Search in Row-wise and Column-wise sorted matrix

Given an **n x n** matrix and a number **x**, find the position of **x** in the matrix if it is present in it. Otherwise, print "Not Found". In the given matrix, every row and column is sorted in increasing order. The designed algorithm should have linear time complexity.

Example:

Input: mat[4][4] = { {10, 20, 30, 40},
15, 25, 35, 45},
{27, 29, 37, 48},
{32, 33, 39, 50}}
x = 29
Output: Found at (2, 1)
Explanation: Element at (2,1) is 29

Input : mat[4][4] = { {10, 20, 30, 40},
{15, 25, 35, 45},
{27, 29, 37, 48},
{32, 33, 39, 50}};
x = 100

> **Output :** Element not found
> **Explanation:** Element 100 is not found

Naive approach: The simple idea is to traverse the array and to search elements one by one.

Algorithm:

1. Run a nested loop, outer loop for row and inner loop for the column
2. Check every element with x and if the element is found then print "element found"
3. If the element is not found, then print "element not found".

Below is the implementation of the above approach:

C++ Java

```
// Java program to search an element in row-wise
// and column-wise sorted matrix

class GFG {
  static int search(int[][] mat, int n, int x)
  {
    if (n == 0)
      return -1;

    // traverse through the matrix
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++)
        // if the element is found
```

```
        if (mat[i][j] == x) {
          System.out.print("Element found at ("
                           + i + ", " + j
                           + ")\n");
          return 1;
        }
      }

    System.out.print(" Element not found");
    return 0;
  }
  public static void main(String[] args)
  {
    int mat[][] = { { 10, 20, 30, 40 },
                    { 15, 25, 35, 45 },
                    { 27, 29, 37, 48 },
                    { 32, 33, 39, 50 } };

    search(mat, 4, 29);
  }
}

// This code is contributed by Aditya Kumar (adityakumar129)
```


Output

```
Element found at (2, 1)
```


Time Complexity: $O(n^2)$
Auxiliary Space: O(1)

A better solution is use Divide and Conquer to find the element which has a time complexity of $O(n^{1.58})$.

Efficient approach: The simple idea is to remove a row or column in each comparison until an element is found. Start searching from the top-right corner of the matrix. There are three possible cases.

1. **The given number is greater than the current number:** This will ensure that all the elements in the current row are smaller than the given number as the pointer is already at the right-most elements and the row is sorted. Thus, the entire row gets eliminated and continues the search for the next row. Here, elimination means that a row needs not be searched.
2. **The given number is smaller than the current number:** This will ensure that all the elements in the current column are greater than the given number. Thus, the entire column gets eliminated and continues the search for the previous column, i.e. the column on the immediate left.
3. **The given number is equal to the current number:** This will end the search.

Algorithm:

1. Let the given element be x, create two variable *i = 0, j = n-1* as index of row and column
2. Run a loop until i = n
3. Check if the current element is greater than x then decrease the count of j. Exclude the current column.
4. Check if the current element is less than x then increase the count of i. Exclude the current row.
5. If the element is equal, then print the position and end.

Below is the implementation of the above approach:

C++ Java

```
// JAVA Code for Search in a row wise and
// column wise sorted matrix

class GFG {

    /* Searches the element x in mat[][]. If the
    element is found, then prints its position
    and returns true, otherwise prints "not found"
    and returns false */
    private static void search(int[][] mat,
                                    int n, int x)
    {

        // set indexes for top right
        int i = 0, j = n - 1;
        // element

        while (i < n && j >= 0)
        {
            if (mat[i][j] == x)
            {
                System.out.print("Found at " + i + " " + j);
                return;
            }
            if (mat[i][j] > x)
```

```
                j--;
            else // if mat[i][j] < x
                i++;
        }

        System.out.print("n Element not found");
        return; // if ( i==n || j== -1 )
    }
    // driver program to test above function
    public static void main(String[] args)
    {
        int mat[][] = { { 10, 20, 30, 40 },
                        { 15, 25, 35, 45 },
                        { 27, 29, 37, 48 },
                        { 32, 33, 39, 50 } };

        search(mat, 4, 29);
    }
}
// This code is contributed by Arnav Kr. Mandal.
```

Output

```
Element found at 2, 1
```

Time Complexity: O(n), Only one traversal is needed, i.e, i from 0 to n and j from n-1 to 0 with at most 2*n steps. The above approach will also work for m x n matrix (not only for n x n). Complexity would be O(m + n).
Auxiliary Space: O(1), No extra space is required.

Dash
All
Articles
Videos
Problems
Quiz