# Search a 2D Matrix

Write an efficient algorithm that searches for a value in an m x n matrix. This matrix has the following properties:

* Integers in each row are sorted from left to right.
* The first integer of each row is greater than the last integer of the previous row.

**Example 1:**



**Input:** matrix = [[1,3,5,7],[10,11,16,20],[23,30,34,60]], target = 3

**Output:** true

**Example 2:**



**Input:** matrix = [[1,3,5,7],[10,11,16,20],[23,30,34,60]], target = 13

**Output:** false

**Constraints:**

* m == matrix.length
* n == matrix[i].length
* 1 <= m, n <= 100
* -104 <= matrix[i][j], target <= 104

Logic:

* Remember the condition…
  + Integers in each row are sorted from left to right.
  + The first integer of each row is greater than the last integer of the previous row.
* Based on the above conditions, traverse the matrix from 0th row and last column.
  + This is because row is sorted, and column is also sorted.
  + If the target is < the element, move to the left.
  + If the target is > the element, move down.
  + Traverse until out of bound. If found, return true else return false.

#include <iostream>

#include <vector>

using namespace *std*;

bool searchMatrix(*vector*<*vector*<int>>& matrix, int target) {

int rows = matrix.*size*() - 1;

int i = 0, j = matrix[0].*size*() - 1;

while (i <= rows && j >= 0)

{

if (matrix[i][j] == target) return true;

if (target > matrix[i][j])

i++; // Move Down.

else

j--; // If target is less than the element, move Left.

}

return false;

}

int main(void)

{

*vector*<*vector*<int>> matrix{ {1, 3, 5, 7}, {10, 11, 16, 20}, {23, 30, 34, 60} };

int target; *cin* >> target;

*cout* << *std*::*boolalpha* << searchMatrix(matrix, target) << *endl*;

return 0;

}