# Search element in a sorted matrix



Given a sorted matrix mat[n][m] and an element 'x'. Find position of x in the matrix if it is present, else print -1. Matrix is sorted in a way such that all elements in a row are sorted in increasing order and for row 'i', where 1 <= i <= n-1, first element of row 'i' is greater than or equal to the last element of row 'i-1'. The approach should have O(log n + log m) time complexity.

Examples:

```
Input : mat[][] = { {1, 5, 9},
                    {14, 20, 21},
                    {30, 34, 43} }
        x = 14
Output : Found at (1, 0)

Input : mat[][] = { {1, 5, 9, 11},
                    {14, 20, 21, 26},
                    {30, 34, 43, 50} }
        x = 42
Output : -1
```

**Recommended: Please try your approach on *{IDE}* first, before moving on to the solution.**

Please note that this problem is different from Search in a row wise and column wise sorted matrix. Here matrix is more strictly sorted as first element of a row is greater than last element of previous row.

A **Simple Solution** is to one by one compare x with every element of matrix. If matches, then return position. If we reach end, return -1. Time complexity of this solution is O(n x m).

An **efficient solution** is to typecast given 2D array to 1D array, then apply binary search on the typecasted array.

**Another efficient approach** that doesn't require typecasting is explained below.

```
1) Perform binary search on the middle column
   till only two elements are left or till the
   middle element of some row in the search is
   the required element 'x'. This search is done
   to skip the rows that are not required
2) The two left elements must be adjacent. Consider
   the rows of two elements and do following
   a) check whether the element 'x' equals to the
      middle element of any one of the 2 rows
   b) otherwise according to the value of the
      element 'x' check whether it is present in
      the 1st half of 1st row, 2nd half of 1st row,
      1st half of 2nd row or 2nd half of 2nd row.

Note: This approach works for the matrix n x m
      where 2 <= n. The algorithm can be modified
      for matrix 1 x m, we just need to check whether
      2nd row exists or not
```

## Example:

```
Consider:     | 1  2  3  4|
x = 3, mat =  | 5  6  7  8|   Middle column:
              | 9 10 11 12|    = {2, 6, 10, 14}
              |13 14 15 16|   perform binary search on them
                              since, x < 6, discard the
                              last 2 rows as 'a' will
                              not lie in them(sorted matrix)
Now, only two rows are left
              | 1  2  3  4|
x = 3, mat =  | 5  6  7  8|   Check whether element is present
                              on the middle elements of these
                              rows = {2, 6}
                              x != 2 or 6
If not, consider the four sub-parts
1st half of 1st row = {1}, 2nd half of 1st row = {3, 4}
1st half of 2nd row = {5}, 2nd half of 2nd row = {7, 8}

According the value of 'x' it will be searched in the
2nd half of 1st row = {3, 4} and found at (i, j): (0, 2)
```

```
// C++ implementation to search an element in a
// sorted matrix
#include <bits/stdc++.h>
using namespace std;

const int MAX = 100;

// This function does Binary search for x in i-th
// row. It does the search from mat[i][j_low] to
// mat[i][j_high]
void binarySearch(int mat[][MAX], int i, int j_low,
                                int j_high, int x)
```

```
{
    while (j_low <= j_high)
    {
        int j_mid = (j_low + j_high) / 2;

        // Element found
        if (mat[i][j_mid] == x)
        {
            cout << "Found at (" << i << ", "
                 << j_mid << ")";
            return;
        }

        else if (mat[i][j_mid] > x)
            j_high = j_mid - 1;

        else
            j_low = j_mid + 1;
    }

    // element not found
    cout << "Element no found";
}

// Function to perform binary search on the mid
// values of row to get the desired pair of rows
// where the element can be found
void sortedMatrixSearch(int mat[][MAX], int n,
                                  int m, int x)
{
    // Single row matrix
    if (n == 1)
    {
        binarySearch(mat, 0, 0, m-1, x);
        return;
    }

    // Do binary search in middle column.
    // Condition to terminate the loop when the
    // 2 desired rows are found
    int i_low = 0;
    int i_high = n-1;
    int j_mid = m/2;
    while ((i_low+1) < i_high)
    {
        int i_mid = (i_low + i_high) / 2;

        // element found
        if (mat[i_mid][j_mid] == x)
        {
            cout << "Found at (" << i_mid << ", "
                 << j_mid << ")";
            return;
        }

        else if (mat[i_mid][j_mid] > x)
            i_high = i_mid;

        else
            i_low = i_mid;
    }

    // If element is present on the mid of the
    // two rows
    if (mat[i_low][j_mid] == x)
        cout << "Found at (" << i_low << ","
             << j_mid << ")";
    else if (mat[i_low+1][j_mid] == x)
        cout << "Found at (" << (i_low+1)
             << ", " << j_mid << ")";
```

```
    // Ssearch element on 1st half of 1st row
    else if (x <= mat[i_low][j_mid-1])
        binarySearch(mat, i_low, 0, j_mid-1, x);

    // Search element on 2nd half of 1st row
    else if (x >= mat[i_low][j_mid+1]  &&
             x <= mat[i_low][m-1])
       binarySearch(mat, i_low, j_mid+1, m-1, x);

    // Search element on 1st half of 2nd row
    else if (x <= mat[i_low+1][j_mid-1])
        binarySearch(mat, i_low+1, 0, j_mid-1, x);

    // search element on 2nd half of 2nd row
    else
        binarySearch(mat, i_low+1, j_mid+1, m-1, x);
}

// Driver program to test above
int main()
{
    int n = 4, m = 5, x = 8;
    int mat[][MAX] = {{0, 6, 8, 9, 11},
                      {20, 22, 28, 29, 31},
                      {36, 38, 50, 61, 63},
                      {64, 66, 100, 122, 128}};

    sortedMatrixSearch(mat, n, m, x);
    return 0;
}
```

Run on IDE

Output:

```
Found at (2, 1)
```

Time complexity: O(log n + log m). O(Log n) time is required to find the two desired rows. Then O(Log m) time is required for binary search in one of the four parts with size equal to m/2.

This article is contributed by **Ayush Jauhari**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.



Divide and Conquer | Matrix | Binary-Search

## Recommended Posts:

Search in a row wise and column wise sorted matrix
Divide and Conquer | Set 6 (Search in a Row-wise and Column-wise Sorted 2D Array)
Print a given matrix in spiral form
Find a peak element in a 2D array
Kth smallest element in a row-wise and column-wise sorted 2D array | Set 1
Number of days after which tank will become empty
Shuffle 2n integers in format {a1, b1, a2, b2, a3, b3, ……, an, bn} without using extra space
Find index of an extra element present in one sorted array
Convex Hull (Simple Divide and Conquer Algorithm)
K-th Element of Two Sorted Arrays

(Login to Rate)

3.4 Average Difficulty : **3.4/5.0**
Based on **5** vote(s)

Basic Easy Medium Hard Expert

Add to TODO List

Mark as DONE

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

Share this post!