Search a 2D Matrix II

C++

```
class Solution {
public:
    bool searchMatrix(vector<vector<int>>& matrix, int target) {
        int row = 0, col = matrix[0].size()-1;
        while (row < matrix.size() && col >= 0) {

            if (target == matrix[row][col])
                return true;
            if (target > matrix[row][col]) {
                row++;
            }
            else{
                col--;
            }
        }
        return false;
    }
};
```

Java

```
class Solution {
    public boolean searchMatrix(int[][] matrix, int target) {
        int rows = matrix.length;
        if(rows == 0)
            return false;
        int cols = matrix[0].length;
        int x = 0, y = cols-1;
        while (x < rows && y >= 0){
            int element = matrix[x][y];
            if(element == target)
                return true;
            if(element < target)
                x++;
            else
                y--;
        }
        return false;
    }
}
```

Python

```
class Solution:
    def searchMatrix(self, matrix: List[List[int]], target: int) -> bool:
        n = len(matrix)
        m = len(matrix[0])

        i, j = 0, m - 1
        while(i < n and j >= 0):
            if matrix[i][j] == target:
                return True
            elif matrix[i][j] < target:
                i = i + 1
            else:
                j = j - 1
        return False
```

C#

```
public class Solution {
    public bool SearchMatrix(int[][] matrix, int target) {
        if (matrix==null&& matrix.Length==0)
            return false;
        int m=matrix.Length;
        int n=matrix[0].Length;
        int curRow=0;
         int curCol=n-1;
        while(curRow < m && curCol >= 0 )
        {

            if(matrix[curRow][curCol]==target)
                return true;
            else if(matrix[curRow][curCol]>target)
                curCol--;
            else
                curRow++;
        }
        return false;
    }
}
```

Javascript

```
var searchMatrix = function(matrix, target) {
    const rowLen = matrix.length;
    const colLen = matrix[0].length;

    if (rowLen < 1 || colLen < 1) {
        return false;
    }

    // start at right top corner
    let row = 0;
    let col = colLen - 1;

    while (row >= 0  && row < rowLen && col >= 0 && col < colLen) {
        const cellValue = matrix[row][col]

        if ( cellValue === target) {
            return true;
        }

           // if target is smaller, move to left else down
        if (cellValue > target) {
            col -= 1;
        } else {
            row += 1;
        }
    }

    return false;
};
```

Range Sum Query 2D - Immutable

C++

```
class NumMatrix {
private:
    vector<vector<int>> prefix;
public:
    NumMatrix(vector<vector<int>>& matrix) {
        if (matrix.size() < 1 || matrix[0].size() < 1)
            return;
        int m = matrix.size() ,n = matrix[0].size();
        prefix.resize(m);
        for (int i=0;i<m;i++)
        {
            prefix[i].resize(n);
        }

        for(int i=0;i<m;i++)
        {
            for(int j=0;j<n;j++)
            {
                prefix[i][j]=matrix[i][j];
            }
        }
        for(int i=0;i<m;i++)
        {
            for(int j=1;j<n;j++)
            {
                prefix[i][j]=prefix[i][j] + prefix[i][j-1];
            }
        }
        for(int i=1;i<m;i++)
        {
            for(int j=0;j<n;j++)
            {
                prefix[i][j]=prefix[i][j] + prefix[i-1][j];
            }
        }
    }

    int sumRegion(int row1, int col1, int row2, int col2) {
        int sum=0;
        if (row1 == 0 && col1 == 0) {
            sum = prefix[row2][col2];
        }
        else if (row1 == 0) {
            sum = prefix[row2][col2]-prefix[row2][col1-1];
        }
        else if (col1 == 0) {
            sum = prefix[row2][col2]-prefix[row1-1][col2];
        }
        else
        {
            sum = prefix[row2][col2]-prefix[row2][col1-1]-prefix[row1-1][col2]+prefix[row1-1][col1-1];
        }
        return sum;
    }
};
```

Java

```
class NumMatrix {

         int[][] prefixSum;

         public NumMatrix(int[][] matrix) {

                  if (matrix == null || matrix.length == 0)

             return;

         int n = matrix.length;

         int m = matrix[0].length;

         prefixSum = new int[n][m];

         // initialise with the same values

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

             for (int j = 0; j < m; j++){

                 prefixSum[i][j] = matrix[i][j];

             }

         }

         // row-wise prefix sum

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

             for (int j = 1; j < m; j++){

                 prefixSum[i][j] = prefixSum[i][j-1] + matrix[i][j];

             }

         }

         // col-wise prefix sum

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

             for (int j = 1; j < n; j++){

                 prefixSum[j][i] = prefixSum[j-1][i] + prefixSum[j][i];

             }

         }

          }

         public int sumRegion(int row1, int col1, int row2, int col2) {

                  int sum = 0;

         if (row1 == 0 && col1 == 0) {

             sum = prefixSum[row2][col2];

         }

         else if (row1 == 0) {

             sum = prefixSum[row2][col2] - prefixSum[row2][col1 - 1] ;

         }

         else if (col1 == 0) {

             sum = prefixSum[row2][col2] - prefixSum[row1 - 1][col2] ;

         }

         else {

             sum = prefixSum[row2][col2] - prefixSum[row2][col1 - 1] - prefixSum[row1 - 1][col2] + prefixSum[row1 - 1][col1 - 1];

         }

                  return sum;

          }

}
```

Python

```
class NumMatrix:


    # self.prefix = []
    def __init__(self, matrix: List[List[int]]):
        if len(matrix) == 0:
            self.prefix = matrix
            return
        n, m = len(matrix), len(matrix[0])
        self.prefix = [[matrix[i][j] for j in range(m)] for i in range(n)]
        for i in range(n):
            for j in range( 1 , m ):
                self.prefix[i][j] = self.prefix[i][j] + self.prefix[i][j-1]
        for i in range(m):
            for j in range( 1 , n ):
                self.prefix[j][i] = self.prefix[j][i] + self.prefix[j-1][i]


    def sumRegion(self, row1: int, col1: int, row2: int, col2: int) -> int:
        if row1 == 0 and col1 == 0:
            return self.prefix[row2][col2]
        elif row1 == 0:
            return self.prefix[row2][col2] - self.prefix[row2][col1-1]
        elif col1 == 0:
            return self.prefix[row2][col2] - self.prefix[row1-1][col2]
        else:
            return self.prefix[row2][col2] - self.prefix[row2][col1-1] -
self.prefix[row1-1][col2] + self.prefix[row1-1][col1-1]
```

Max Chunks To Make Sorted

C++

```
class Solution {
public:
    int maxChunksToSorted(vector<int>& arr) {
        int ma=0,c=0,i=0;
        for(auto it:arr)
        {
            ma=max(ma,it);
            if(ma==i)
                c++;
            i++;
        }
        return c;
    }
};
```

Java

```
class Solution {
    public int maxChunksToSorted(int[] arr) {
        int max = -1;
        int ans = 0;
        for(int i=0; i<arr.length; i++){
            max = Math.max(max, arr[i]);
            if(max == i){
                ans++;// whenever the max upto that point is equal to the index
                            // at that point, we increment the ans
            }
        }
        return ans;
    }
}
```

Python

```
class Solution:
    def maxChunksToSorted(self, arr: List[int]) -> int:
        max_ele = -99999
        ans = 0
        for i in range(len(arr)):
            max_ele = max(max_ele, arr[i])
            if max_ele == i:
                ans += 1
        return ans
```

C#

```
public class Solution {
     public int MaxChunksToSorted(int[] arr) {
            int chunks = 0;
            int currSum = 0;
            int sortedSum = 0;

            for (int i = 0; i < arr.Length; i++) {
                sortedSum += i;
                currSum += arr[i];

                if(sortedSum == currSum) {
                    chunks++;
                }
            }
            return chunks;
        }
}
```

Javascript

```
var maxChunksToSorted = function(arr) {
    let r = 0;
    let max = 0;
    for (let i = 0; i < arr.length; i++) {
        max = Math.max(max, arr[i]);
        if (i >= max) {
            r++;
        }
    }
    return r;
};
```

[Sum of all Submatrices of a Given Matrix]

→ **Code is there on geeksforgeeks**