**Find a specific pair in Matrix**

Given an n x n matrix mat[n][n] of integers, find the maximum value of mat(c, d) – mat(a, b) over all choices of indexes such that both c > a and d > b.  
**Example:**

Input:

mat[N][N] = {{ 1, 2, -1, -4, -20 },

{ **-8**, -3, 4, 2, 1 },

{ 3, 8, 6, 1, 3 },

{ -4, -1, 1, 7, -6 },

{ 0, -4, **10**, -5, 1 }};

Output: 18

The maximum value is 18 as mat[4][2]

- mat[1][0] = 18 has maximum difference.

The program should do only ONE traversal of the matrix. i.e. expected time complexity is O(n2)  
A **simple solution** would be to apply Brute-Force. For all values mat(a, b) in the matrix, we find mat(c, d) that has maximum value such that c > a and d > b and keeps on updating maximum value found so far. We finally return the maximum value.  
Below is its implementation.

// A Naive method to find maximum value of mat[d][e]

// - ma[a][b] such that d > a and e > b

#include <bits/stdc++.h>

using namespace std;

#define N 5

// The function returns maximum value A(d,e) - A(a,b)

// over all choices of indexes such that both d > a

// and e > b.

int findMaxValue(int mat[][N])

{

// stores maximum value

int maxValue = INT\_MIN;

// Consider all possible pairs mat[a][b] and

// mat[d][e]

for (int a = 0; a < N - 1; a++)

for (int b = 0; b < N - 1; b++)

for (int d = a + 1; d < N; d++)

for (int e = b + 1; e < N; e++)

if (maxValue < (mat[d][e] - mat[a][b]))

maxValue = mat[d][e] - mat[a][b];

return maxValue;

}

// Driver program to test above function

int main()

{

int mat[N][N] = {

{ 1, 2, -1, -4, -20 },

{ -8, -3, 4, 2, 1 },

{ 3, 8, 6, 1, 3 },

{ -4, -1, 1, 7, -6 },

{ 0, -4, 10, -5, 1 }

};

cout << "Maximum Value is "

<< findMaxValue(mat);

return 0;

}

The above program runs in O(n^4) time which is nowhere close to expected time complexity of O(n^2)  
An **efficient solution** uses extra space. We pre-process the matrix such that index(i, j) stores max of elements in matrix from (i, j) to (N-1, N-1) and in the process keeps on updating maximum value found so far. We finally return the maximum value.

// An efficient method to find maximum value of mat[d]

// - ma[a][b] such that c > a and d > b

#include <bits/stdc++.h>

**using** **namespace** std;

#define N 5

// The function returns maximum value A(c,d) - A(a,b)

// over all choices of indexes such that both c > a

// and d > b.

**int** findMaxValue(**int** mat[][N])

{

    //stores maximum value

**int** maxValue = INT\_MIN;

    // maxArr[i][j] stores max of elements in matrix

    // from (i, j) to (N-1, N-1)

**int** maxArr[N][N];

    // last element of maxArr will be same's as of

    // the input matrix

    maxArr[N-1][N-1] = mat[N-1][N-1];

    // preprocess last row

**int** maxv = mat[N-1][N-1];  // Initialize max

**for** (**int** j = N - 2; j >= 0; j--)

    {

**if** (mat[N-1][j] > maxv)

            maxv = mat[N - 1][j];

        maxArr[N-1][j] = maxv;

    }

    // preprocess last column

    maxv = mat[N - 1][N - 1];  // Initialize max

**for** (**int** i = N - 2; i >= 0; i--)

    {

**if** (mat[i][N - 1] > maxv)

            maxv = mat[i][N - 1];

        maxArr[i][N - 1] = maxv;

    }

    // preprocess rest of the matrix from bottom

**for** (**int** i = N-2; i >= 0; i--)

    {

**for** (**int** j = N-2; j >= 0; j--)

        {

            // Update maxValue

**if** (maxArr[i+1][j+1] - mat[i][j] >

                                            maxValue)

                maxValue = maxArr[i + 1][j + 1] - mat[i][j];

            // set maxArr (i, j)

            maxArr[i][j] = max(mat[i][j],

                               max(maxArr[i][j + 1],

                                   maxArr[i + 1][j]) );

        }

    }

**return** maxValue;

}

// Driver program to test above function

**int** main()

{

**int** mat[N][N] = {

                      { 1, 2, -1, -4, -20 },

                      { -8, -3, 4, 2, 1 },

                      { 3, 8, 6, 1, 3 },

                      { -4, -1, 1, 7, -6 },

                      { 0, -4, 10, -5, 1 }

                    };

    cout << "Maximum Value is "

         << findMaxValue(mat);

**return** 0;

}

**Kth element in Matrix**

Given a N x N matrix, where every row and column is sorted in non-decreasing order. Find the kth smallest element in the matrix.

**Example 1:**

**Input:**

N = 4

mat[][] = {{16, 28, 60, 64},

{22, 41, 63, 91},

{27, 50, 87, 93},

{36, 78, 87, 94 }}

K = 3

**Output:** 27

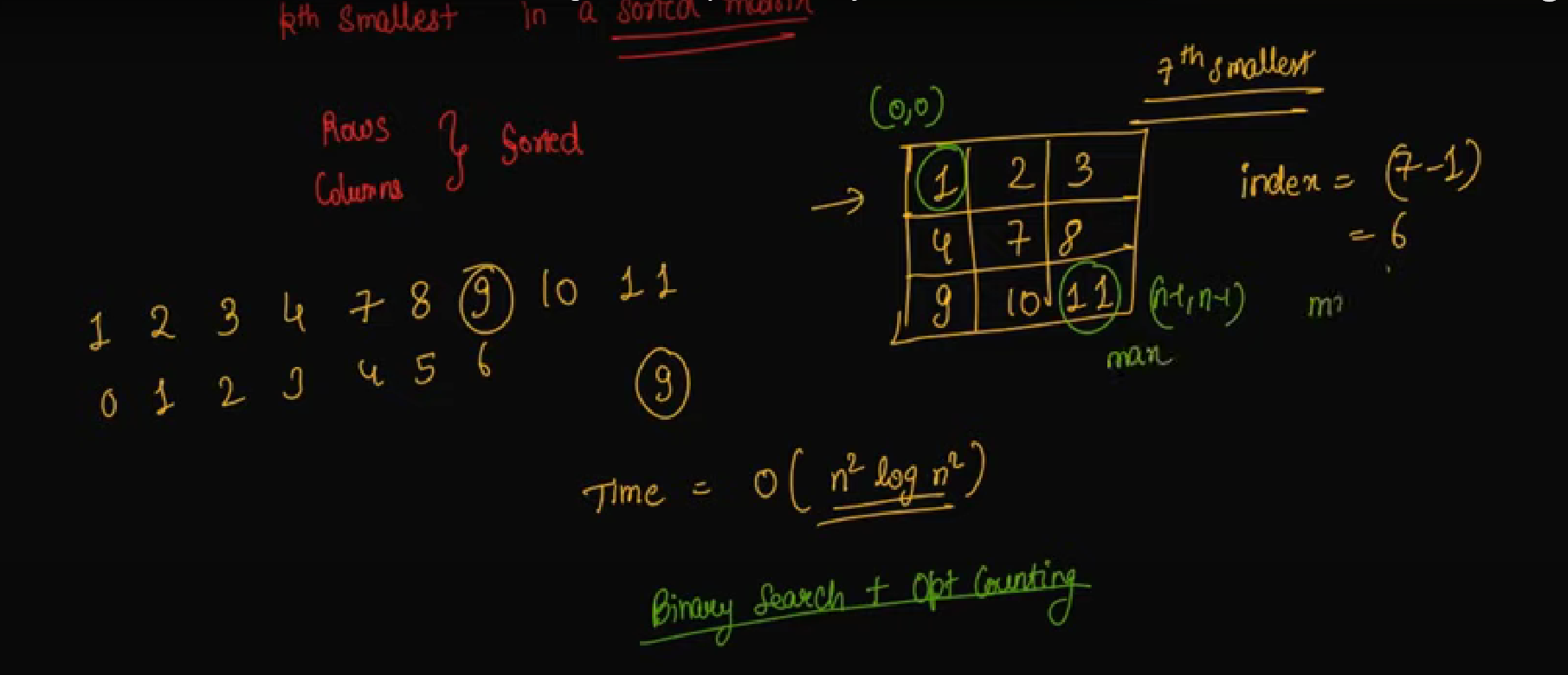
**Explanation:** 27 is the 3rd smallest element.

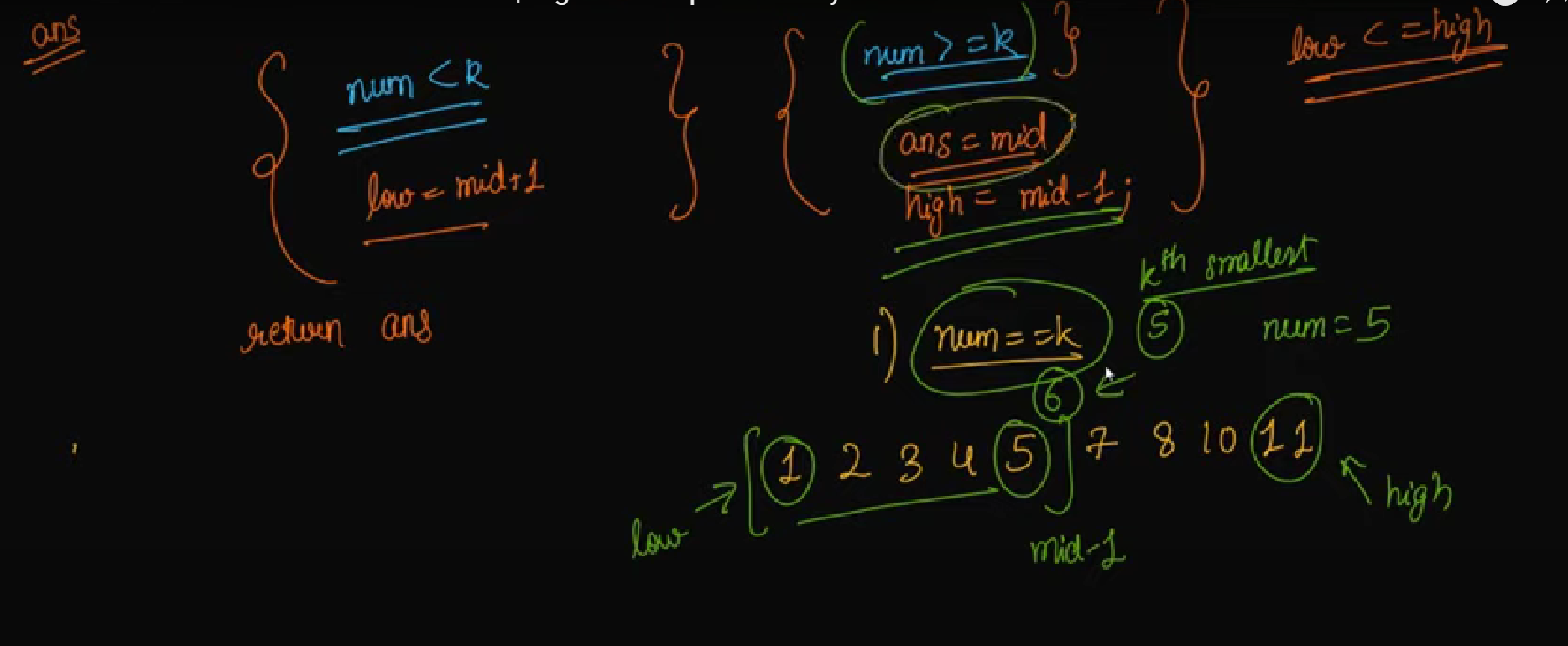
Approaches:

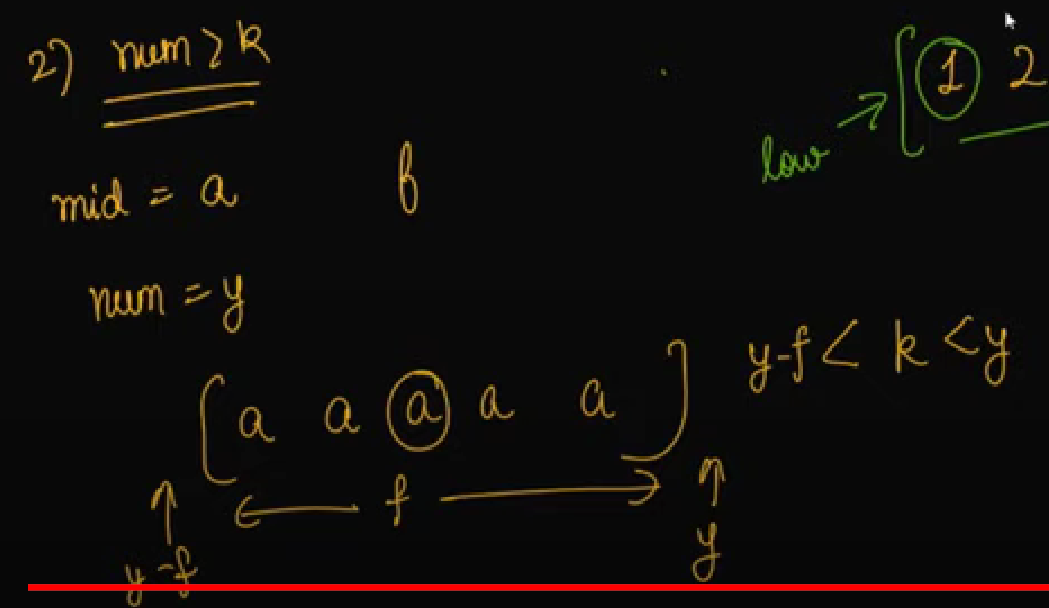
1)Array + sort

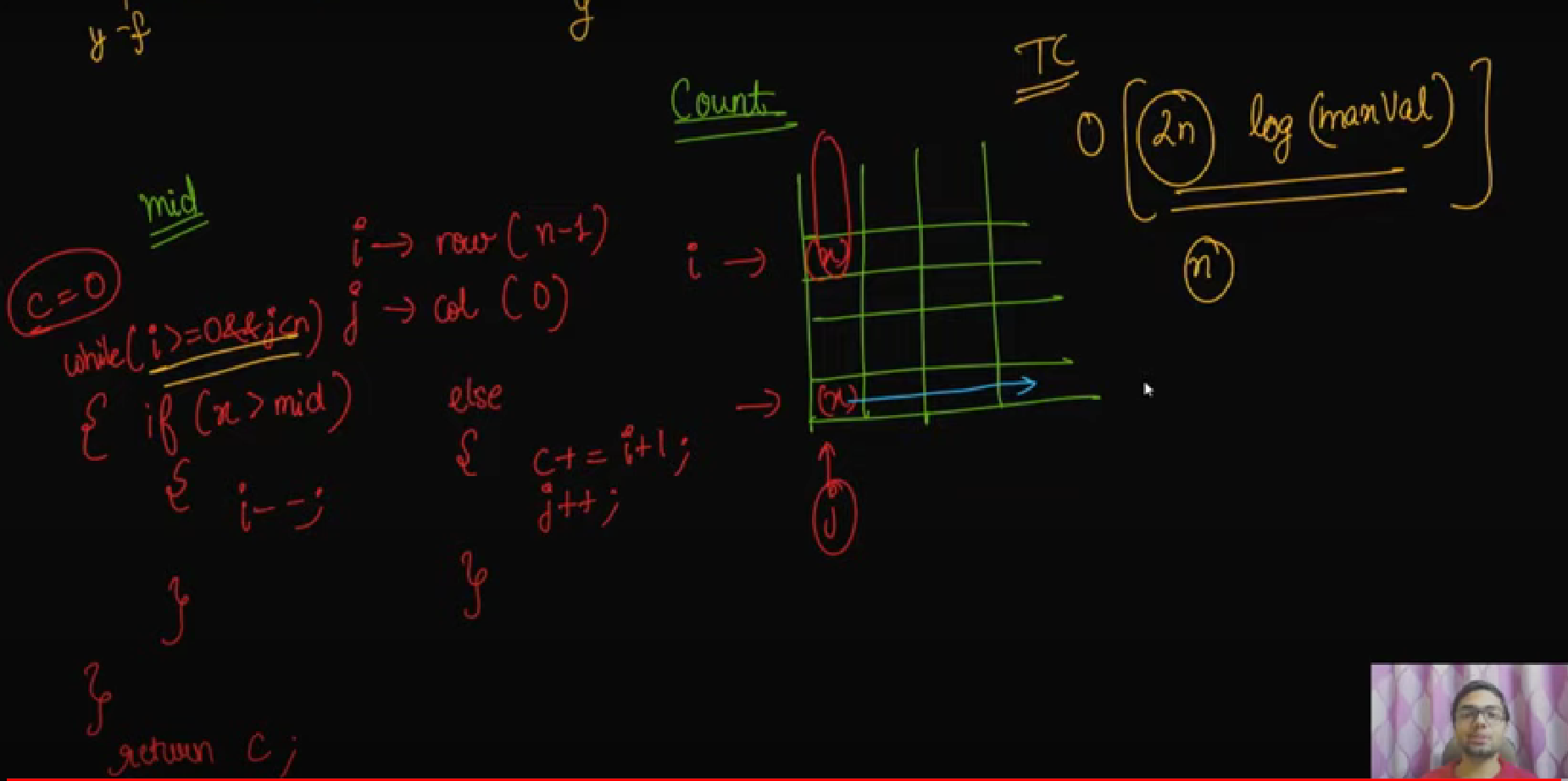
2) Priority Queue

3) Binary Search









int kthSmallest(int mat[MAX][MAX], int n, int k)

{

int l = 0, r =10000 ;

while(l<r){

int mid = (l+r)/2 ;

int count = 0 ;

for(int i = 0 ; i<n ; i++){

count = count + (upper\_bound(mat[i],mat[i]+n, mid) - mat[i]);

}

if(count < k){

l = mid+1 ;

}else{

r = mid ;

}

}

return l ;

}

int kthSmallest(int mat[MAX][MAX], int n, int k)

{

//Your code here

priority\_queue <int, vector<int>, greater<int>> pq;

for(int i=0; i<n; i++)

{

for(int j=0; j<n; j++)

{

pq.push(mat[i][j]);

}

}

for(int i=1; i<k; i++)

{

pq.pop();

}

return pq.top();

}

# Common elements in all rows of a given matrix

Given an m x n matrix, find all common elements present in all rows in O(mn) time and one traversal of matrix.  
**Example:**

**Input:**

mat[4][5] = {{1, 2, 1, 4, 8},

{3, 7, 8, 5, 1},

{8, 7, 7, 3, 1},

{8, 1, 2, 7, 9},

};

**Output:**

1 8 or 8 1

8 and 1 are present in all rows.

Approaches:

1. consider every element and check if it is present in all rows. If present, then print it.
2. Maintain IDX array ->Put every row occurance
3. Use Maps -> maintain distincvalue

The idea is to use maps. We initially insert all elements of the first row in an map. For every other element in remaining rows, we check if it is present in the map. If it is present in the map and is not duplicated in current row, we increment count of the element in map by 1, else we ignore the element. If the currently traversed row is the last row, we print the element if it has appeared m-1 times before.   
Below is the implementation of the idea:

// A Program to prints common element in all

// rows of matrix

#include <bits/stdc++.h>

**using** **namespace** std;

// Specify number of rows and columns

#define M 4

#define N 5

// prints common element in all rows of matrix

**void** printCommonElements(**int** mat[M][N])

{

    unordered\_map<**int**, **int**> mp;

    // initialize 1st row elements with value 1

**for** (**int** j = 0; j < N; j++)

        mp[mat[0][j]] = 1;

    // traverse the matrix

**for** (**int** i = 1; i < M; i++)

    {

**for** (**int** j = 0; j < N; j++)

        {

            // If element is present in the map and

            // is not duplicated in current row.

**if** (mp[mat[i][j]] == i)

            {

               // we increment count of the element

               // in map by 1

                mp[mat[i][j]] = i + 1;

                // If this is last row

**if** (i==M-1 && mp[mat[i][j]]==M)

                  cout << mat[i][j] << " ";

            }

        }

    }

}

// driver program to test above function

**int** main()

{

**int** mat[M][N] =

    {

        {1, 2, 1, 4, 8},

        {3, 7, 8, 5, 1},

        {8, 7, 7, 3, 1},

        {8, 1, 2, 7, 9},

    };

    printCommonElements(mat);

**return** 0;

}

The time complexity of this solution is O(m \* n) and we are doing only one traversal of the matrix.