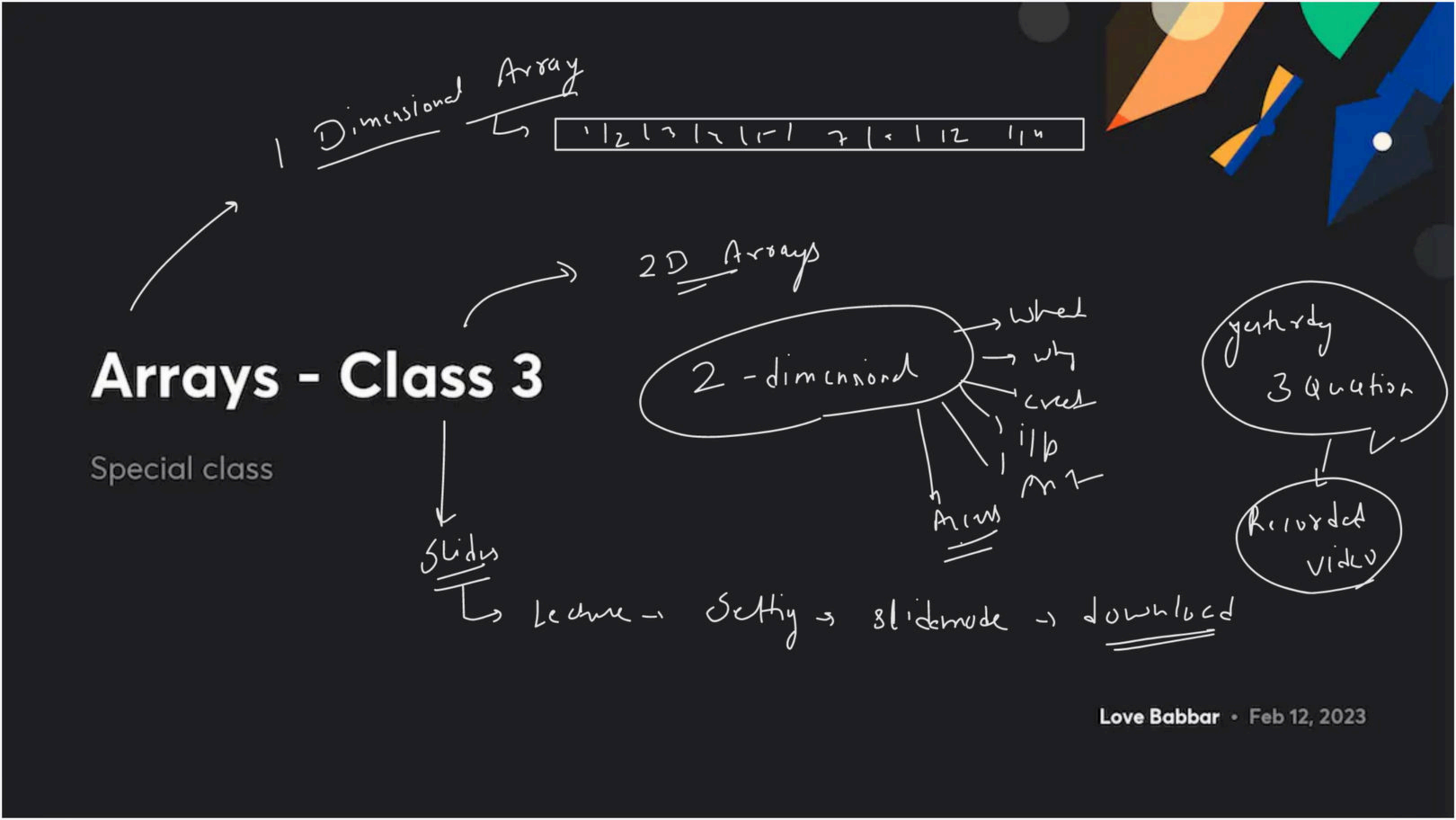


Love Babbar • Feb 12, 2023

①

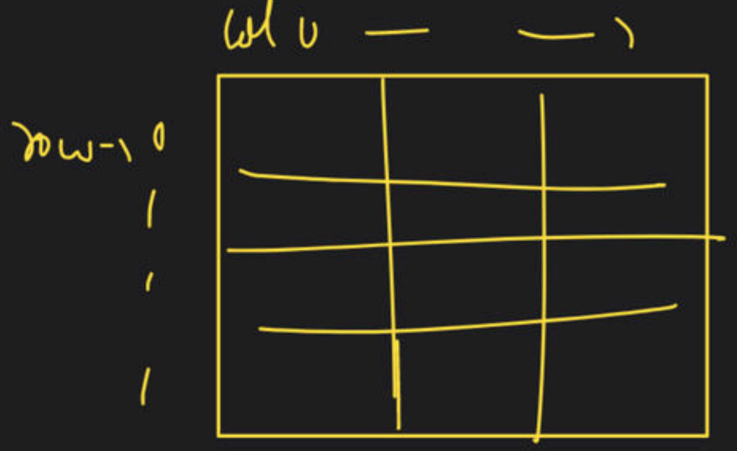
```
int arr[] = {1, 2, 5, 7};
```

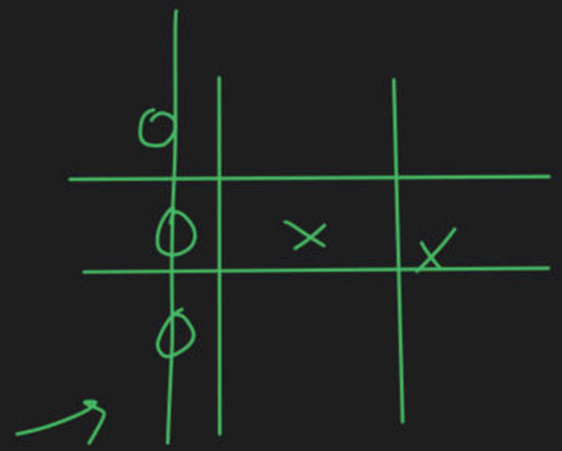
1	2	5	7

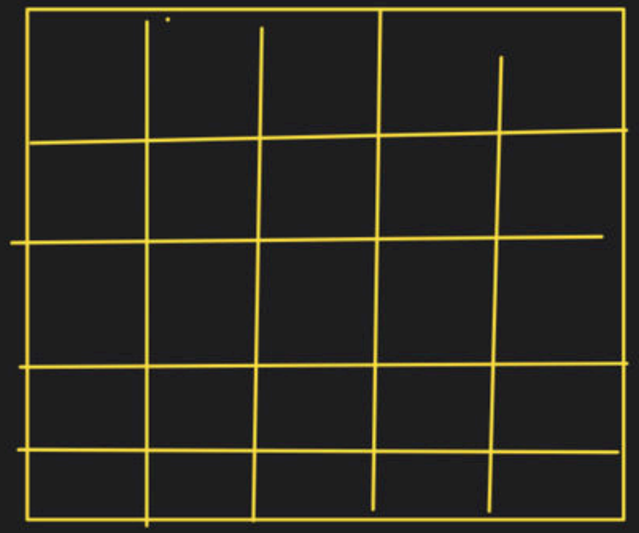
```
int a = 5;
```

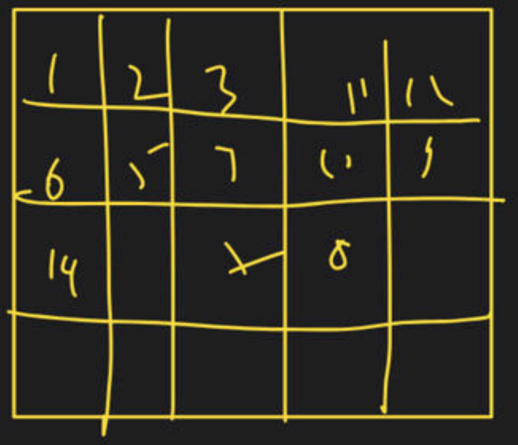

```
int arr[4];
```

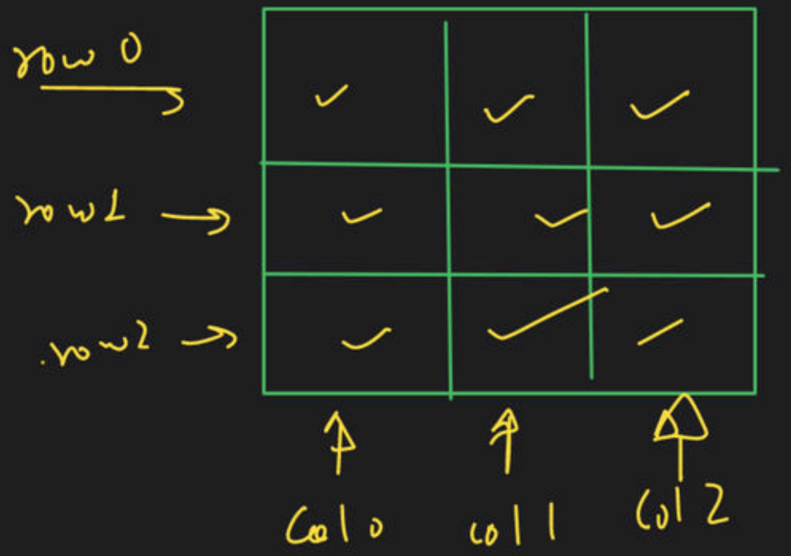
2-D Array

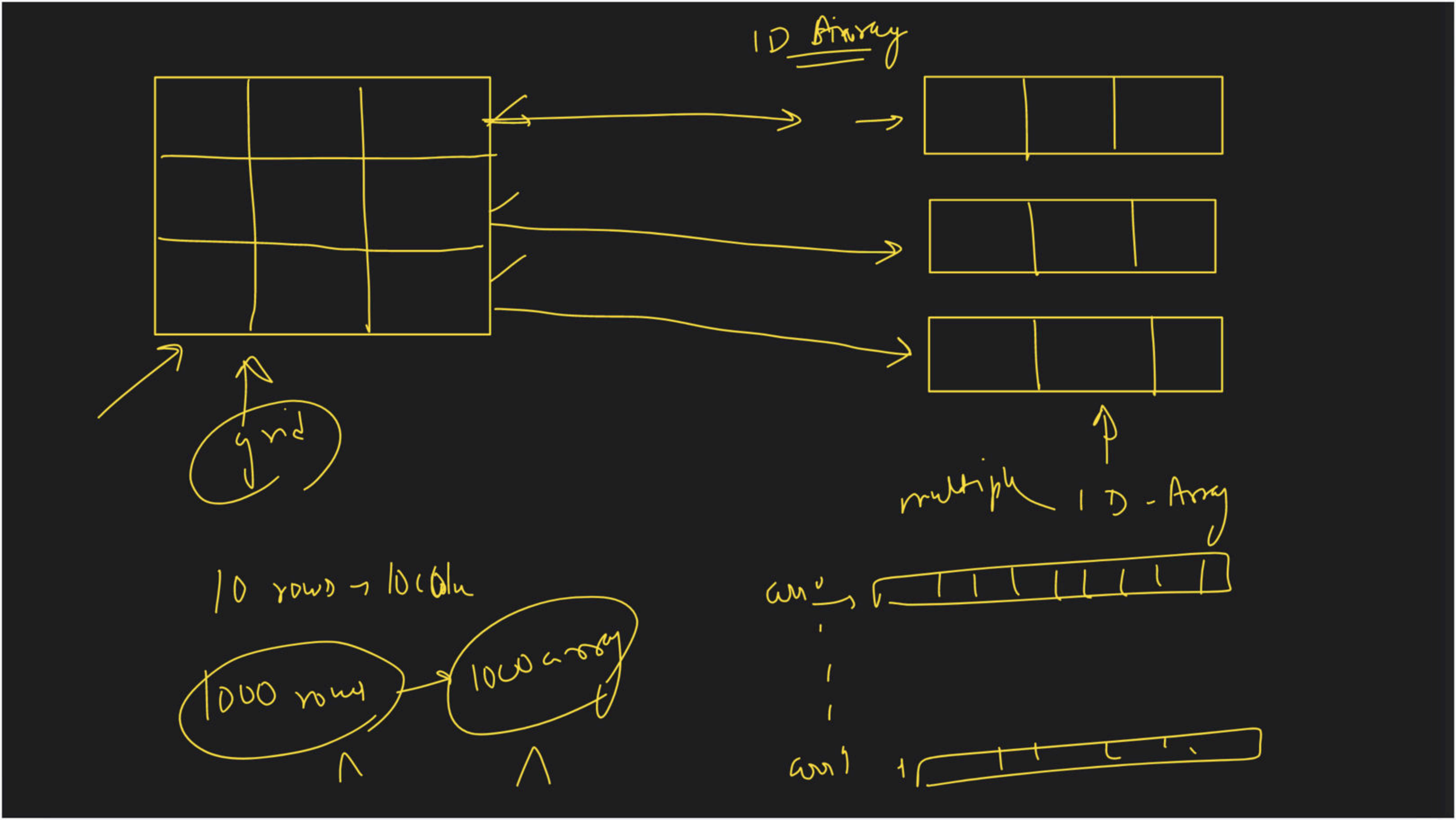




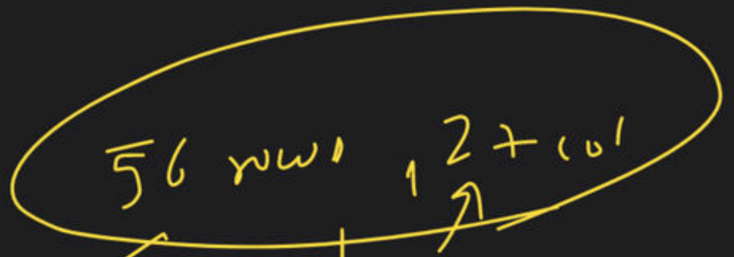






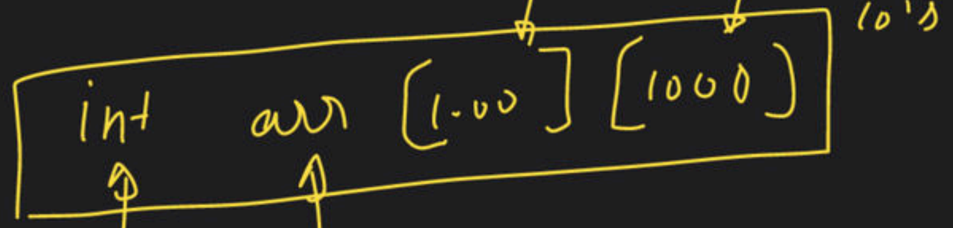

1D Array
grid
multiple 1D-Array
10 rows → 10 Cols
1000 rows
1000 array

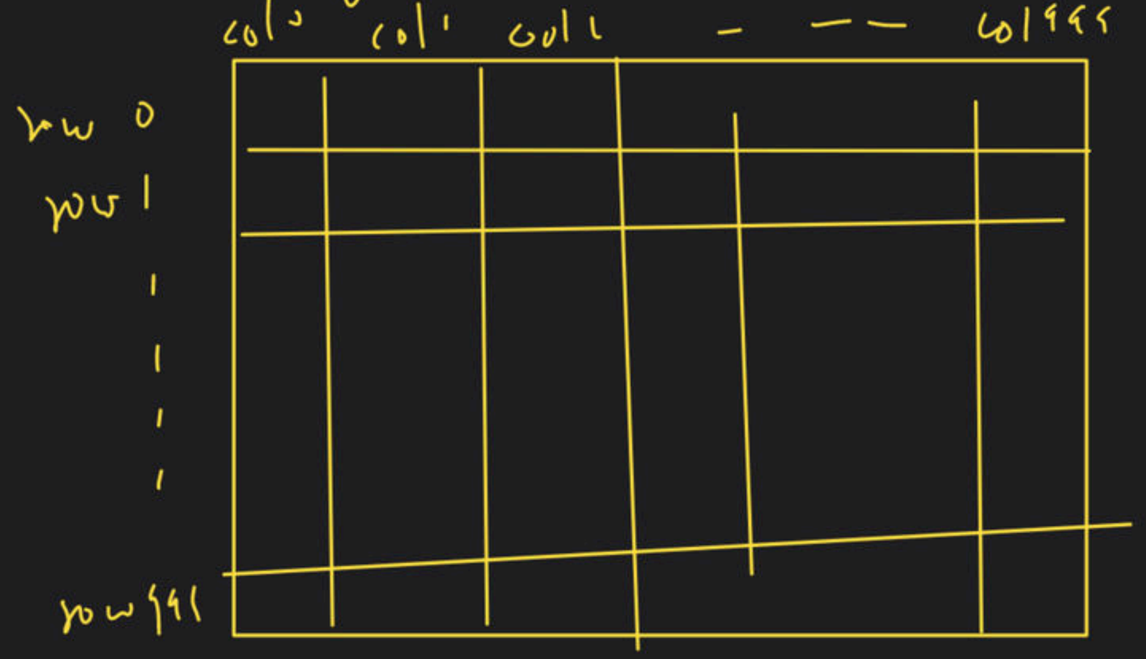
100 row, 1000 col ⟶ 2D array



2D array?

int arr [56] [27]





int arr[4] = {1, 2, 3, 4}

total no. of element = 3 × 3 = 9

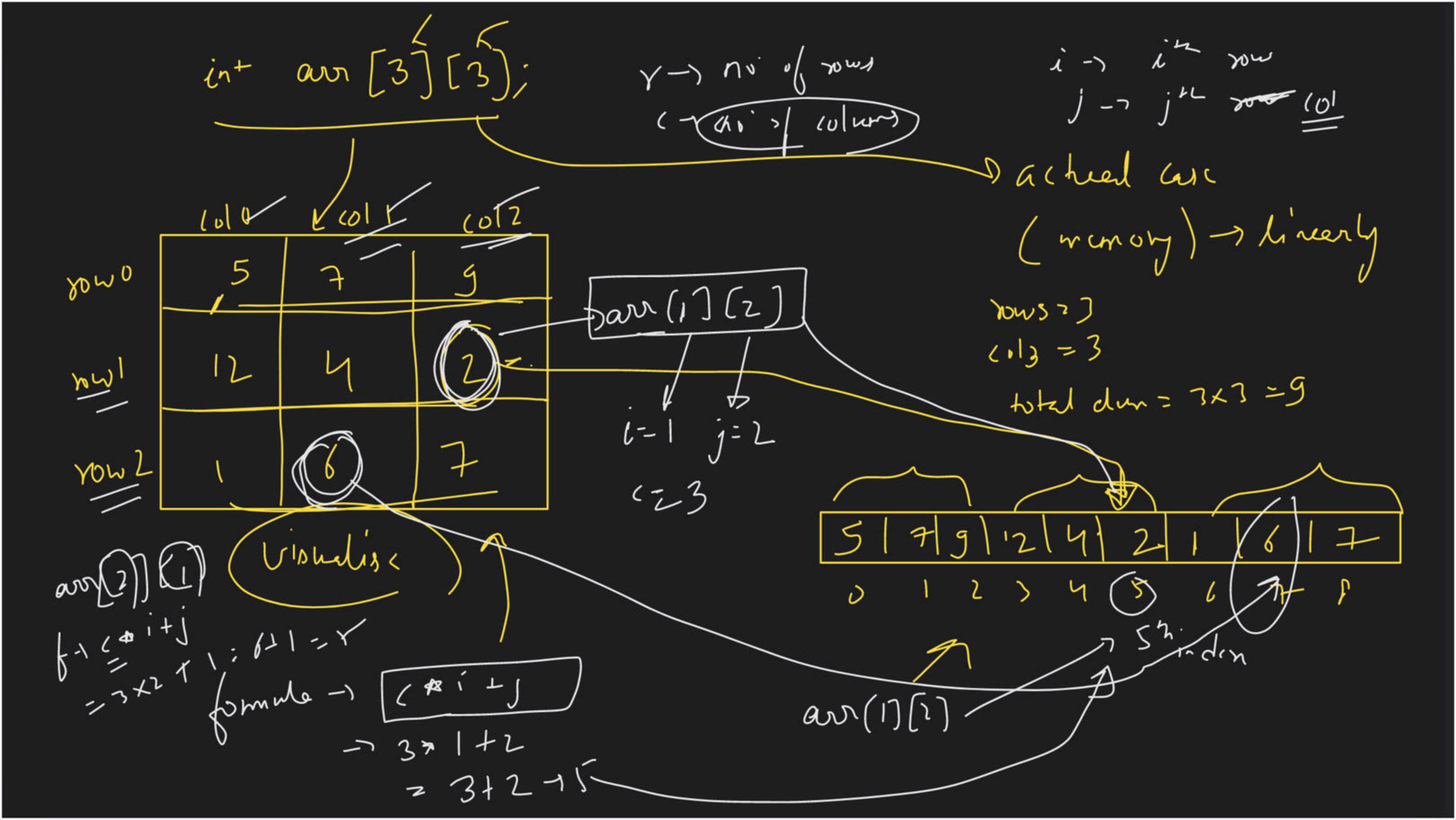
int arr[3][3] ...

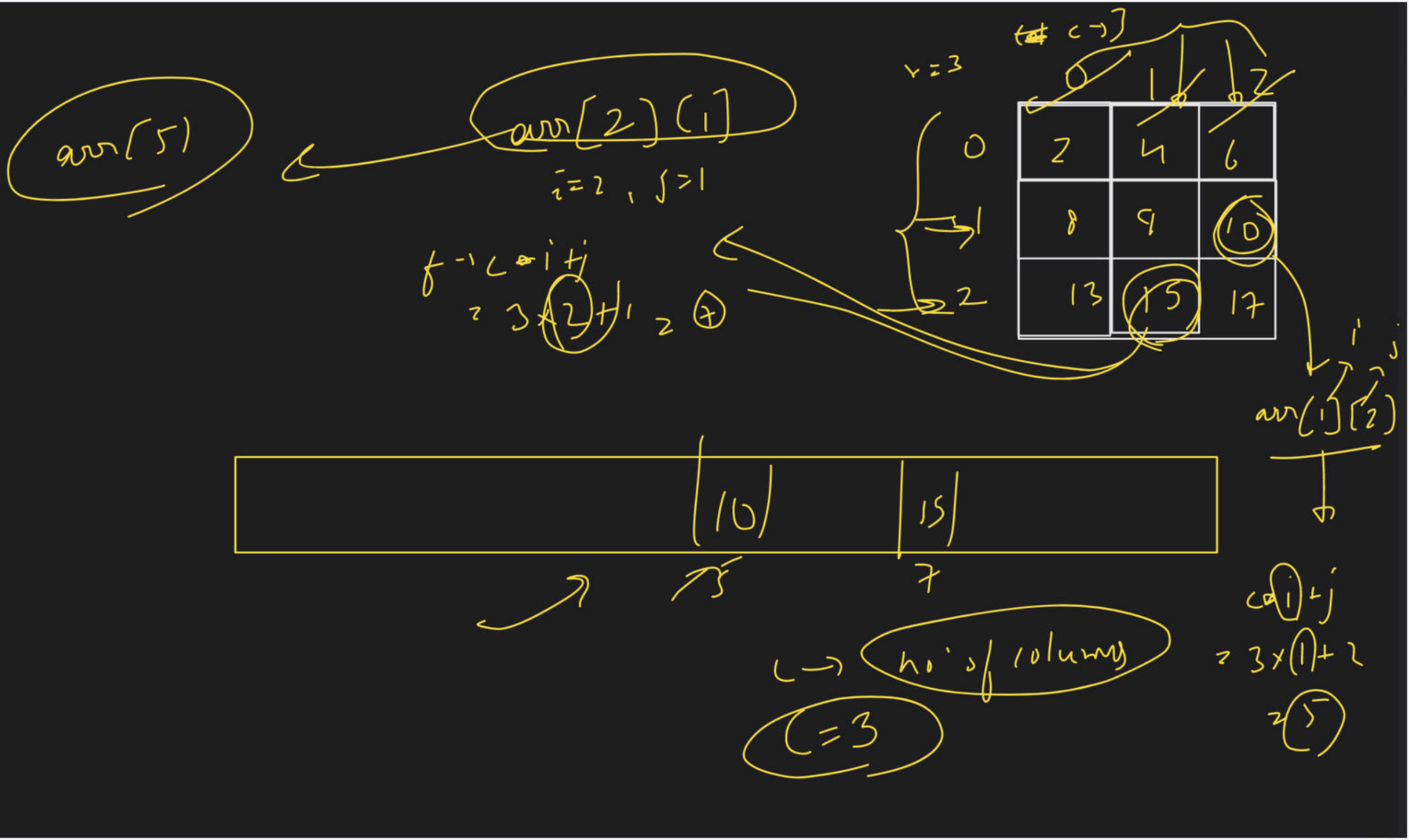
arr

2D arr

arr

visualize


int arr[3][3];
r -> no of rows
c -> no of columns
i -> ith row
j -> jth col
actual case (memory) -> linearly
col0
col1
col2
row0
row1
row2
5
7
9
12
4
2
1
6
7
arr[1][2]
i=1 j=2
<=3
rows=3
cols=3
total elem = 3x3 = 9
5 | 7 | 9 | 12 | 4 | 2 | 1 | 6 | 7
0 1 2 3 4 5 6 7 8
arr[2][1]
visualise
f-1 <* i+j
= 3x2 + 1 ; f+1 = 7
formula -> <* i + j
-> 3*1+2
= 3+2 -> 5
arr[1][2]
5th index

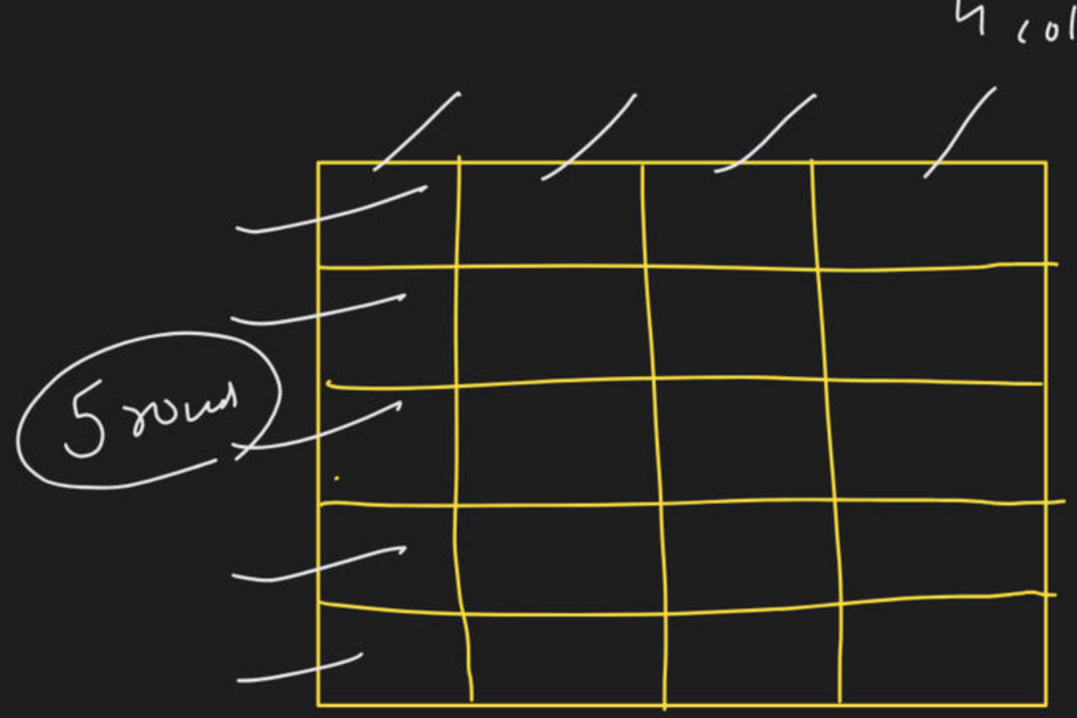

arr[5]
arr[2][1]
i=2, j=1
r=3
0
1
2
0 1 2
2 4 6
8 9 10
13 15 17
f-1(i,j)
= 3*(2)+1 = 7
arr[1][2]
i
j
10
15
7
(i,j)
= 3x(1)+2
= 5
no. of columns
C=3

what is 2D array

Why 2D

How it is stores in memory

Visualize



Access

arr[i][j]

i → row index

j → col index

(i * c + j)

c → no of columns

i → i^th row

j → j^th col

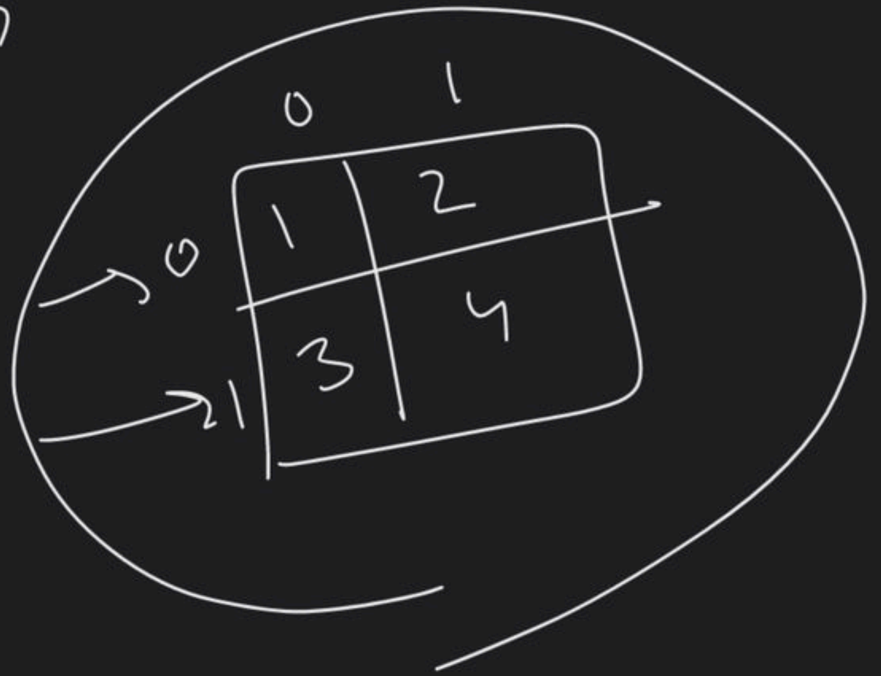
Creation:-

int arr[2][2] = { {1,2} {3,4} }

declaration

int arr[4][4];

init



min 3 rows

i/p

o/p

2D array → i/p

```
cin >> arr[i][j];
```

1D array → i/p

```
cin >> arr[i]
```

o/p →

```
cout << arr[i][j];
```

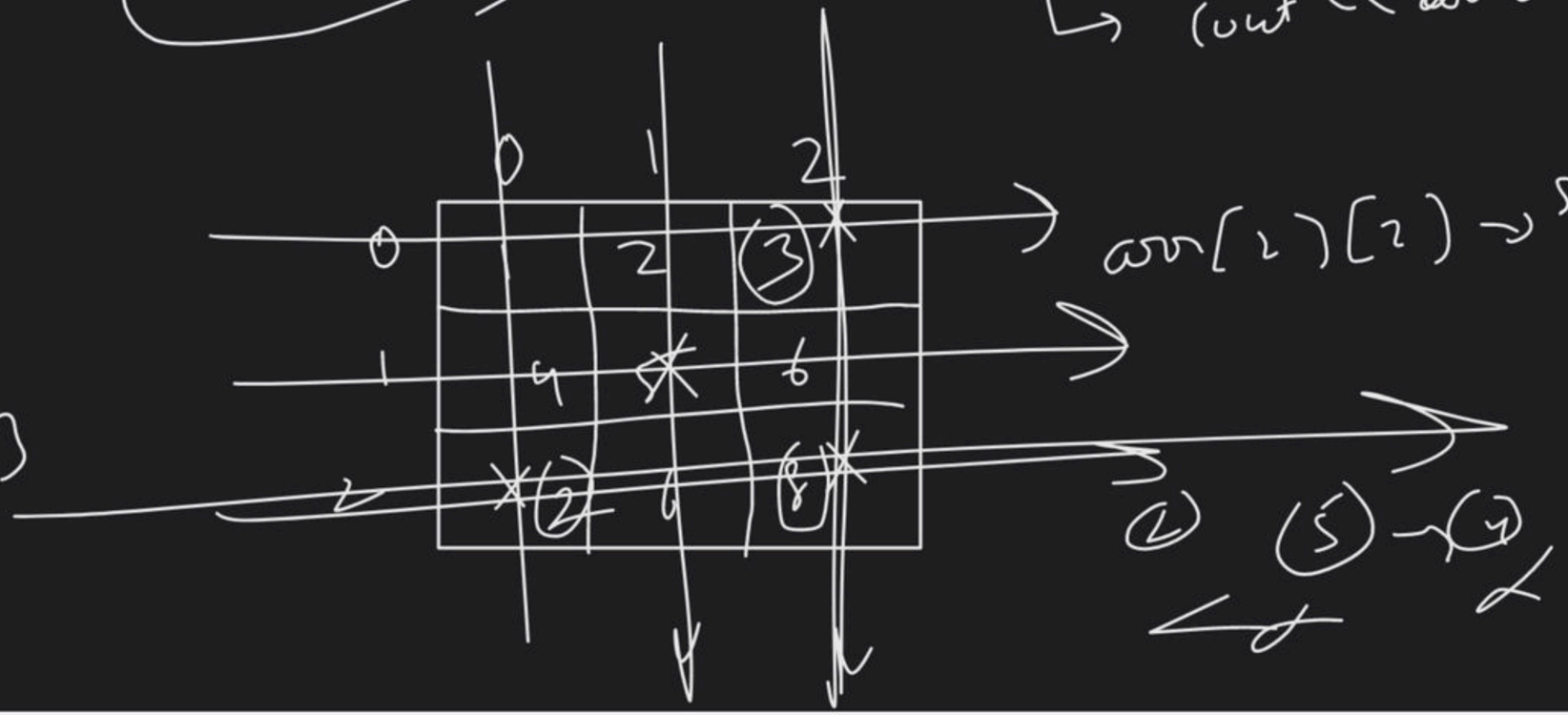
1D array → o/p

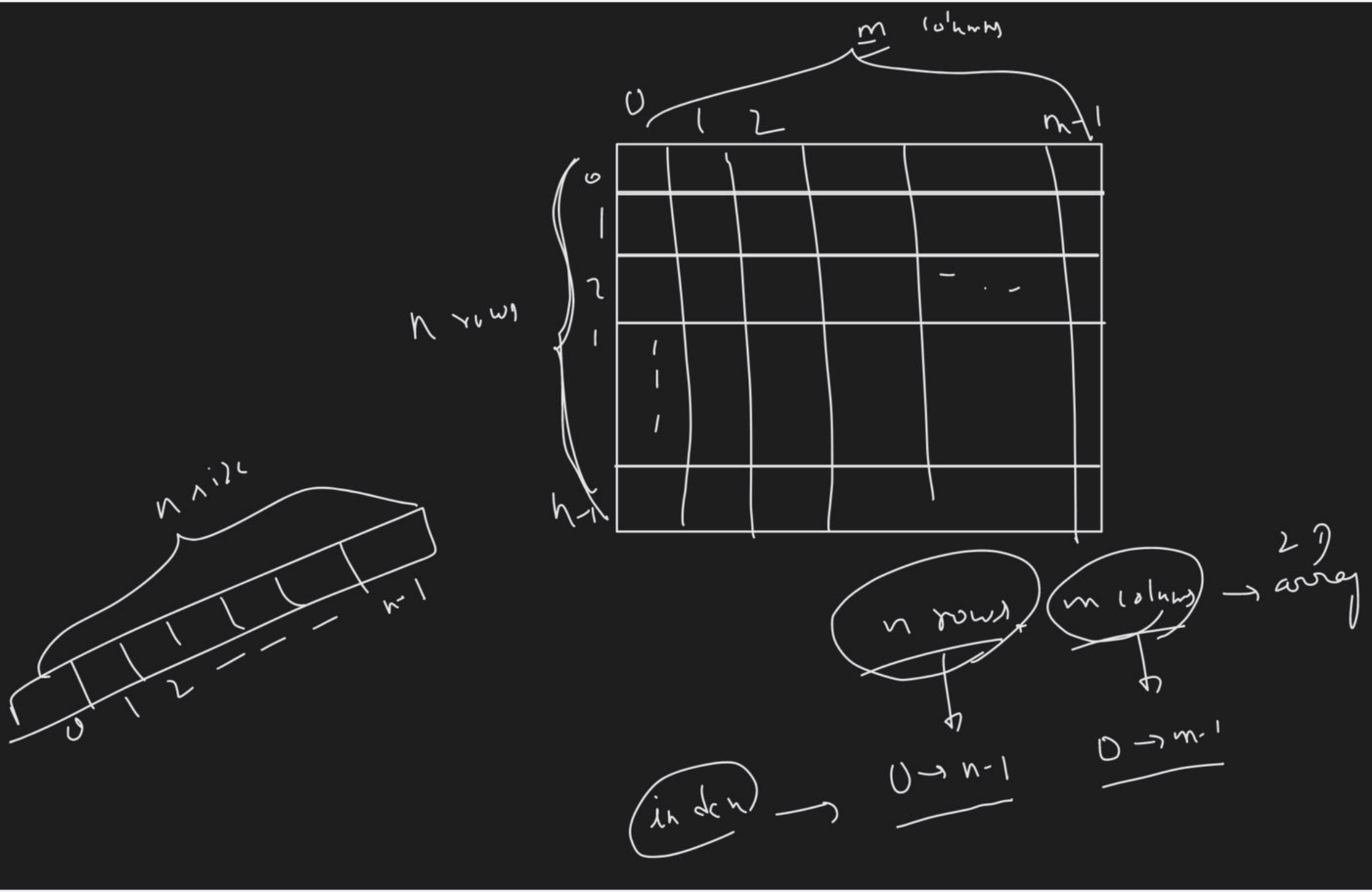
```
cout << arr[i];
```

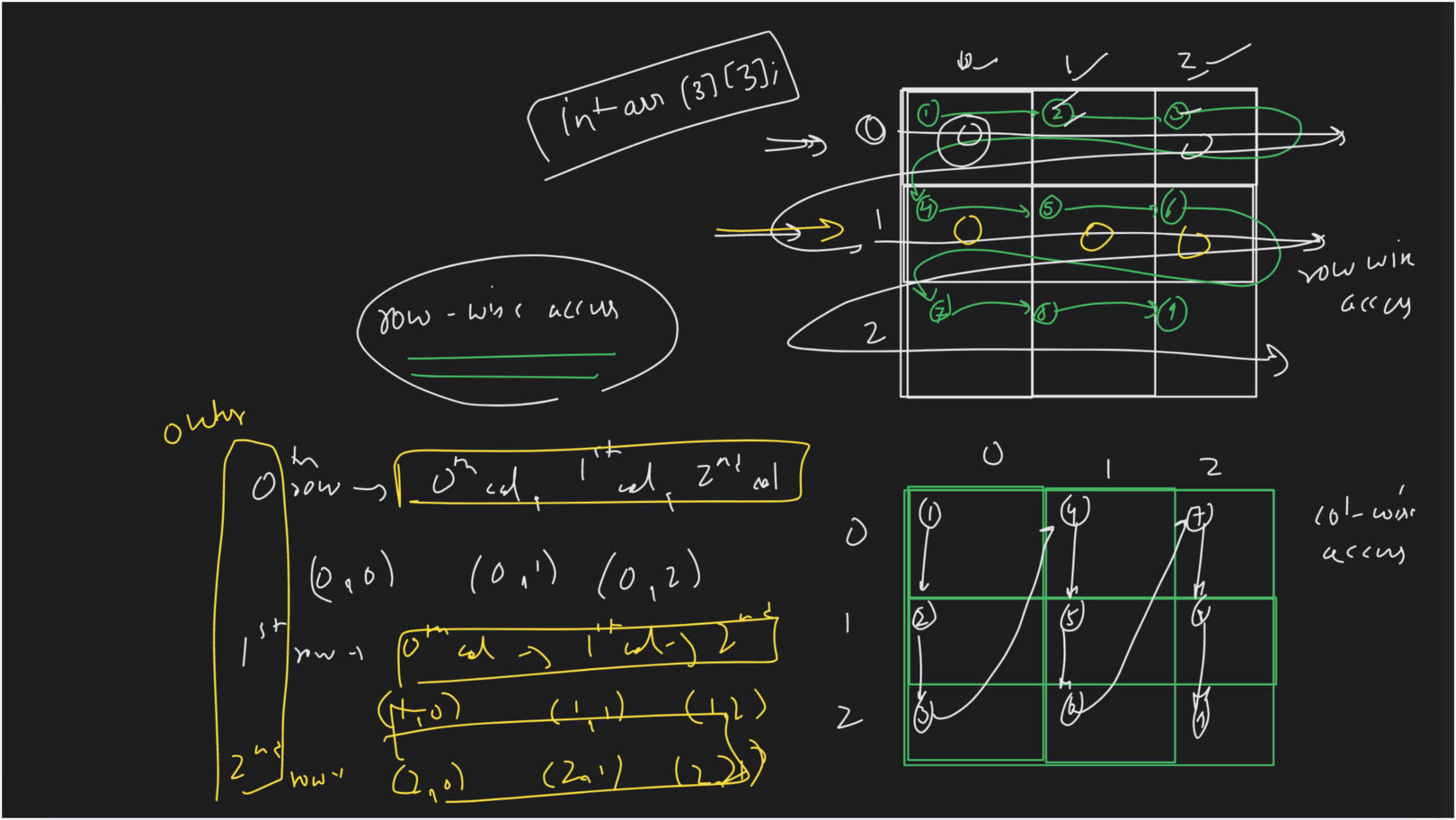
arr[2][0] = 2

arr[1][1] = 5

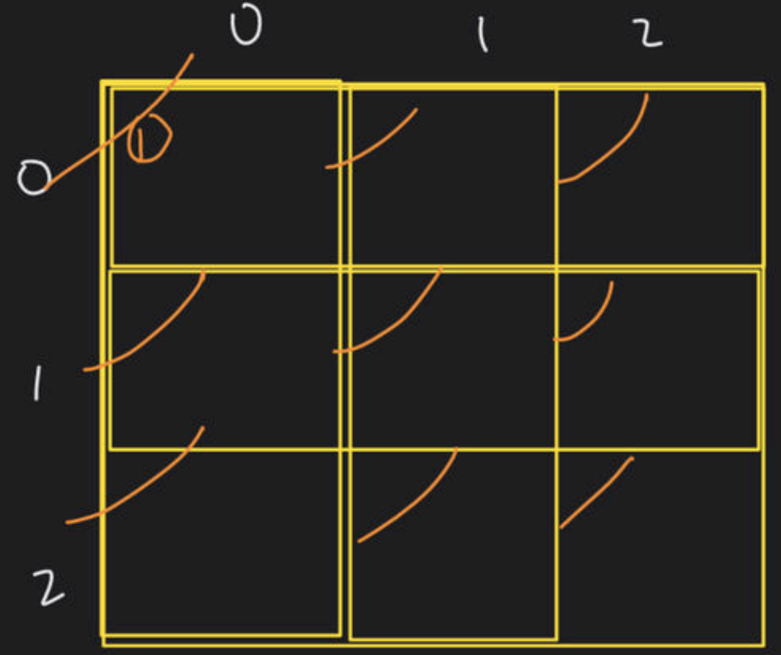
arr[0][2] = 3

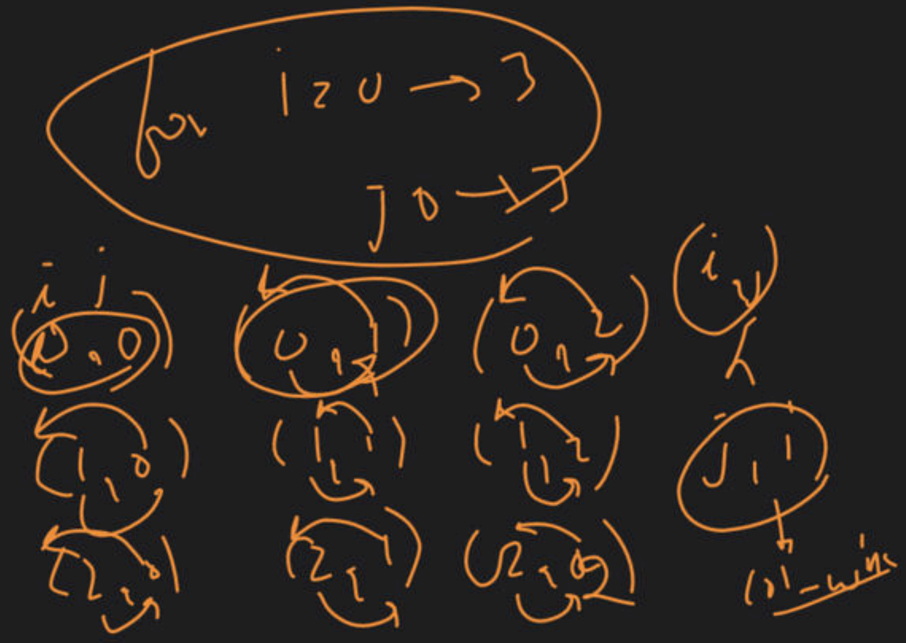


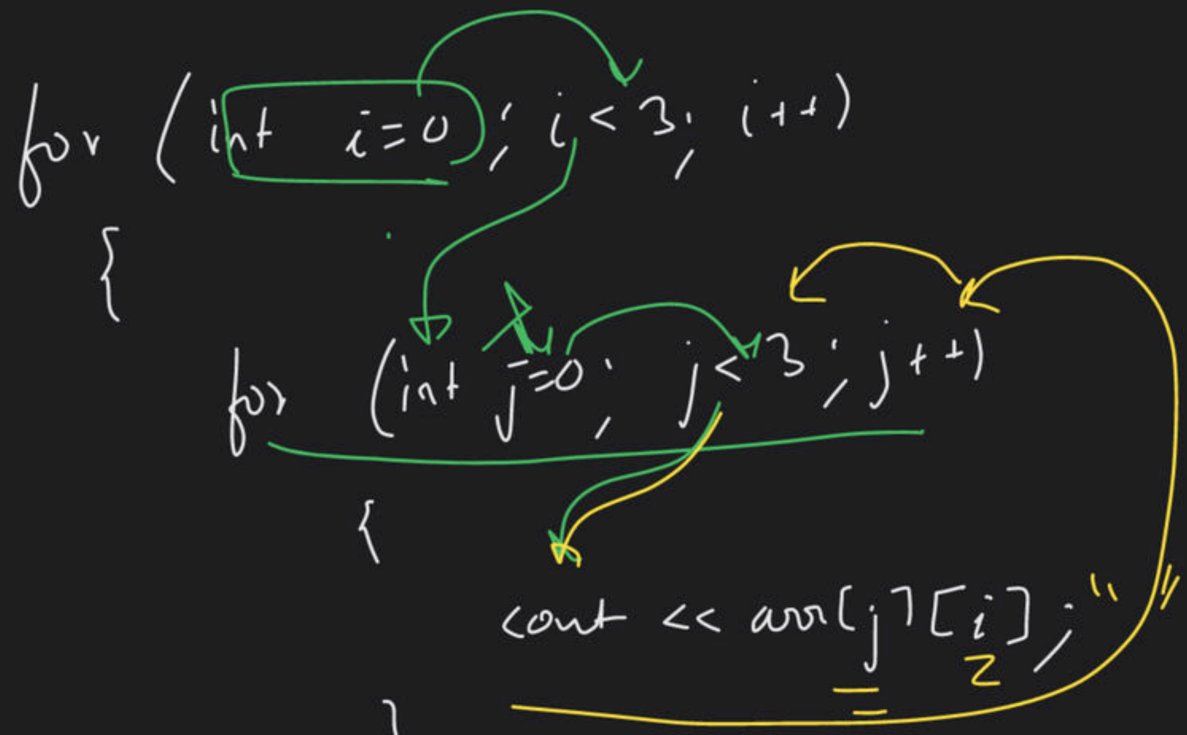

m columns
0 1 2 m-1
n rows
0
1
2
n-1
n size
0 1 2 n-1
n rows
m columns
2-D array
0 → n-1
0 → m-1
index → 0 → n-1

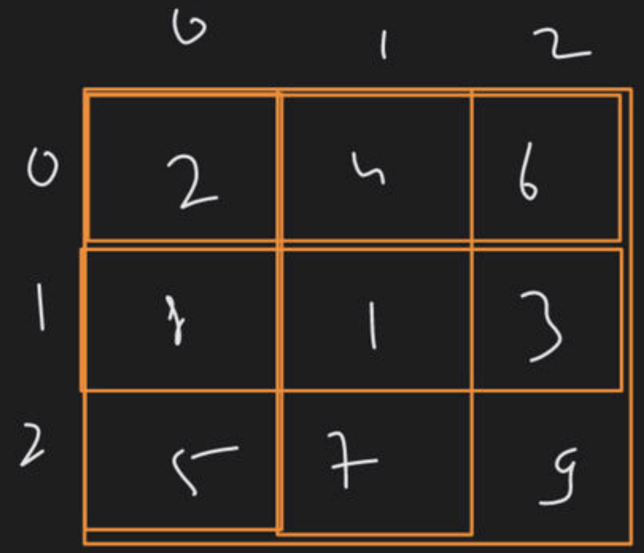

int arr[3][3];
row - wise access
row wise access
column wise access


(0,0)
(1,0)
(2,0)
(0,1)
(1,1)
(2,1)
(0,2)
(1,2)
(2,2)


0
1
2
0
1
2


i = 0 → 3
j = 0 → 3
i
j
(0,0)
(0,1)
(0,2)
(1,0)
(1,1)
(1,2)
(2,0)
(2,1)
(2,2)





i=0
j=0
print arr[0][0]
j=1
print arr[1][0]
j=2
print arr[2][0]

i=1
j=0
print arr[0][1]
j=1
print arr[1][1]
j=2
print arr[2][1]

i=2
j=0
arr[0][2]
j=1
arr[1][2]
j=2
arr[2][2]

→ declare → int arr[5][6];

→ initialize → int arr[2][2] = {{1,2},{3,4}};

→ print → 2D array →

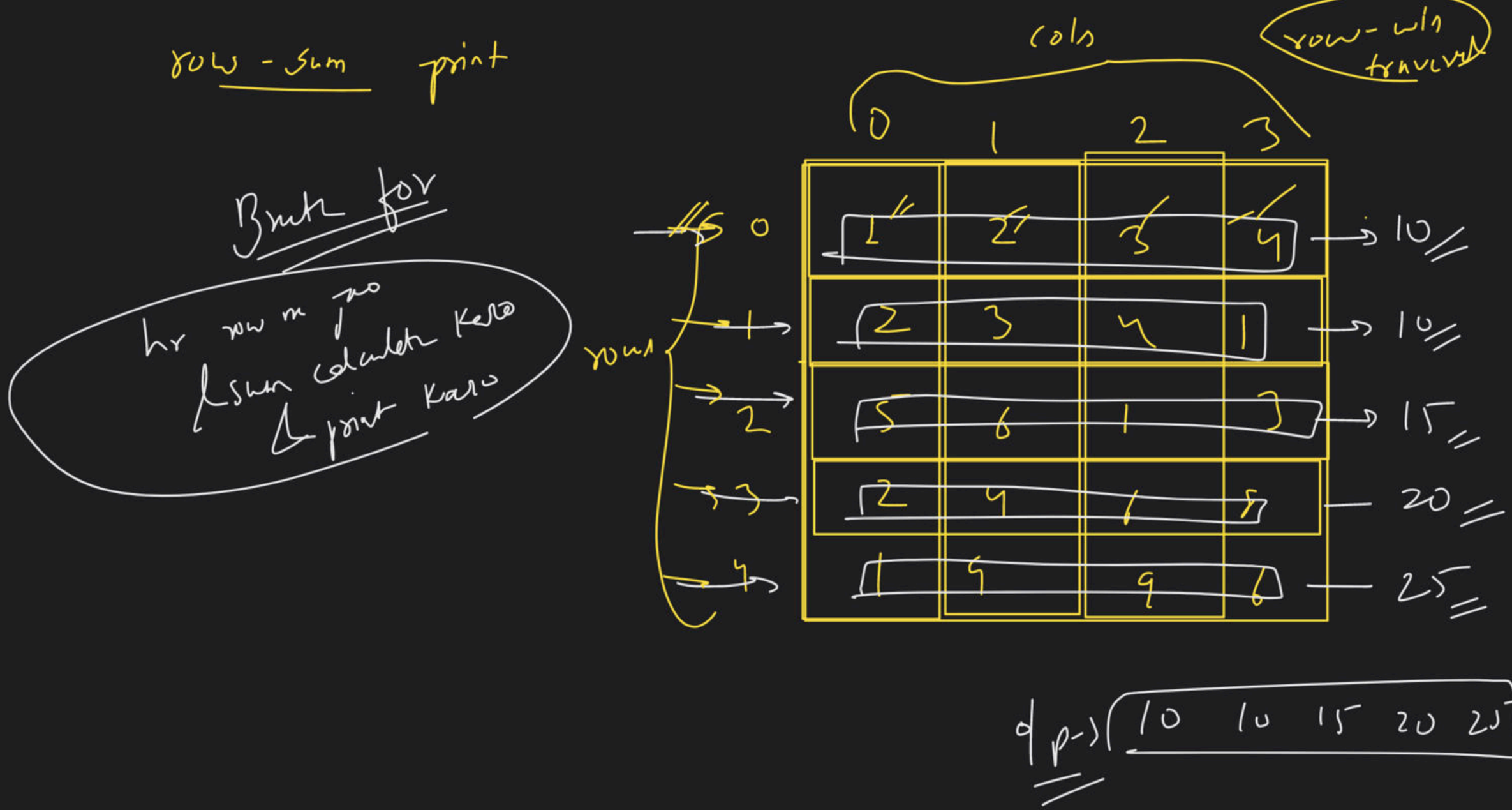
```
for( i -> 0-> <rows)
{
    for(j -> 0 -> <cols)
    {
        cout <<arr[i][j];
    }
}
```

→ i/p → 2D array

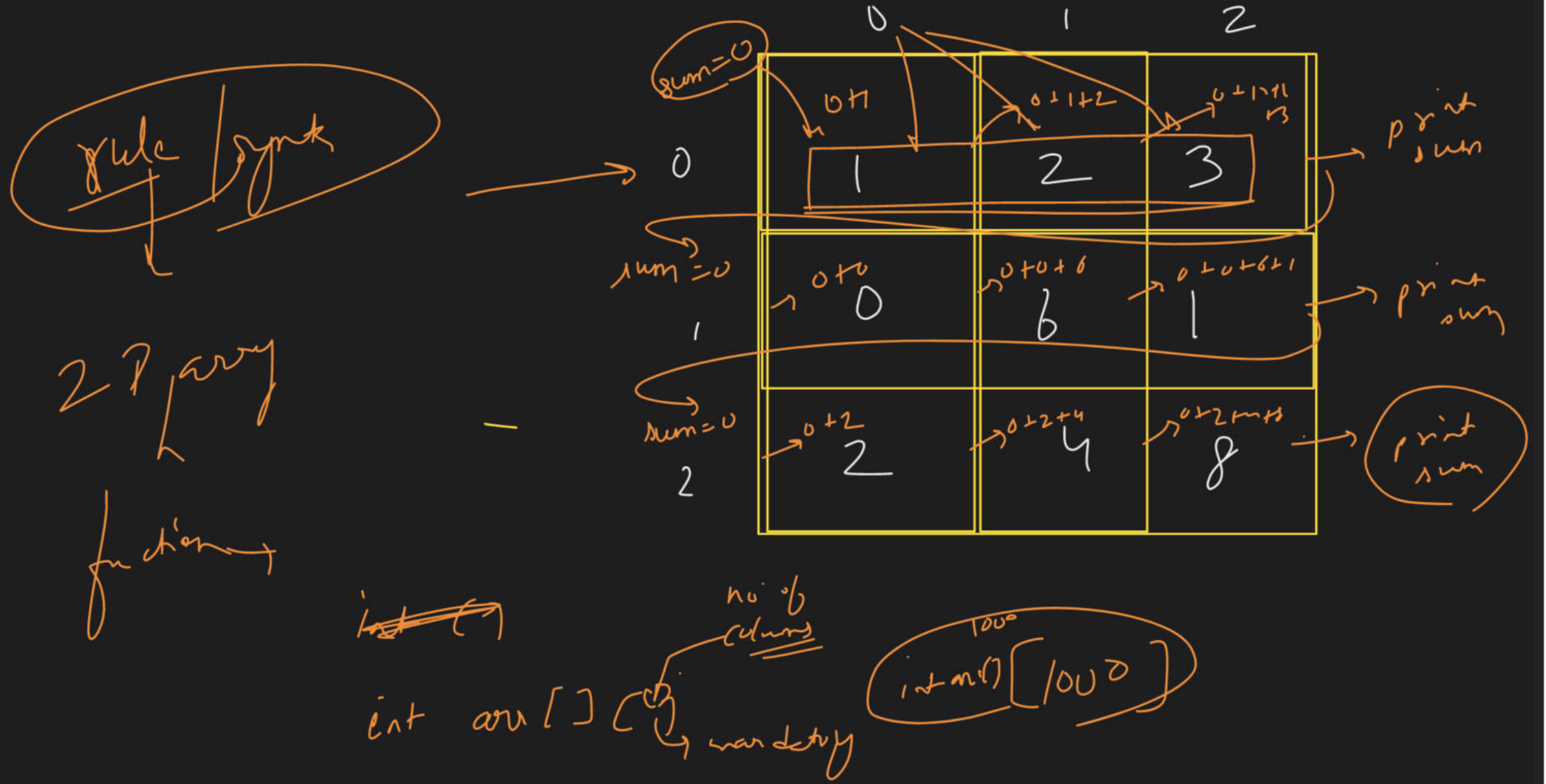
row - sum print

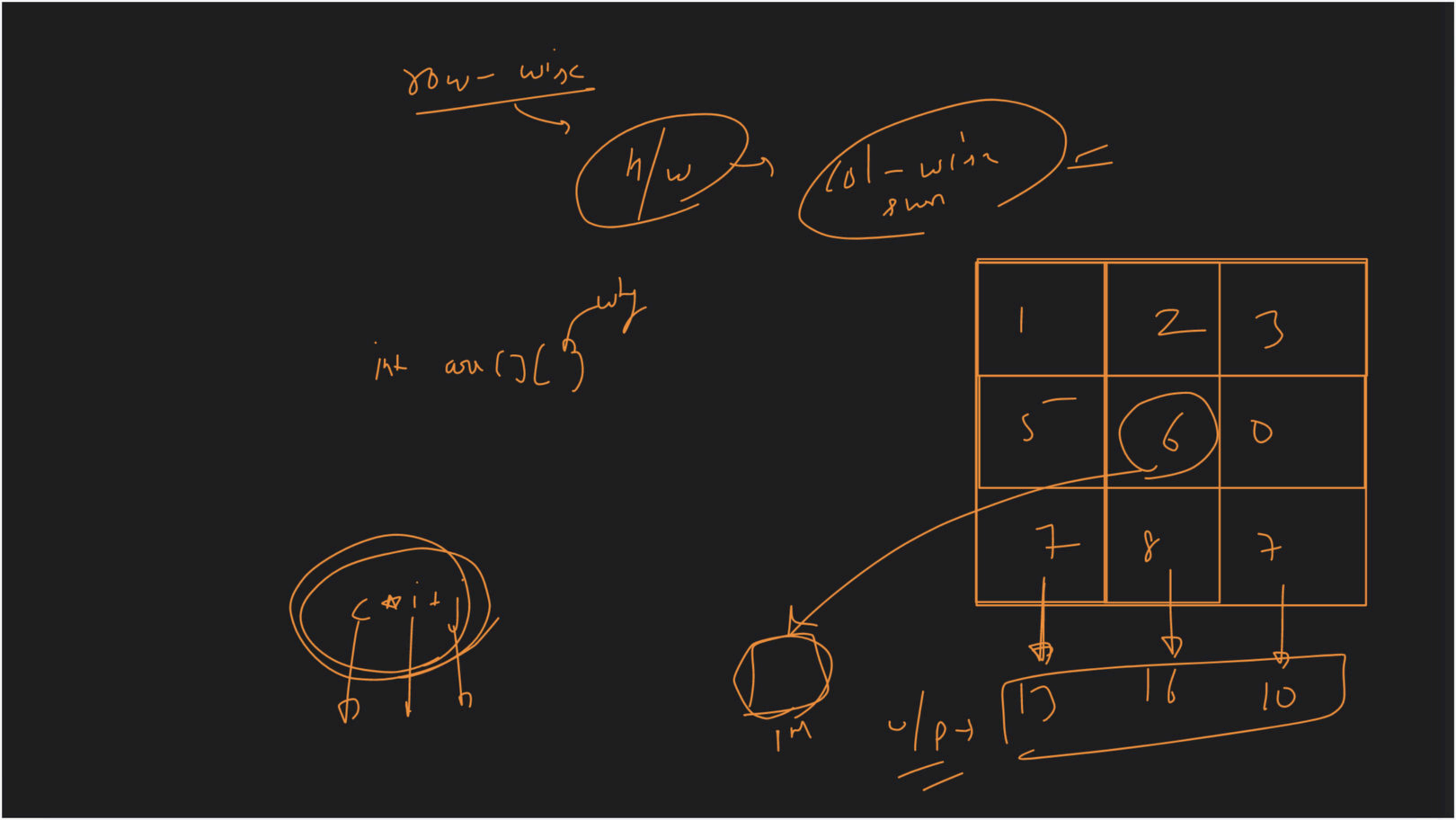
Brute for

har row m jao
sum calculate karo
& print karo

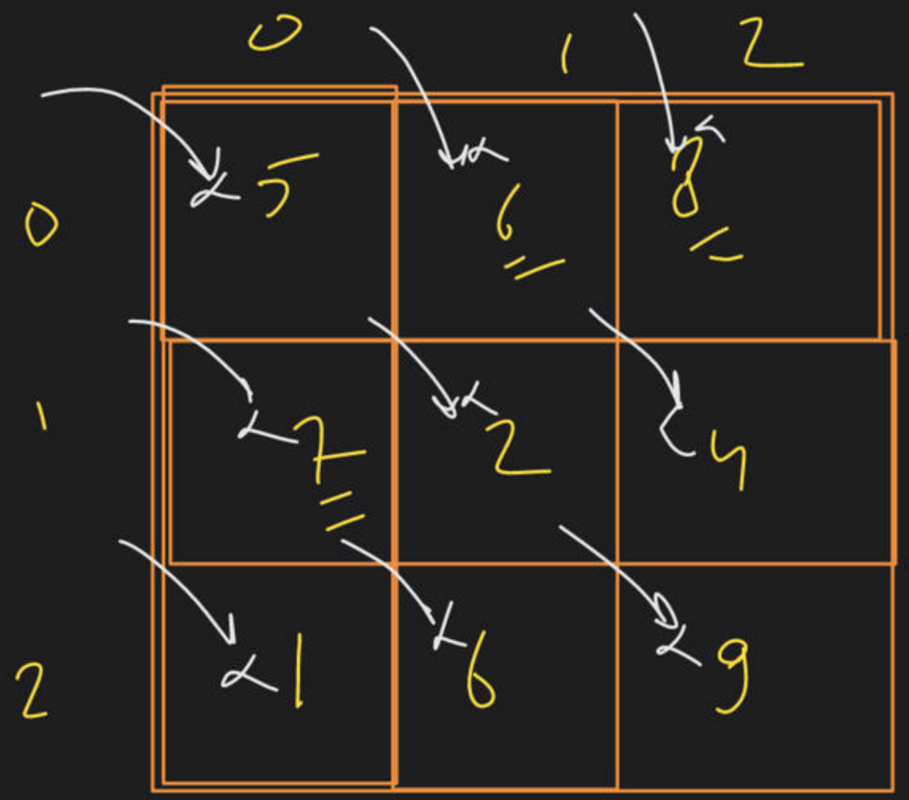


o/p :- 10 10 15 20 25


rule / syntax
2D array
function
0
1
2
sum=0
0+1
0+1+2
0+1+2+3
1
2
3
print sum
sum=0
0+0
0+0+6
0+0+6+1
0
6
1
print sum
sum=0
0+2
0+2+4
0+2+4+8
2
4
8
print sum
int arr[][]
no of columns
mandatory
100
int arr[][100]


row - wise
h/w
col - wise
sum
int arr[][3]
1
2
3
5
6
0
7
8
7
i
17
16
10
o/p

Linear Search

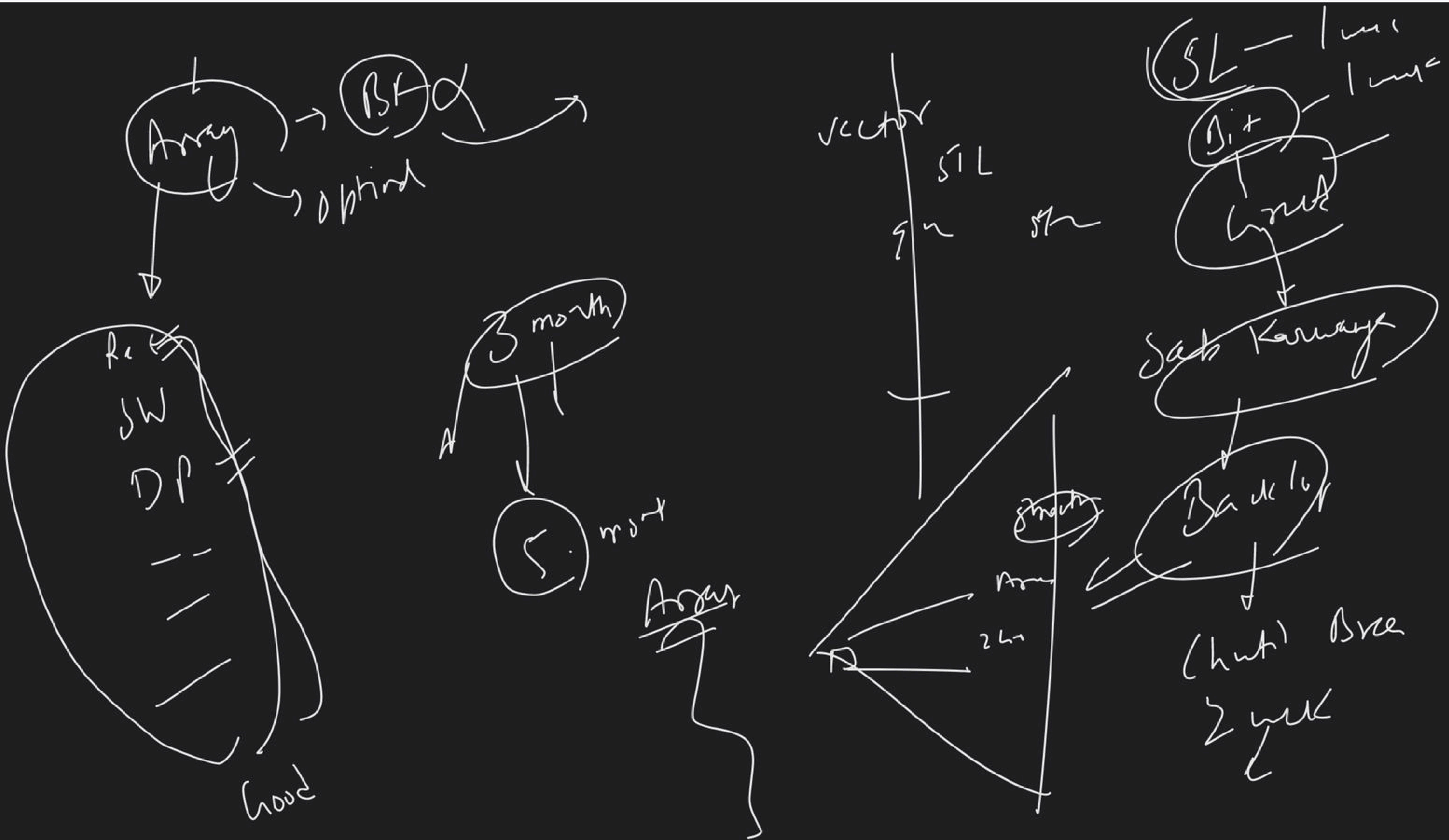


(12) = Key

No. (7)

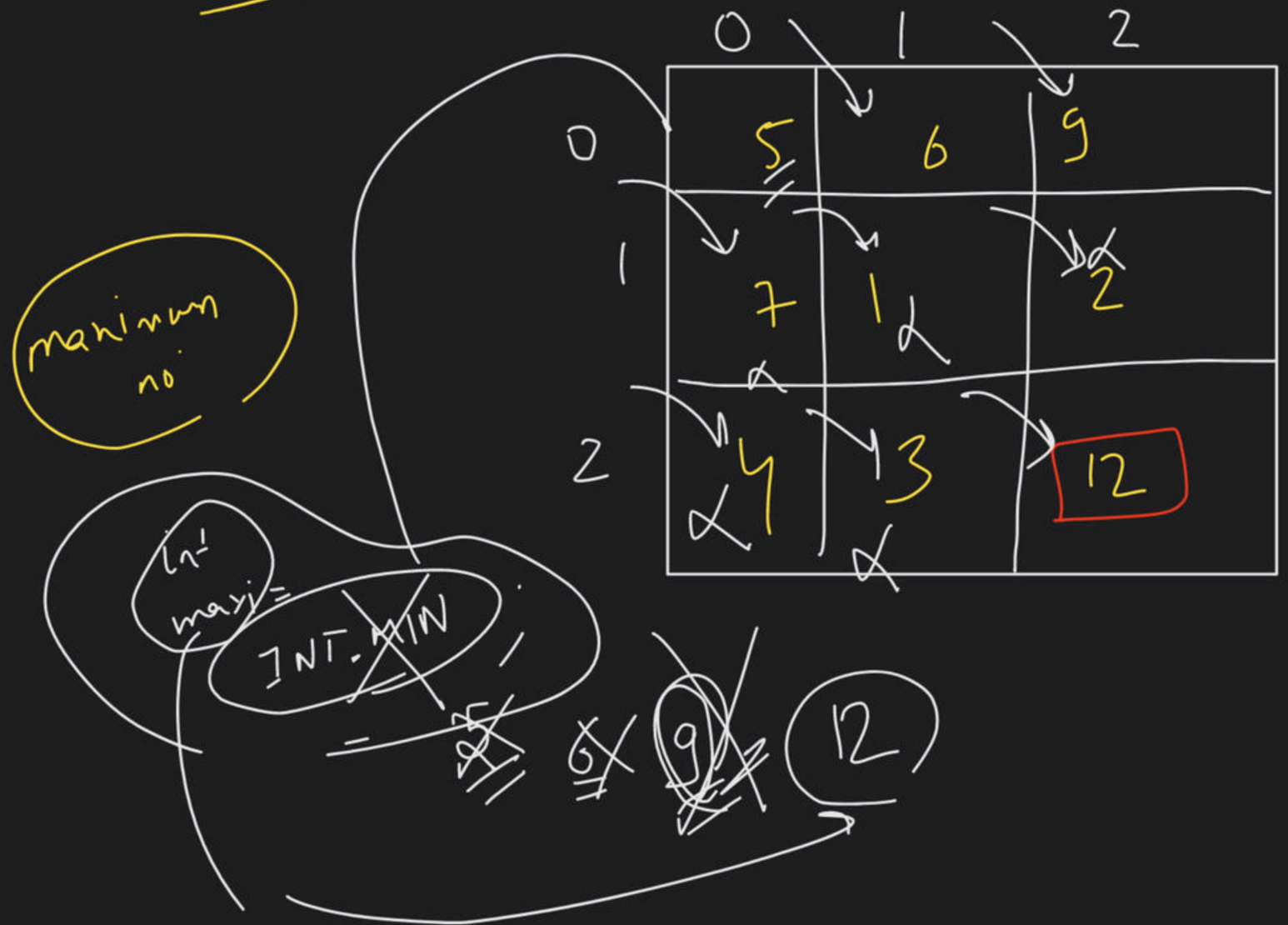
search ( ) → T F

(8)



Array
BFS
Optional
DP
Good
3 month
vector
STL
Backlog
2 week

→ Max / min 2D array



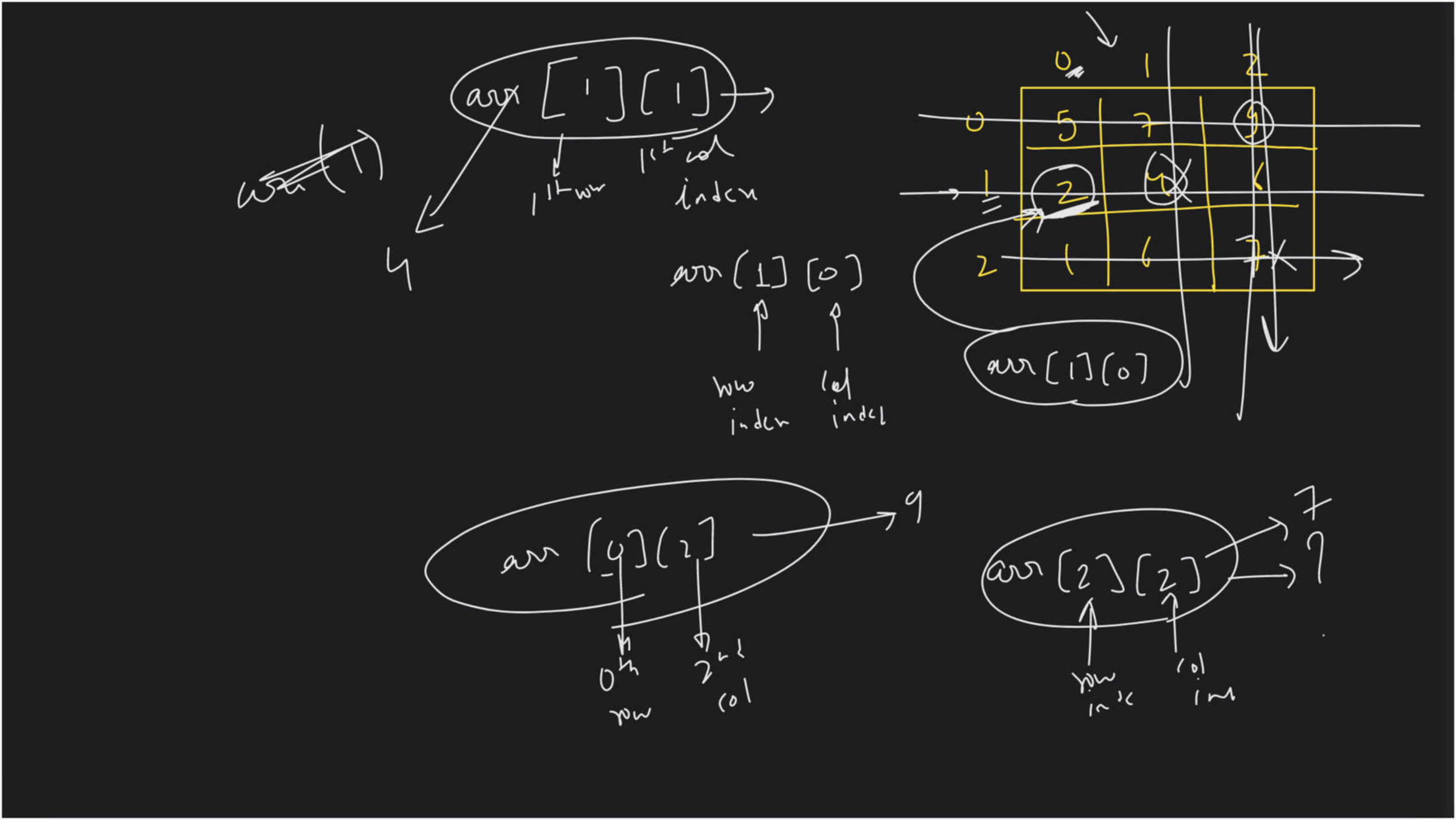
The Boys

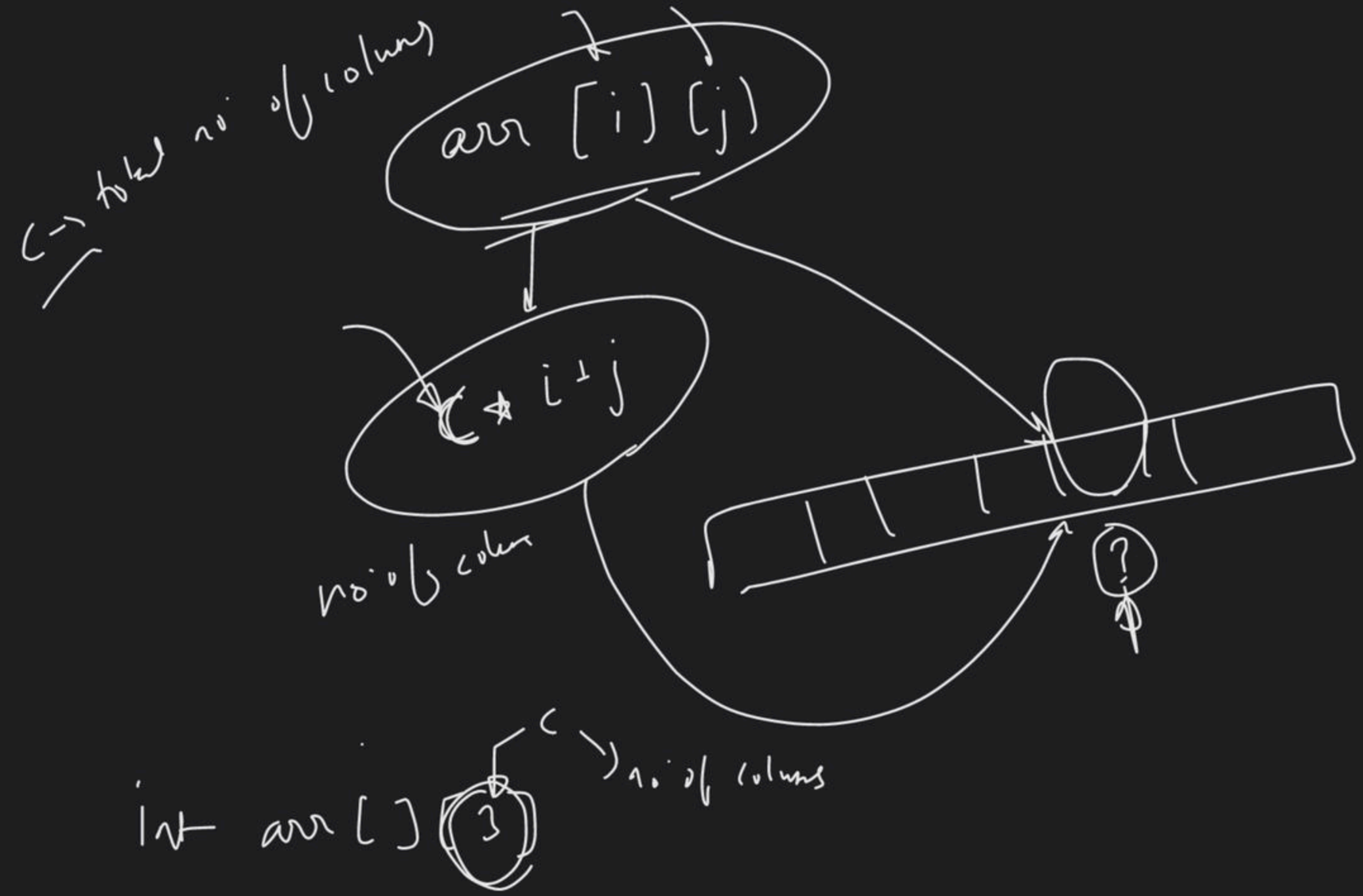
Edit

2:30 hr

→ maxi = INT_MIN

→ mini = INT_MAX


arr [1][1]
1st row index
1st col index
4
arr[1][0]
row index
col index
0
1
2
5
7
9
2
4
6
1
6
7
arr[1][0]
arr [4][2]
9
0th row
2nd col
arr [2][2]
7
7
row index
col index


arr [i] [j]
(*arr + i) + j
no. of columns
no. of columns
int arr [ ] [3]
no. of columns
?

Transpose a matrix

row → col

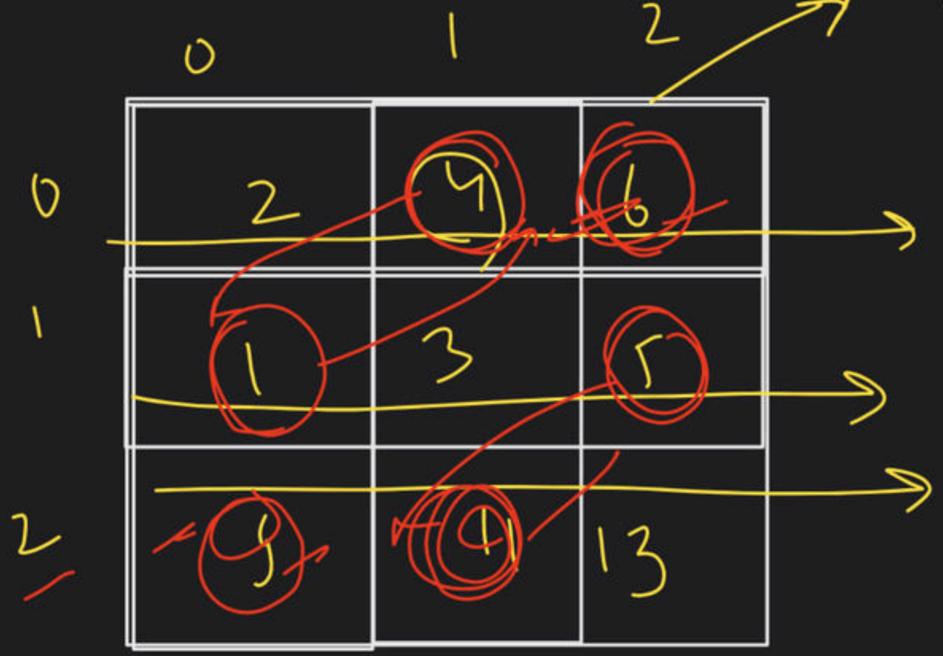
col → row

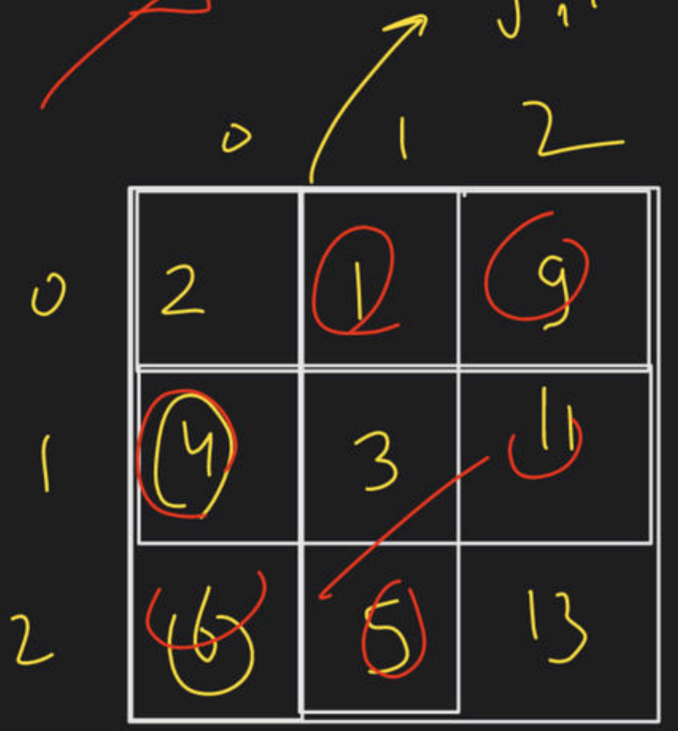
i , j

arr[i][j] ↔ arr[j][i]

swap

j , i





2 → arr[0][0]

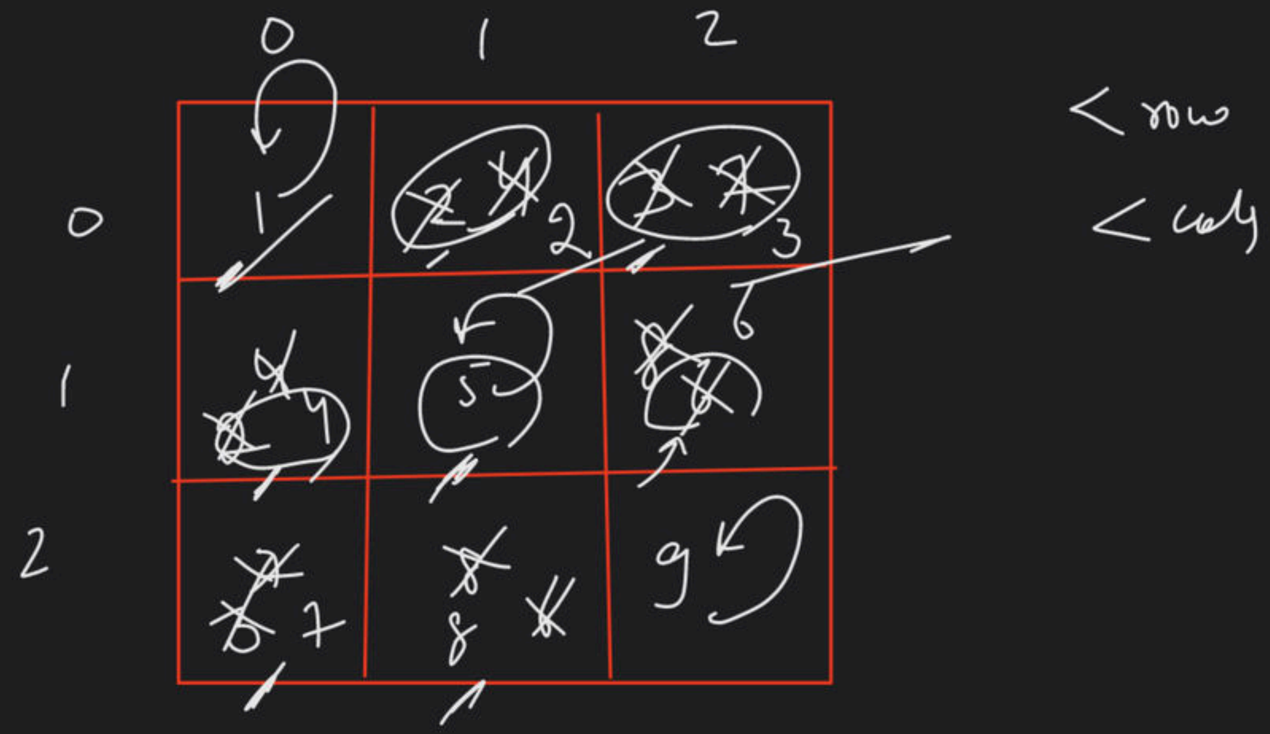
4 → arr[0][1]

6 → arr[0][2]

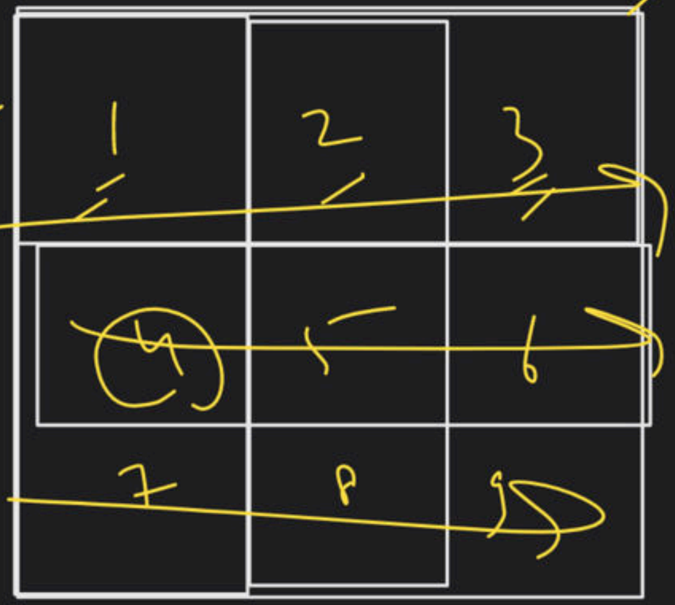
2 → arr[0][0]

4 → arr[1][0]

6 → arr[2][0]


0
1
2
0
1
2
1
2
3
5
6
7
8
9
< row
< col

confusion → DRY RUN



3×3 matrix

2 → (0,1)

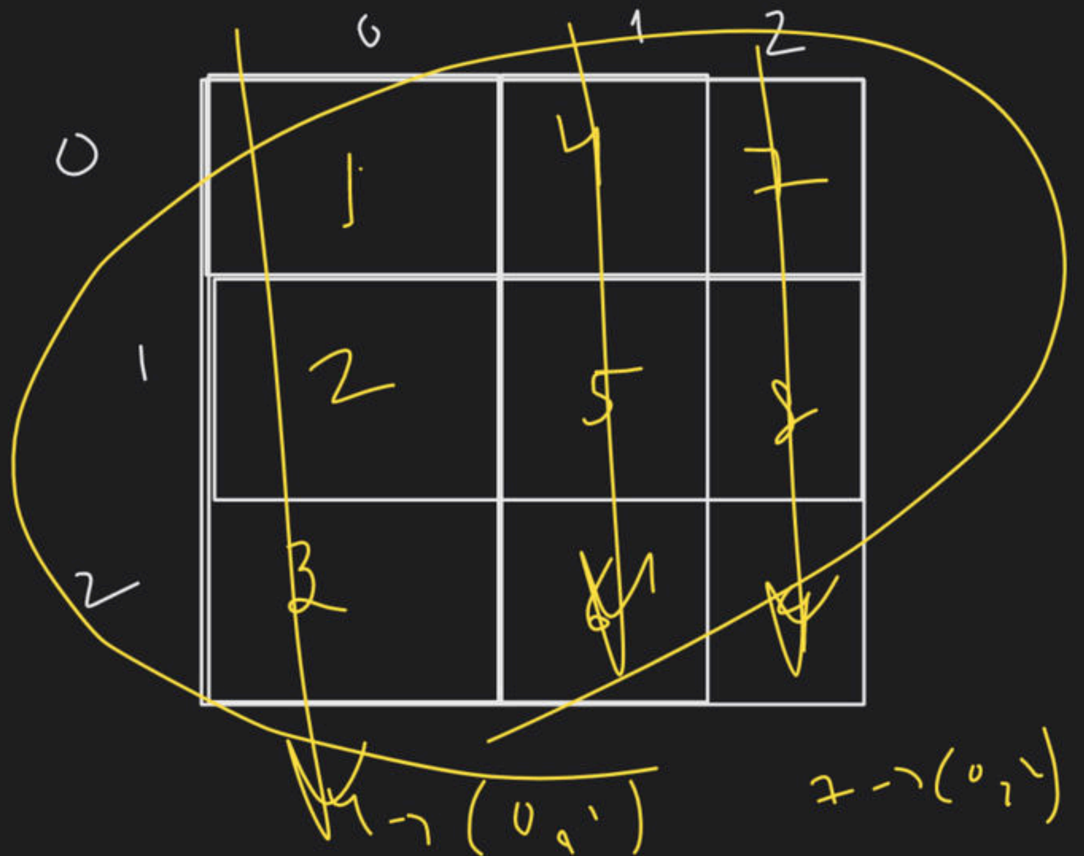
3 → (0,2)

4 → (1,0) 7 → (2,0)

5 → (1,1) 8 → (2,1)

6 → (1,2) 9 → (2,2)

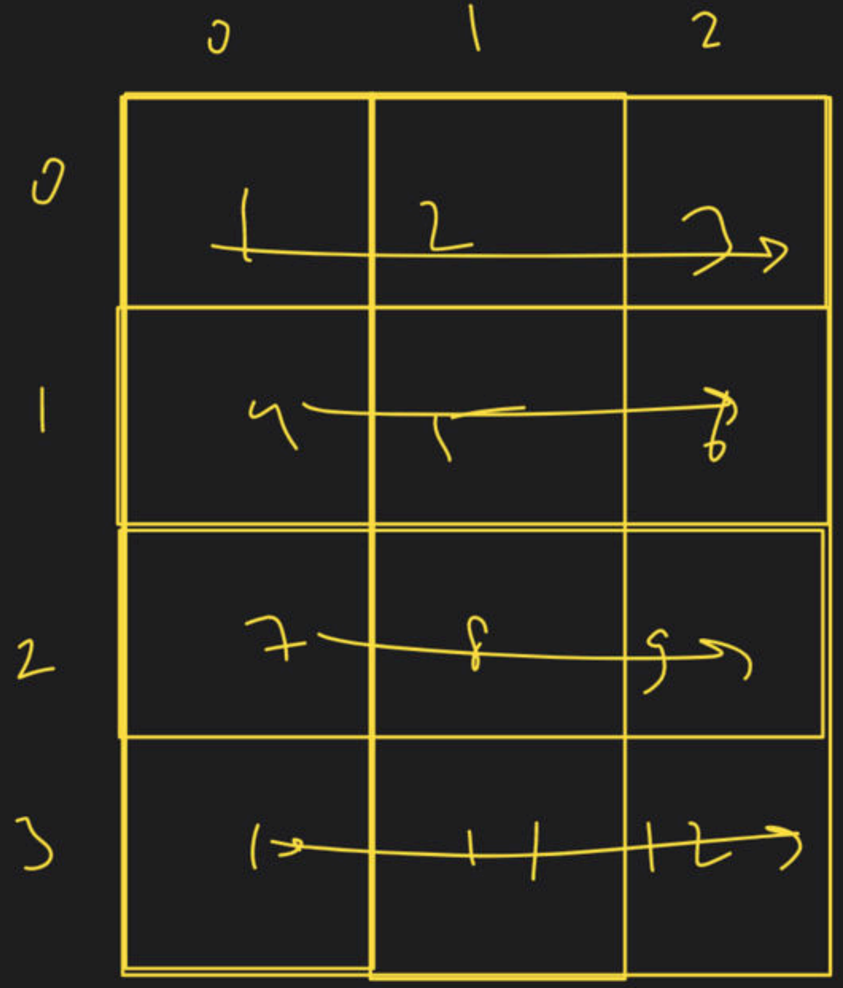
2 → (1,0) 3 → (2,0)



4 → (0,1) 7 → (0,2)

5 → (1,1) 8 → (1,2)

6 → (2,1) 9 → (2,2)



4 x 3 matrix

1	4	7	10
2	5	8	11
3	6	9	12

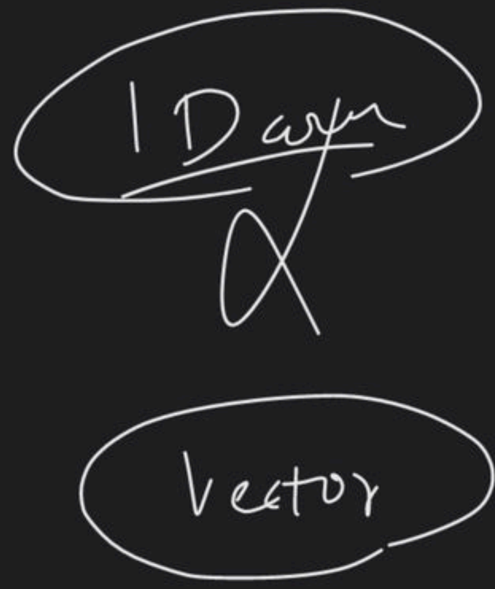
3 x 4 matrix

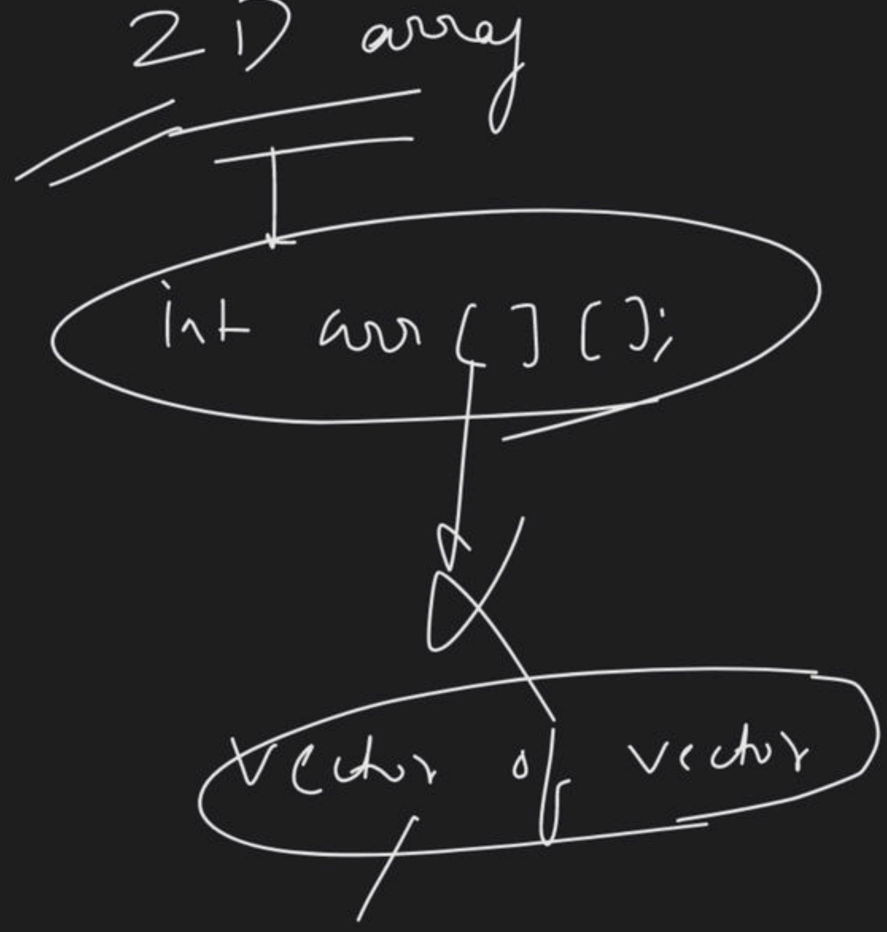
Rotate a matrix by 90°

Spiral print

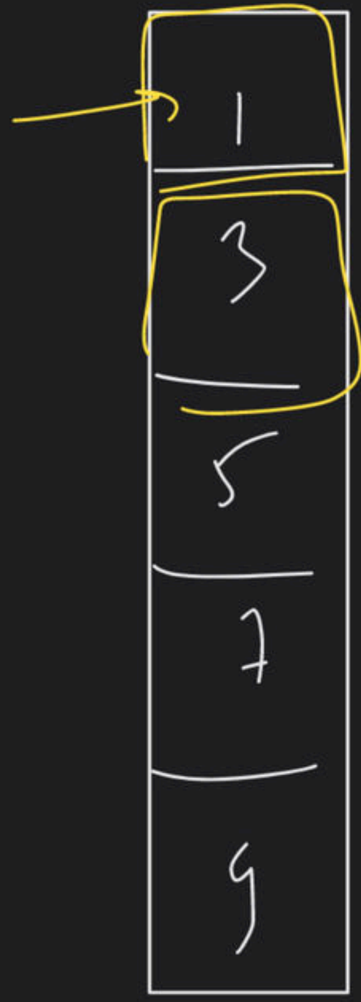
wave print (zig zag print)

median of two sorted array


1D array
Vector

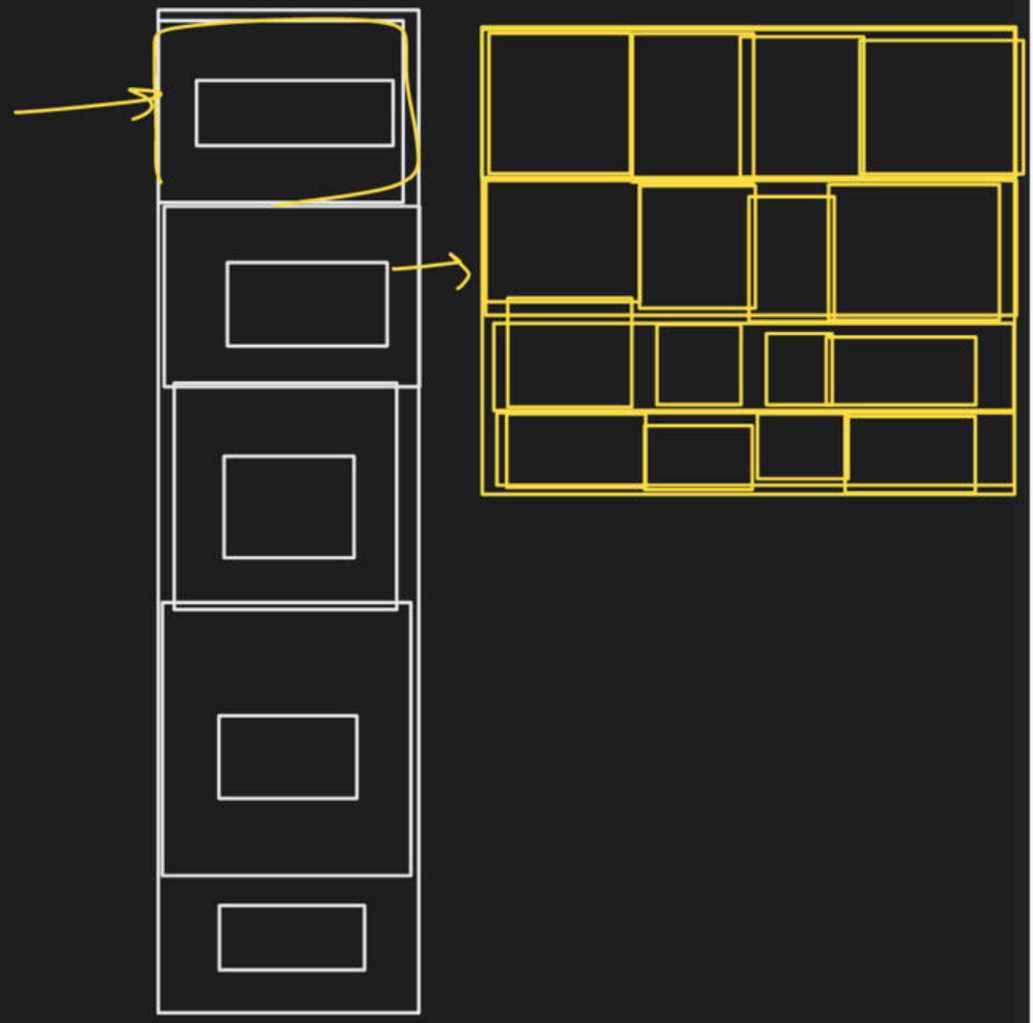

2D array
int arr[][];
Vector of vector

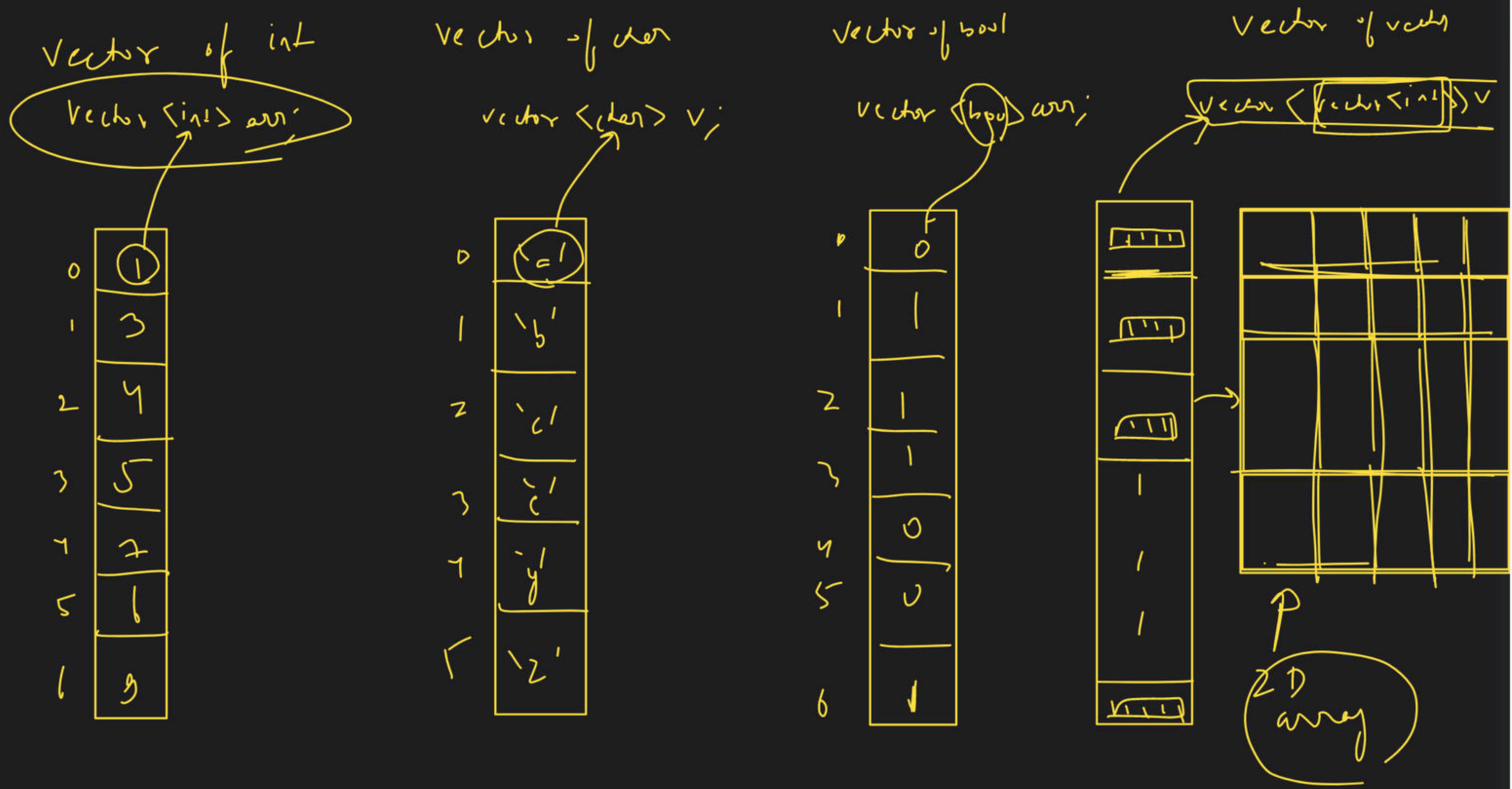
Vector of int

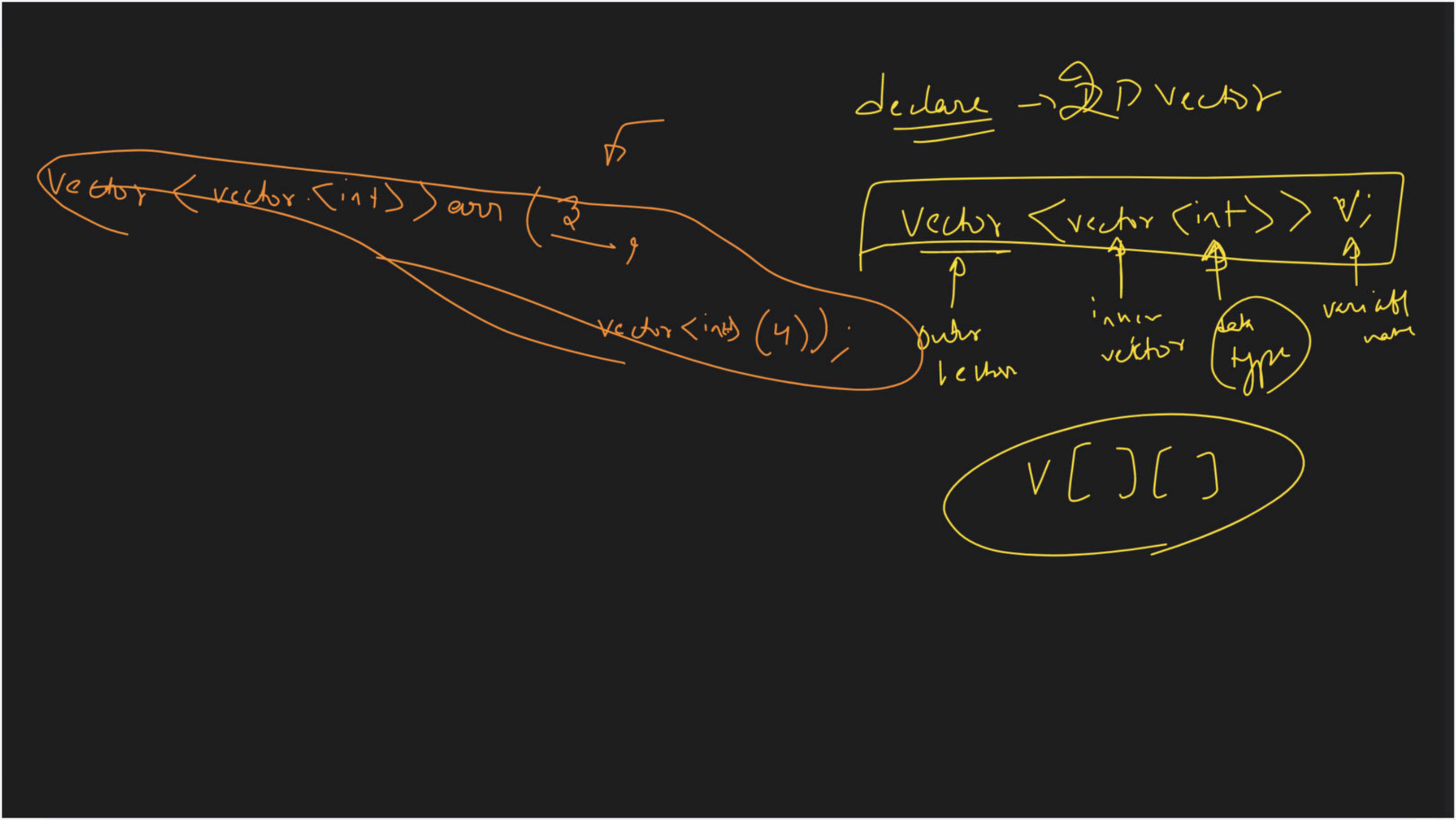


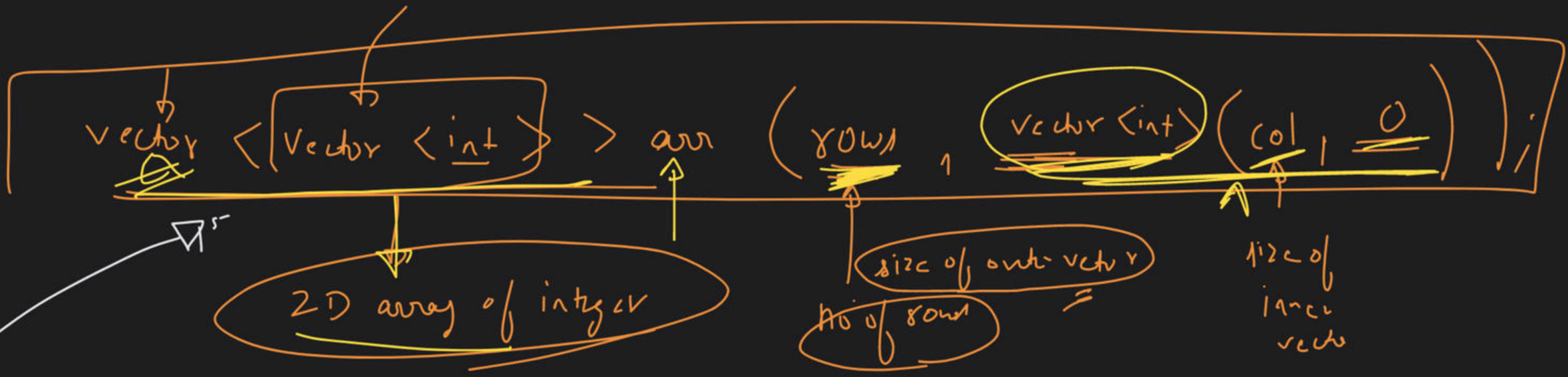
① Rotate by 90°
② Spiral Print
③ Wave / Zig Zag print.

Vector of vector




Vector of int
vector<int> arr;
0
1
2
3
4
5
6
Vector of char
vector<char> V;
'a'
'b'
'c'
'd'
'y'
'z'
Vector of bool
vector<bool> arr;
Vector of vector
vector<vector<int>> V
2D array

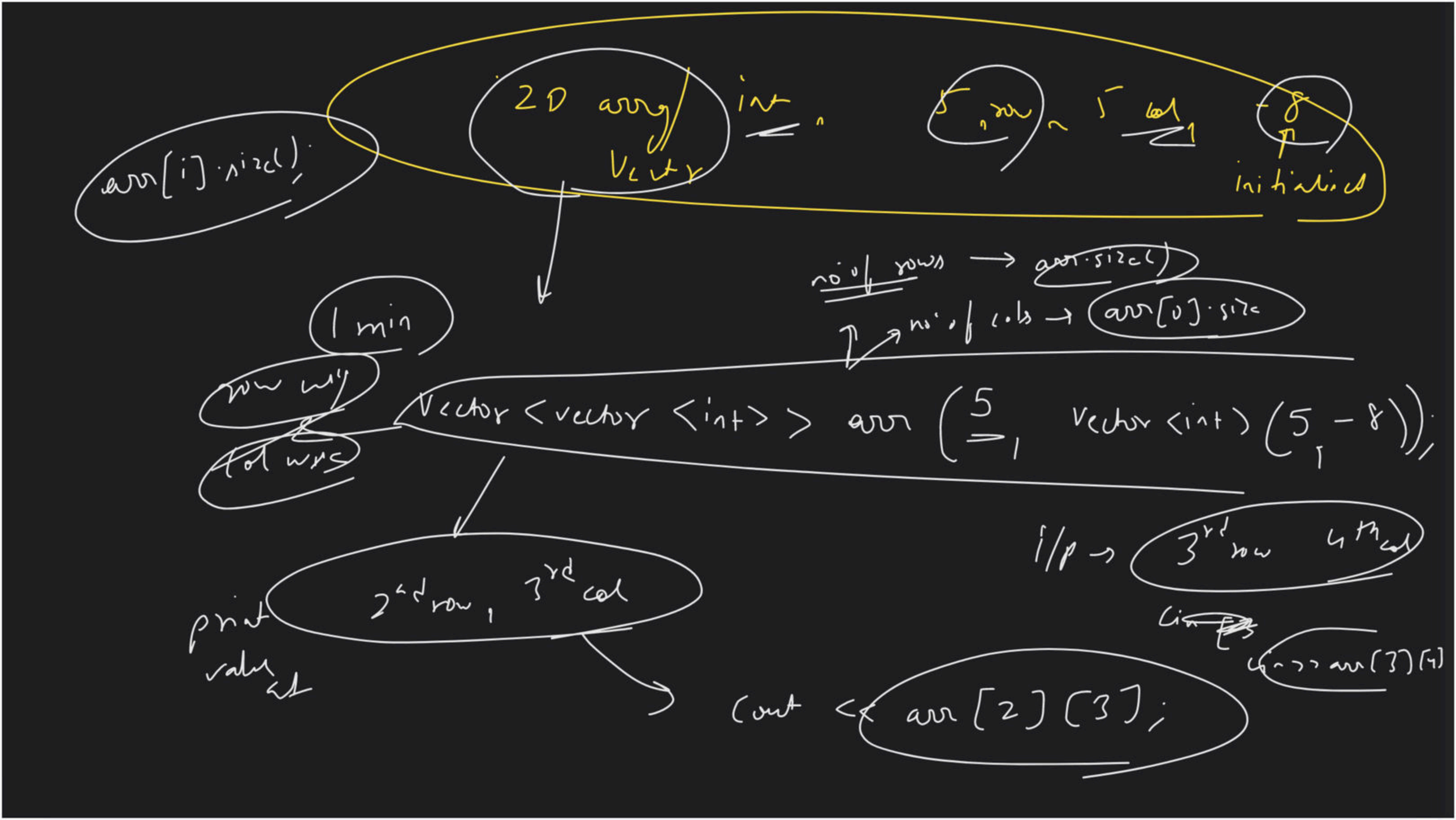

declare → 2D vector
vector<vector<int>> arr (3, vector<int> (4));
vector<vector<int>> v;
outer vector
inner vector
data type
variable name
v[][]

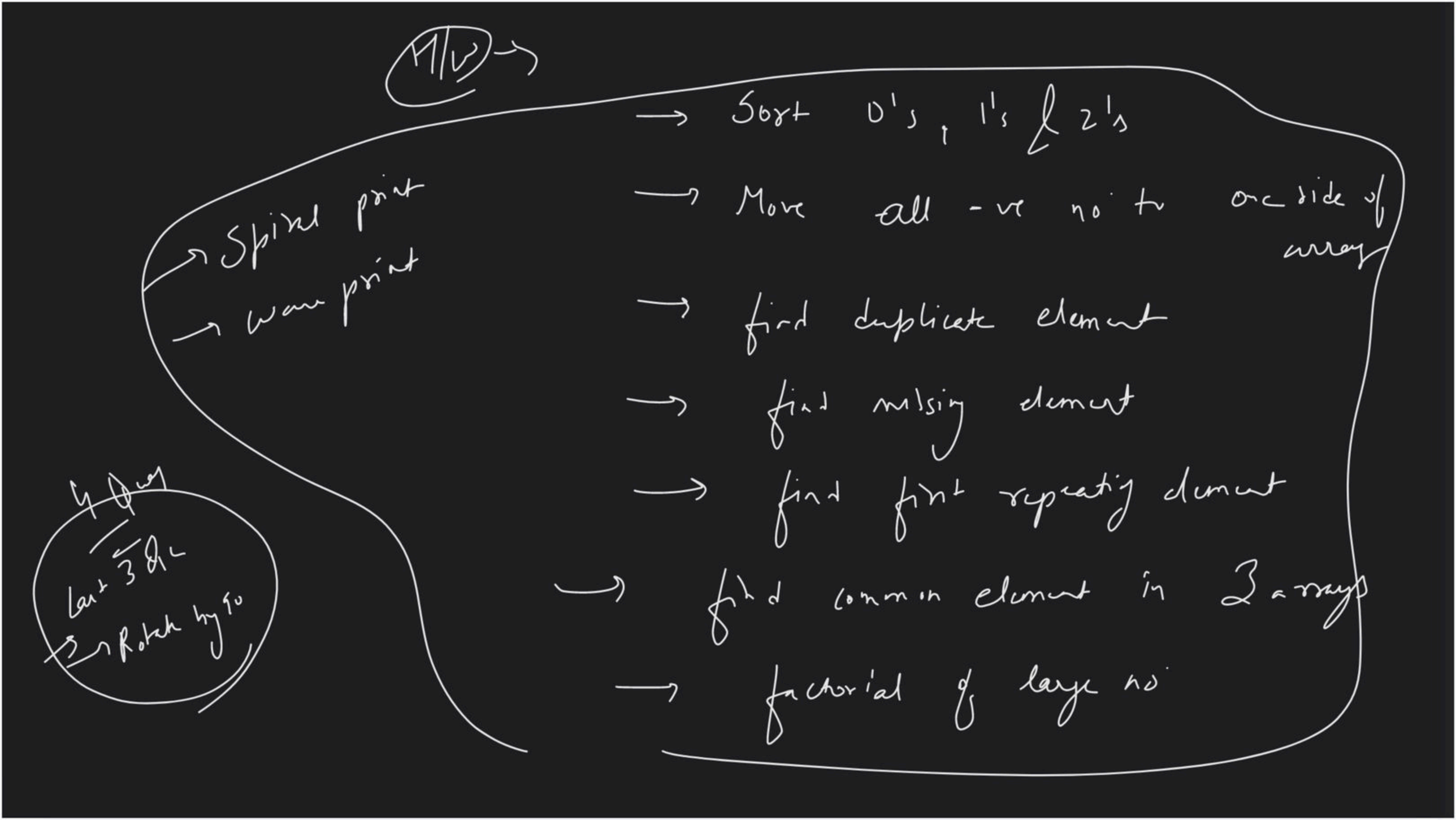


vector<int> a (n, 0)

n: size

0: initial value


2D array / Vector
int
5 rows
5 cols
8
initialized
arr[i].size();
no. of rows → arr.size()
no. of cols → arr[0].size
1 min
row wise
col wise
Vector<vector<int>> arr(5, Vector<int>(5, 8));
print value at
2nd row, 3rd col
i/p → 3rd row 4th col
cin >> arr[3][4]
cout << arr[2][3];


MTW →
→ Sort 0's, 1's & 2's
→ Move all -ve no to one side of array
→ Spiral print
→ Wave print
→ find duplicate element
→ find missing element
→ find first repeating element
→ find common element in 3 arrays
→ factorial of large no
4 Ques
Last 3 Qs
→ Rotate by 90