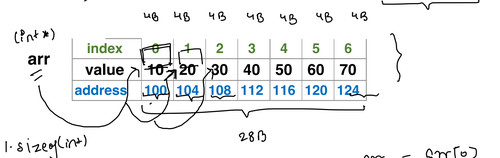
Pointers and 1D-arrays

In C++, we can think of **name** of an array as a **pointer** to the element at the **0th** index.



arr+1 = arr + 1·sizeof(int)
= 100 + 1·4
= 104

arr+i = arr + i·4B

arr
arr+1
arr+2
⋮
arr+i

&arr[0]
&arr[1]
&arr[2]
⋮
&arr[i]

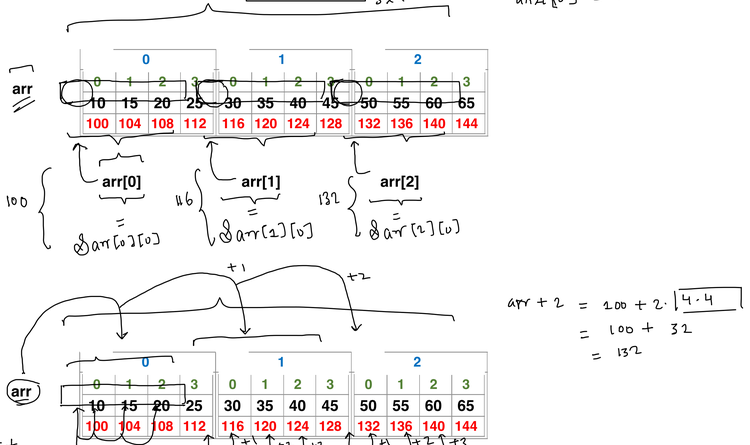
* arr = arr[0]
*(arr+1) = arr[1]
*(arr+2) = arr[2]
⋮
*(arr+i) = arr[i]

arr[i][j]

	0	1	2	3
0	10	25	20	25
1	30	35	40	45
2	50	55	60	65

Pointers and 2D-arrays

arr[i] = &arr[i][0]



arr + 2 = 100 + 2·4·4
= 100 + 32
= 132

arr+1 = arr + 1·sizeof(row)
= 100 + 1·44
= 116

arr[0] (int*)
arr[1]
arr[2]

arr[i] + j = arr[i] + j * sizeof(int)

arr of integers

#col * sizeof(int)

arr + i = arr + i * sizeof(row)

arr of 1D arrays
rows

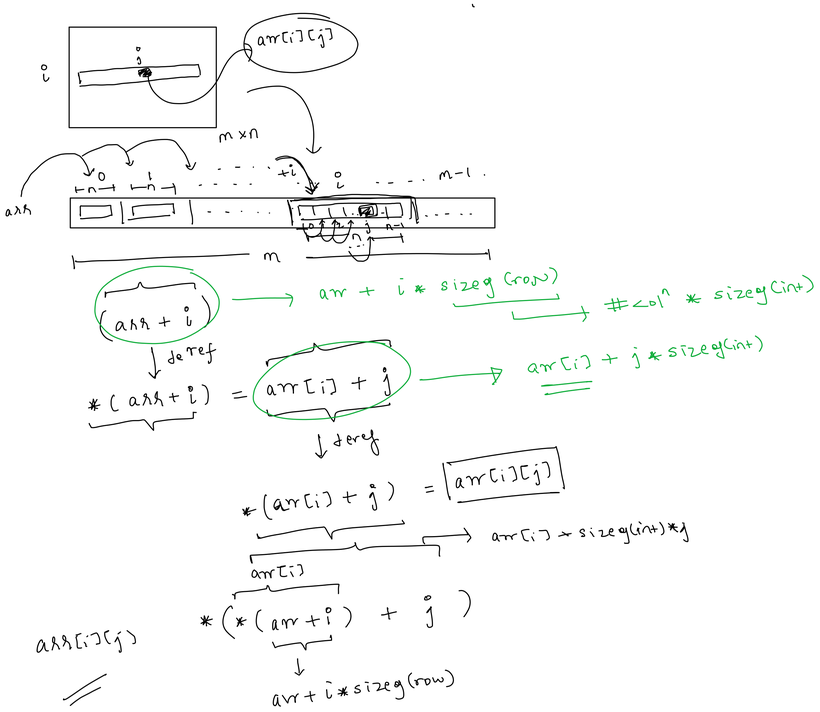
arr[i][j] = interval ? col

arr[]

0	1	2	.....
10	20	20	....

arr[0] = *arr
arr[1] = *(arr+1)
arr[2] = *(arr+2)
⋮
arr[i] = *(arr + j)

arr + i * sizeof(int)


arr[i][j]
i
j
m x n
0
1
i
i+1
m-1
arr
n
m
(arr + i)
arr + i * sizeof(row)
#cols * sizeof(int)
deref
*(arr + i) = arr[i] + j
arr[i] + j * sizeof(int)
deref
*(arr[i] + j) = arr[i][j]
arr[i] + j * sizeof(int)
arr[i]
((arr + i) + j)
arr[i][j]
arr + i * sizeof(row)

Matrix Search

Given an integer matrix of dimensions **m x n**, and a target integer **T**, write a program to search for the target in the given matrix.

Example

Input : T = 10

	0	1	2
0	50	80	20
1	90	10	70
2	60	30	40

Output : True

Sorted Matrix Search I

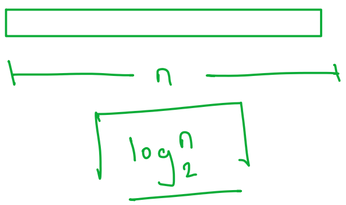
Given an integer matrix of dimensions **m x n**, which is **sorted** *row-wise* and a *target* integer **T**, write a program to **search** for the target in the given matrix.

Example

Input : T = 50

	0	1	2
0	40	50	60
1	10	20	30
2	70	80	90

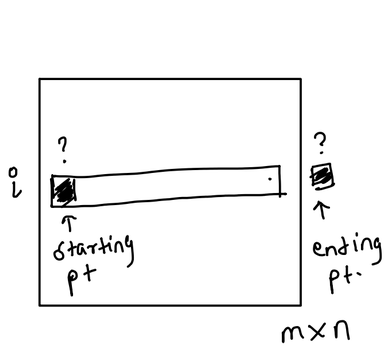
m x n

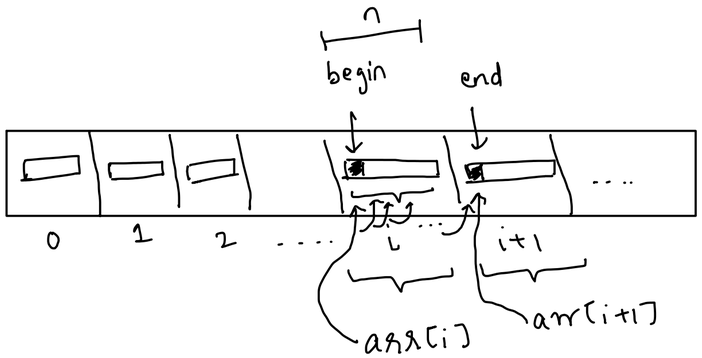


Output : True

O(m . log n)
2

↓ # rows ↓ time spent on each row

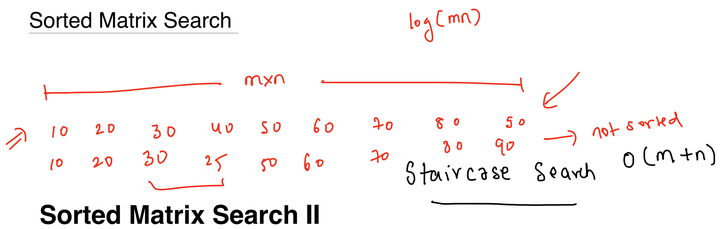




binary-search(arr[i], arr[i+1], t) = &arr[i][0] = &arr[i+1][0]

arr[i] or arr[i] + n ↓ begin ↓ end

Sorted Matrix Search



Sorted Matrix Search II

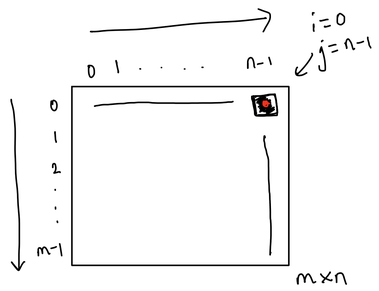
Given an integer matrix of dimensions **m x n**, which is **sorted** *row-wise* and *column-wise* and a *target* integer **T**, write a program to **search** for the target in the given matrix.

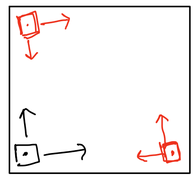
Example

Input : T = 50

	0	1	2
0	10	20	30
1	40	50	60
2	70	80	90

Output : True





i = m-1
j = 0

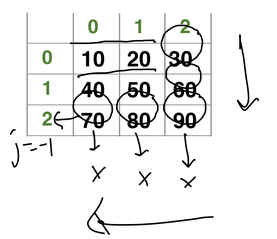
t > mat[i][j]
j++
t < mat[i][j]
i--
while (i >= 0 and j <= n-1)

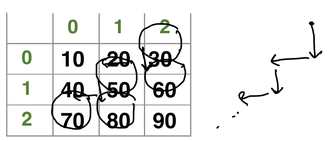
i=0
j=n-1
while(i<=m-1 and j>=0)
if(mat[i][j]==t) {
return true;
} else {
if (t > mat[i][j]) {
i++;
} else { // t < mat[i][j]
j--;
}
}

t = 65

t = 55

	0	1	2
0	10	20	30
1	40	50	60
2	70	80	90





Diagonal Traversal

Given an integer matrix of dimensions **m x n**, design an algorithm to **traverse** the matrix **diagonally**.

Example

Input

	0	1	2	3
0	11	12	13	14
1	15	16	17	18
2	19	20	21	22

Output

19	15	20	11	16	21	12	17	22	13	18	14

	0	1	2	3
0	11	12	13	14
1	22	23	24	15
2	21	26	25	16
4	20	19	18	17

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

(0,0) (0,1) (0,2) (0,3)

(0,j) 0 ≤ j ≤ n-1

1 ≤ i ≤ m-1

0,0 → 3 ✓ 1,0 → 2 ✓
0,1 → 3 ✓ 2,0 → 1 ✓
0,2 → 2 ✓
0,3 → 1

i=1 j=0
min(3-1, 4-0)
i=2 j=0
min(3-2, 4-0)

i,j → min(m-i, n-j)

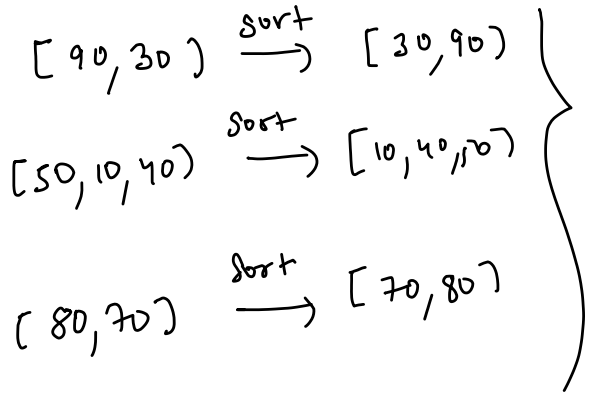
i=0 j=0
(3-0, 4-0) min
i=0 j=1
(3-0, 4-1) min
i=0 j=2
(3-0, 4-2) min
i=0 j=3
(3-0, 4-3) min

Diagonal Sort a Matrix

Given an integer matrix of dimensions **m x n**, write a program to **sort** it **diagonally**.

Input

	0	1	2
0	50	80	20
1	90	10	70
2	60	30	40



Output

	0	1	2
0	10	70	20
1	30	40	80
2	60	90	50

Spiral Print a Matrix

Given an integer matrix of dimensions **m x n**, design an algorithm to print the matrix spirally in a **clock-wise** direction.

Example

Input

	0	1	2	3
0	11	12	13	14
1	22	23	24	15
2	21	26	25	16
4	20	19	18	17

Output

11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

sr sc ec ec

	0	1	2	3
0	11	12	13	14
1	22	23	24	15
2	21	26	25	16
3	20	19	18	17

4x4

sr = 0
sc = 0
er = m-1 = 3
ec = n-1 = 3

sr <= er
and
sc <= ec

// 1. print sr
it. from sc to ec

sr++

// 2. print ec
it from sr to er

ec--

// 3. print er
it from ec to sc

er--

// 4. print sc
it fr~ er to sr

sc++