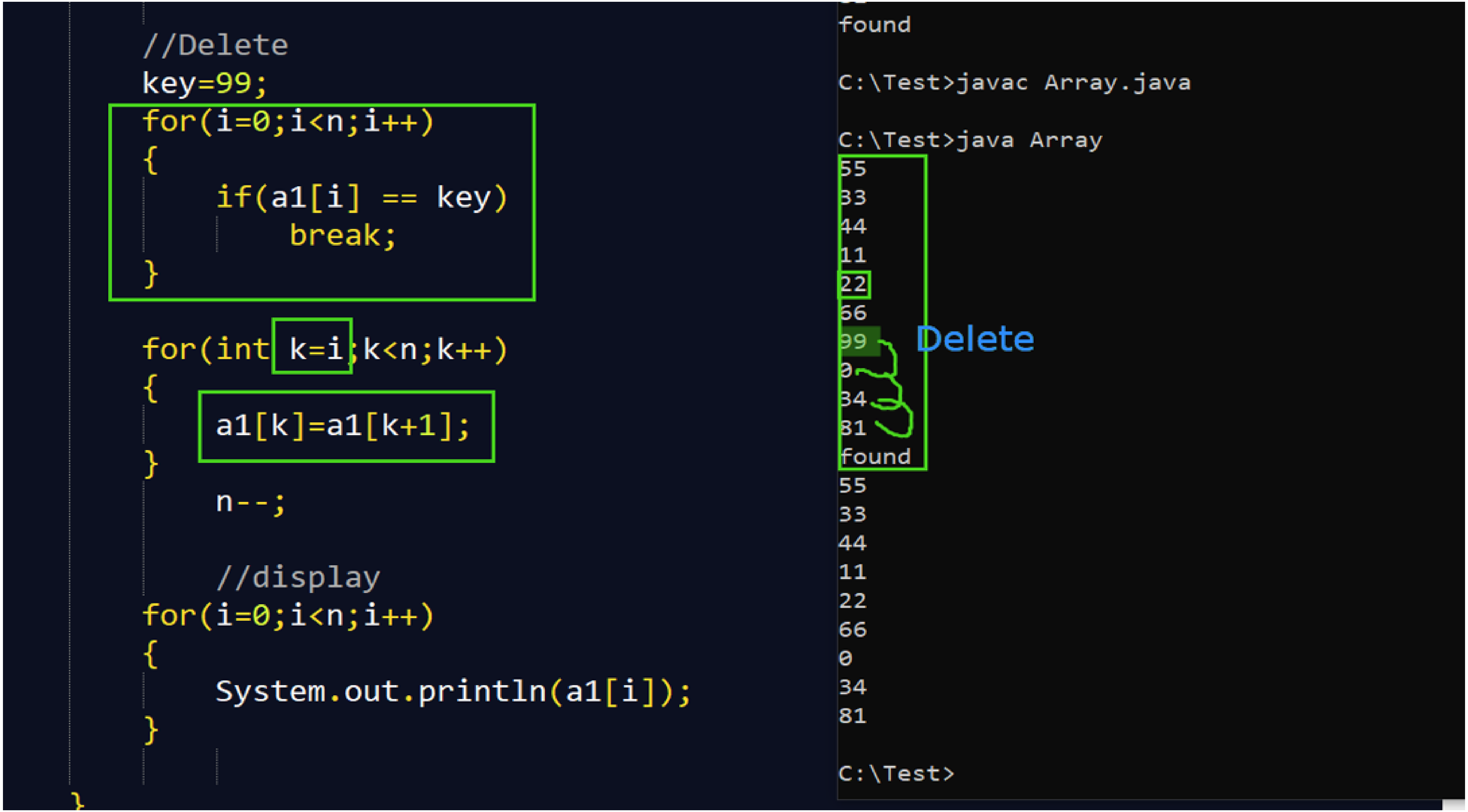
Upper Right Triangular

Lower Left Triangular

Lower Right Triangular



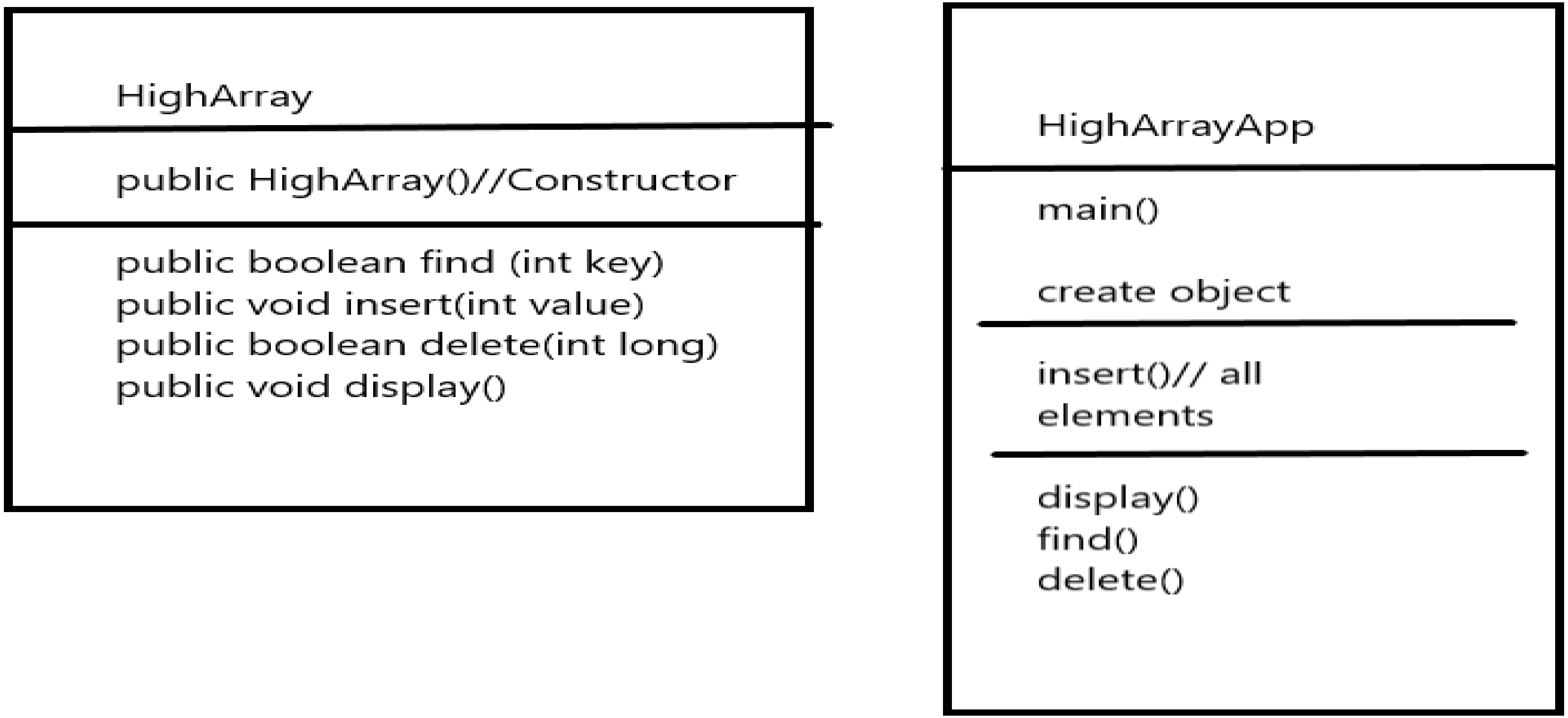
```
public static void main(String args[])
{
    int[] a1;
    a1= new int[100];
    int i, n=0;

    a1[0]=55;
    a1[1]=33;
    a1[2]=44;
    a1[3]=11;
    a1[4]=22;
    a1[5]=66;
    a1[6]=~~99;~~
    a1[7]=0;
    a1[8]=34;
    a1[9]=81;
    n=10;

    //display
    for(i=0;i<n;i++)
    {
        System.out.println(a1[i]);
    }
```



```
    //Delete
    key=99;
    for(i=0;i<n;i++)
    {
        if(a1[i] == key)
            break;
    }

    for(int k=i;k<n;k++)
    {
        a1[k]=a1[k+1];
    }
        n--;

        //display
    for(i=0;i<n;i++)
    {
        System.out.println(a1[i]);
    }

}
```

```
found

C:\Test>javac Array.java

C:\Test>java Array
55
33
44
11
22
66
99
0
34
81
found
55
33
44
11
22
66
0
34
81

C:\Test>
```

Program 2



HighArray

public HighArray()//Constructor

public boolean find (int key)
public void insert(int value)
public boolean delete(int long)
public void display()

HighArrayApp

main()

create object

insert()// all
elements

display()
find()
delete()