# Problem 74: Search a 2D Matrix

## Problem Information

**Difficulty:** Medium
**Acceptance Rate:** 0.00%
**Paid Only:** No

## Problem Description

You are given an

m x n

integer matrix

matrix

with the following two properties:

Each row is sorted in non-decreasing order.

The first integer of each row is greater than the last integer of the previous row.

Given an integer

target

, return

true

if

target

is in

matrix

or

false

otherwise

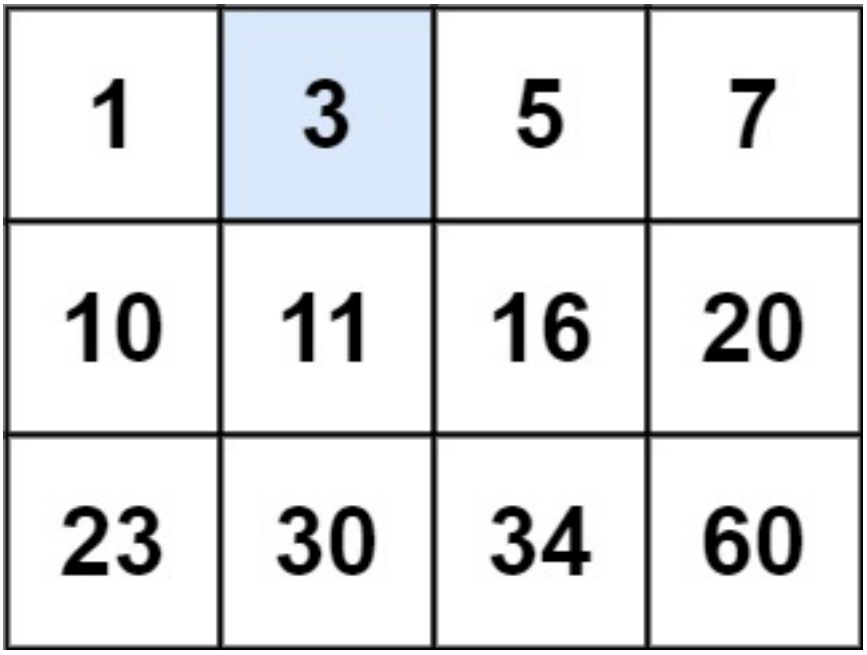
.

You must write a solution in

O(log(m * n))

time complexity.

Example 1:



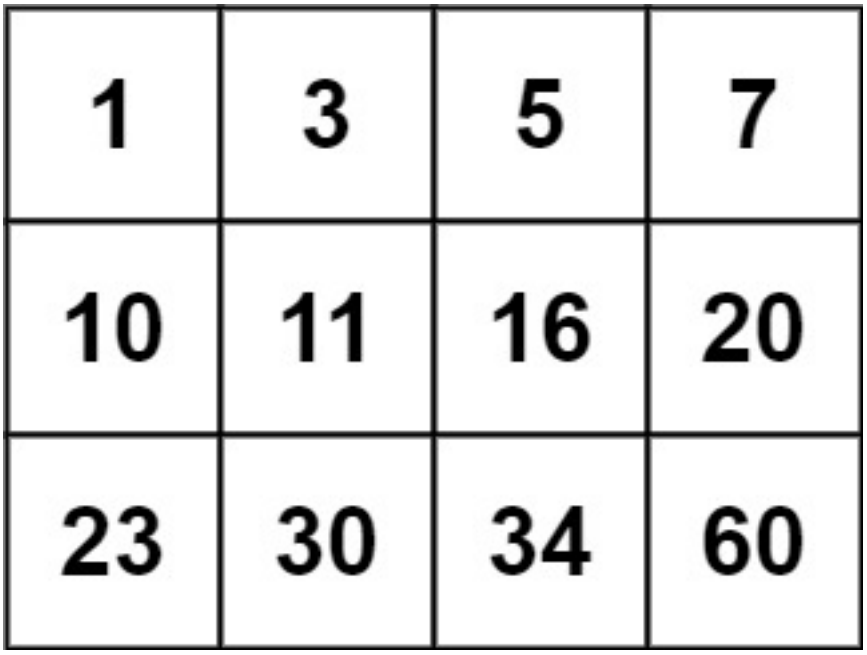

Input:

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

-10

4

<= matrix[i][j], target <= 10

4

## Code Snippets

**C++:**

```
class Solution {
public:
bool searchMatrix(vector<vector<int>>& matrix, int target) {

}
};
```

**Java:**

```
class Solution {
public boolean searchMatrix(int[][] matrix, int target) {

}
}
```

**Python3:**

```
class Solution:
def searchMatrix(self, matrix: List[List[int]], target: int) -> bool:
```

**Python:**

```
class Solution(object):
def searchMatrix(self, matrix, target):
"""
:type matrix: List[List[int]]
:type target: int
:rtype: bool
"""
```

**JavaScript:**

```
/**
* @param {number[][]} matrix
* @param {number} target
* @return {boolean}
*/
var searchMatrix = function(matrix, target) {

};
```

**TypeScript:**

```
function searchMatrix(matrix: number[][], target: number): boolean {

};
```

**C#:**

```
public class Solution {
public bool SearchMatrix(int[][] matrix, int target) {

}
}
```

**C:**

```
bool searchMatrix(int** matrix, int matrixSize, int* matrixColSize, int
target) {

}
```

**Go:**

```
func searchMatrix(matrix [][]int, target int) bool {

}
```

**Kotlin:**

```
class Solution {
fun searchMatrix(matrix: Array<IntArray>, target: Int): Boolean {

}
}
```

**Swift:**

```
class Solution {
func searchMatrix(_ matrix: [[Int]], _ target: Int) -> Bool {

}
}
```

**Rust:**

```
impl Solution {
pub fn search_matrix(matrix: Vec<Vec<i32>>, target: i32) -> bool {

}
}
```

**Ruby:**

```
# @param {Integer[][]} matrix
# @param {Integer} target
# @return {Boolean}
def search_matrix(matrix, target)

end
```

**PHP:**

```
class Solution {

/**
* @param Integer[][] $matrix
* @param Integer $target
* @return Boolean
*/
function searchMatrix($matrix, $target) {

}
}
```

**Dart:**

```
class Solution {
bool searchMatrix(List<List<int>> matrix, int target) {
```

```
}
}
```

**Scala:**

```
object Solution {
def searchMatrix(matrix: Array[Array[Int]], target: Int): Boolean = {

}
}
```

**Elixir:**

```
defmodule Solution do
@spec search_matrix(matrix :: [[integer]], target :: integer) :: boolean
def search_matrix(matrix, target) do

end
end
```

**Erlang:**

```
-spec search_matrix(Matrix :: [[integer()]], Target :: integer()) ->
boolean().
search_matrix(Matrix, Target) ->
.
```

**Racket:**

```
(define/contract (search-matrix matrix target)
(-> (listof (listof exact-integer?)) exact-integer? boolean?)
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Search a 2D Matrix
```

```
* Difficulty: Medium
* Tags: array, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
bool searchMatrix(vector<vector<int>>& matrix, int target) {

}
};
```

### Java Solution:

```
/**
* Problem: Search a 2D Matrix
* Difficulty: Medium
* Tags: array, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public boolean searchMatrix(int[][] matrix, int target) {

}
}
```

### Python3 Solution:

```
"""
Problem: Search a 2D Matrix
Difficulty: Medium
Tags: array, sort, search

Approach: Use two pointers or sliding window technique
```

```
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def searchMatrix(self, matrix: List[List[int]], target: int) -> bool:
# TODO: Implement optimized solution
pass
```

**Python Solution:**

```
class Solution(object):
def searchMatrix(self, matrix, target):
"""
:type matrix: List[List[int]]
:type target: int
:rtype: bool
"""
```

**JavaScript Solution:**

```
/**
* Problem: Search a 2D Matrix
* Difficulty: Medium
* Tags: array, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number[][]} matrix
* @param {number} target
* @return {boolean}
*/
var searchMatrix = function(matrix, target) {

};
```

**TypeScript Solution:**

```
/**
* Problem: Search a 2D Matrix
* Difficulty: Medium
* Tags: array, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function searchMatrix(matrix: number[][], target: number): boolean {

};
```

### C# Solution:

```
/*
* Problem: Search a 2D Matrix
* Difficulty: Medium
* Tags: array, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public bool SearchMatrix(int[][] matrix, int target) {

}
}
```

### C Solution:

```
/*
* Problem: Search a 2D Matrix
* Difficulty: Medium
* Tags: array, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
```

```
*/

bool searchMatrix(int** matrix, int matrixSize, int* matrixColSize, int
target) {

}
```

### Go Solution:

```
// Problem: Search a 2D Matrix
// Difficulty: Medium
// Tags: array, sort, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func searchMatrix(matrix [][]int, target int) bool {

}
```

### Kotlin Solution:

```
class Solution {
fun searchMatrix(matrix: Array<IntArray>, target: Int): Boolean {

}
}
```

### Swift Solution:

```
class Solution {
func searchMatrix(_ matrix: [[Int]], _ target: Int) -> Bool {

}
}
```

### Rust Solution:

```
// Problem: Search a 2D Matrix
// Difficulty: Medium
```

```
// Tags: array, sort, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn search_matrix(matrix: Vec<Vec<i32>>, target: i32) -> bool {

}
}
```

### Ruby Solution:

```
# @param {Integer[][]} matrix
# @param {Integer} target
# @return {Boolean}
def search_matrix(matrix, target)

end
```

### PHP Solution:

```
class Solution {

/**
* @param Integer[][] $matrix
* @param Integer $target
* @return Boolean
*/
function searchMatrix($matrix, $target) {

}
}
```

### Dart Solution:

```
class Solution {
bool searchMatrix(List<List<int>> matrix, int target) {

}
```

```
}
```

### Scala Solution:

```
object Solution {
def searchMatrix(matrix: Array[Array[Int]], target: Int): Boolean = {

}
}
```

### Elixir Solution:

```
defmodule Solution do
@spec search_matrix(matrix :: [[integer]], target :: integer) :: boolean
def search_matrix(matrix, target) do

end
end
```

### Erlang Solution:

```
-spec search_matrix(Matrix :: [[integer()]], Target :: integer()) ->
boolean().
search_matrix(Matrix, Target) ->
.
```

### Racket Solution:

```
(define/contract (search-matrix matrix target)
(-> (listof (listof exact-integer?)) exact-integer? boolean?)
)
```