10/02/2026
Tuesday

Kth element 2 sorted arrays

arr1 = [1 3 4 7 10 12] k=4

arr2 = [2 3 6 15]

for kth element in 2 sorted arrays

1 2 3 4 3 4 6 7 10 12 15

is 3

We do same like previous approach where we are calculated median by taking $\left(\frac{n_1+n_2+1}{2}\right)$ elements to left side but for this question we don't need to find median Here we need to find kth element

Here we are taking only 4 elements on left side and rest are right side

$n_1 + n_2 - K$

$10 - 4 = 6$

4

1 l_1 | r_1 3 4 7 10 12

2 3 6 l_2 | r_2 15

6

The array is sorted only when it satisfies this conditions

$l_1 < r_2$ and $l_2 < r_1$

$l_1 = 1$ $l_2 = 6$ $r_1 = 3$ $r_2 = 15$

$1 < 15$ $6 < 3$

false

Now we take two elements from arr1

4

1 3 l_1 | r_1 4 7 10 12

2 3 l_2 | r_2 6 15

6

$3 < 6$ and $3 < 4$

true

So here array is in sorted order so the answer is $\max(l_1, l_2)$

Here we have edges cases

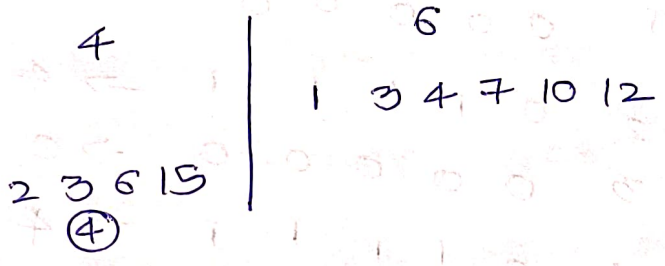
arr1 = [1 3 4 7 10 12] $K = 4$

$n_1 = 6$

so if we take elements at left side then it exceeds K elements

to overcome this we take min(n1,k);
so it will take k elements only

one more edge case
you are not taking any element from arr1

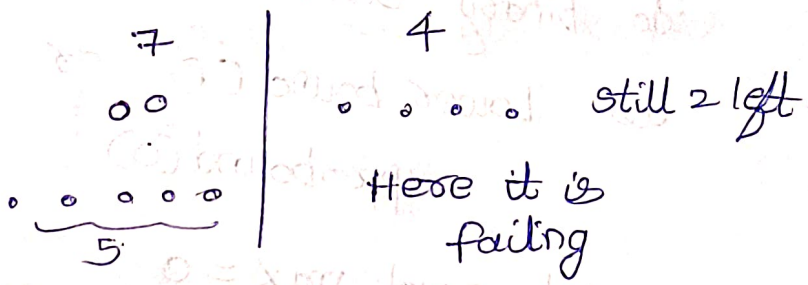


take this example

arr1 → { } 6

k=7

arr2 → { } 5



so low is max(0, k-n2)
↓ smallest size array

Code is same as previous
Median of two sorted arrays

Changes are

low = max(0, k-n2)

high = min(k, n2)

left = k;

BS on 2D arrays

Find the row with maximum 1's

0	0	0	1	1	1	→ 3
1	0	0	0	0	0	→ 0
2	0	1	1	1	1	→ 4
3	0	0	0	0	0	→ 0
4	0	1	1	1	1	→ 4

if two are same
return smallest row no

Approach

each row is sorted order so I will do binary search and find no of one's as Lower bound (1) or upperbound (0)

```
int count_max = 0
int index = -1
for(int i=0; i<n; i++)
{
    int count_ones = lowerbound
                     m- (mat[i], m, 1);
    if (count_ones > count_max)
    {
        count_max = count_ones;
        index = i;
    }
}
return index;
```

Search in a 2D Matrix

mat[][] = $\begin{bmatrix} 30 & 4 & 1 & 72 & 93 \\ 12 & 13 & 16 & 18 & \\ 20 & 21 & 23 & 29 & \\ 8 & 9 & 10 & 11 & \end{bmatrix}$ target = 23

0 1 2 3

0 1 2 3 4 5 6 7

4

4/4

1

4%4

for row = mid/column mid/m

for col = mid % column mid%m

Here every row and every column is sorted

one approach

```
for (i=0 -> n-1){
    if (mat[i][0] <= target &&
            target <= mat[i][m-1]) {
        return bs(mat[i], target);
    }
}
```

for this T.C is

$O(n) + \log_2 m$

only doing one time

```
int low=0 , high= n*m-1;
while (low<= high){
    int mid= low+ (high-low)/2;
    int i= mid/m;
    int j= mid%m;
    if (mat[i][j] == target)
            return true
    else if (mat[i][j] <target)
            low=mid+1;
    else
        high=mid-1;
}
return false;
```

Search target in 2D array and return co-ordinates

1	4	7	11	15
2	5	8	12	19
3	6	9	16	22
10	13	14	17	24
18	21	23	26	30

tar=14

first we take last element in the first row

15 > tar

So we move left side

So we decrease

col--;

Now it is 11

11 < 14 (tar)

So we move downwards

Now it is 12

12 < 14 (tar)

So we move downwards

Now it is 16

16 > 14 So

we move left side

Now it is 9

9 < 14 (tar)

So we move downwards

Now it is 14

14 == 14

So we will return row & col

```
public int solve (int[][] mat, int n, int m)
                                    int target
{
    int row=0;
    int col=m-1;
    while (row<n && col>=0){
        if (mat[row][col]==target)
            return new int[] {row,col}
```

```
else if (mat[row][col] < tar)
{
    row++;
}
else
{
    col--;
}
}
return new int[] {-1,-1};
}
```

Find the peak element in 2-D matrix

2 adjacent cells are not same

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

Here peak element should be greater than its upper element and Lower/bottom and left and right element.

Brute-force

we check for every element whether it satisfies 4 cond^n^s

T.C:-

In worst-case it gives you 4 O(n×m)

Approach-2

If we find Largest element in this 2D array then that is the answer

Time Complexity : O(n×m)

Still need to optimise

Approach-3

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

Here we will do binary search on Columns

low=0 mid=2 high=5

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

we got mid=2

in that column we find max ele

~~For~~ that max ele we need that max element row

Here if we find max element in column then no need to check bottom element and top element

we need to check only left element and right element

Here for the column 2
maxi element is 6 and
its row is 2

Now we know row and col
we will check if its left ele is
greater then we remove right
half
if it's right ele is greater then
we remove left half

row=2 mid=2

if(mat[2][2]/mat[row][mid] < mat[row][mid-
then we remove right half
high=mid-1

else if (mat[row][mid] < mat[row][mid+1])
low=mid+1;
we remove left half

Here 6<7 so
high=mid-1

low=0
mid=0

4
2
1
3

high=1

2
9
7
6

Now again we find max ele row and check it's left and right if it satisfies this then we return that element

```
public int solve (int[][] mat, int n, int m)
{
    int low=0;
    int high=m-1;
    while (low<=high){
        int mid = (low+ high)/2
        int row= maxele (mat,n,m,mid);
        if(mat[row]
        int left = (mid-1 >=0)? mid-1 : -1
        int right = (mid+1 <m)? mid+1 : -1
```

mat[row][mid-1]

mat[row][mid+1]

```
    if (mat[row][mid] > left &&
            mat[row][mid] > right)
        return new int[] {row,mid};
    else if (mat[row][mid] < mat[row][mid
                                left)
        high=mid-1;
    else
        low=mid+1;
    }
    return new int[] {-1,-1};
}
```

Similarly we can do using rows find max ele in a row and find its ~~of~~ similarly

Median of row-wise sorted matrix

$$\begin{bmatrix} 1 & 5 & 7 & 9 & 11 \\ 2 & 3 & 4 & 5 & 10 \\ 9 & 10 & 12 & 14 & 16 \end{bmatrix}$$

n × m
↓ ↓
odd odd

brute force

I will add all elements into my list and sort the list and then return middle element median

T.C $O(n \times m) + n \times m\, O(\log(n \times m))$

optimal approach

1 ~~2~~ 3 4 5 5 ~~7~~ 9 9 ~~10~~ 10 ~~11~~ 12 ~~14~~ ~~16~~
1 2 3 4 6 6 7

x{1 2 3 4 5 5 7}x

no of elements <=5

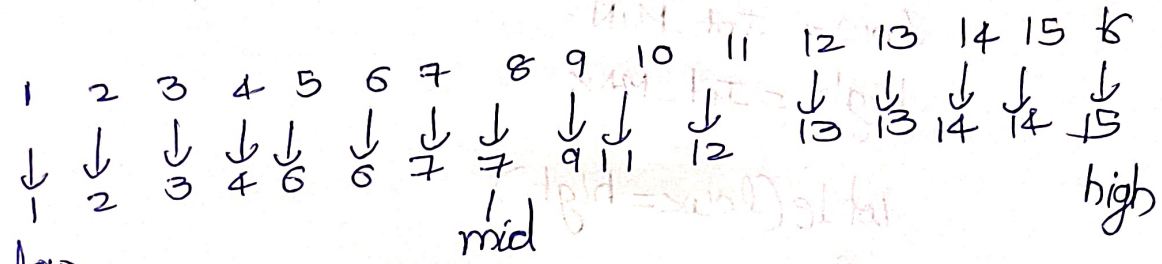
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
1 2 3 4 6 6 7 7 9 11 12 13 13 14 14 15

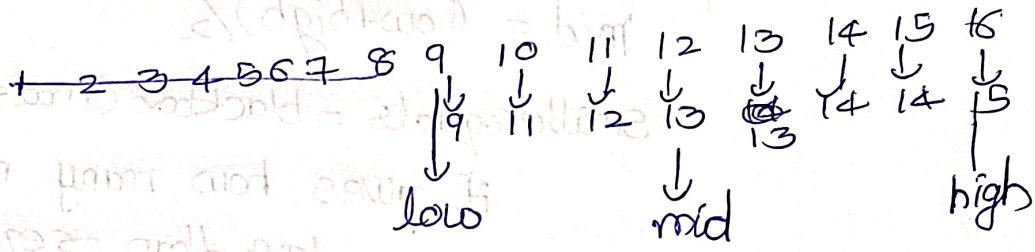
↳ no of elements <=9

1 2 3 4 5 7 7 (9) 9 10 10 11 12 14 16
7 m 7

9 is median

(no of elements <= median) > $\left(\frac{n \times m}{2}\right)$

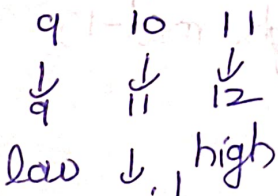




mid=12 13>7

so we move left side

high = mid-1

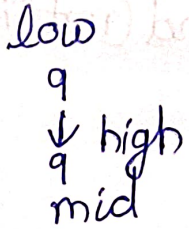


mid=10 11>7

but we need first one

so we remove right half

high = mid-1;



9>7

high = mid-1

high=8 low=9

low crosses high

answer is Low

```
public
function (mat, n, m)
{
    low = Int_MIN
    high = Int_MAX

    while (low <= high)
    {
        mid = (low+high)/2
        smallerequals = blackbox (mat, n, m, mid)
                it gives how many no's are
                        less than or equal to mid

        if (smallerequals <= req)
                low = mid+1;
        else
                high = mid-1;
    }
}
function blackbox (mat, n, m, mid)
{
    count = 0;
    for (i = 0 -> n-1)
        count += upperbound (mat[i], mid);

    return count;
}
```