Problem 3446: Sort Matrix by Diagonals

Problem Information

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

Problem Description

You are given an

n x n

square matrix of integers

grid

. Return the matrix such that:

The diagonals in the

bottom-left triangle

(including the middle diagonal) are sorted in

non-increasing order

.

The diagonals in the

top-right triangle

are sorted in

non-decreasing order

.

Example 1:

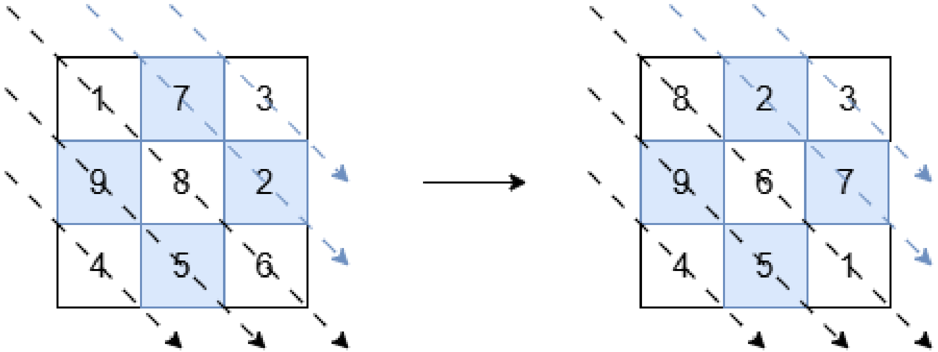
Input:

grid = [[1,7,3],[9,8,2],[4,5,6]]

Output:

[[8,2,3],[9,6,7],[4,5,1]]

Explanation:



The diagonals with a black arrow (bottom-left triangle) should be sorted in non-increasing order:

[1, 8, 6]

becomes

[8, 6, 1]

.

[9, 5]

and

[4]

remain unchanged.

The diagonals with a blue arrow (top-right triangle) should be sorted in non-decreasing order:

[7, 2]

becomes

[2, 7]

.

[3]

remains unchanged.

Example 2:

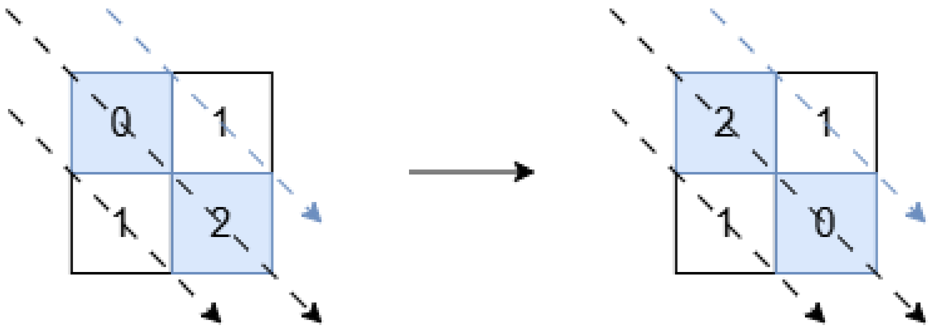
Input:

grid = [[0,1],[1,2]]

Output:

[[2,1],[1,0]]

Explanation:



The diagonals with a black arrow must be non-increasing, so

[0, 2]

is changed to

[2, 0]

. The other diagonals are already in the correct order.

Example 3:

Input:

grid = [[1]]

Output:

[[1]]

Explanation:

Diagonals with exactly one element are already in order, so no changes are needed.

Constraints:

grid.length == grid[i].length == n

1 <= n <= 10

-10

5

<= grid[i][j] <= 10

5

Code Snippets

C++:

```
class Solution {
public:
vector<vector<int>> sortMatrix(vector<vector<int>>& grid) {

}
};
```

Java:

```
class Solution {
public int[][] sortMatrix(int[][] grid) {

}
}
```

Python3:

```
class Solution:
def sortMatrix(self, grid: List[List[int]]) -> List[List[int]]:
```

Python:

```
class Solution(object):
def sortMatrix(self, grid):
"""
:type grid: List[List[int]]
:rtype: List[List[int]]
"""
```

JavaScript:

```
/**
* @param {number[][]} grid
* @return {number[][]}
*/
var sortMatrix = function(grid) {

};
```

TypeScript:

```
function sortMatrix(grid: number[][]): number[][] {

};
```

C#:

```
public class Solution {
public int[][] SortMatrix(int[][] grid) {

}
}
```

C:

```
/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** sortMatrix(int** grid, int gridSize, int* gridColSize, int* returnSize,
int** returnColumnSizes) {

}
```

Go:

```
func sortMatrix(grid [][]int) [][]int {

}
```

Kotlin:

```
class Solution {
fun sortMatrix(grid: Array<IntArray>): Array<IntArray> {

}
}
```

Swift:

```
class Solution {
func sortMatrix(_ grid: [[Int]]) -> [[Int]] {

}
}
```

Rust:

```
impl Solution {
pub fn sort_matrix(grid: Vec<Vec<i32>>) -> Vec<Vec<i32>> {

}
}
```

Ruby:

```
# @param {Integer[][]} grid
# @return {Integer[][]}
def sort_matrix(grid)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $grid
* @return Integer[][]
*/
function sortMatrix($grid) {

}
}
```

Dart:

```
class Solution {
List<List<int>> sortMatrix(List<List<int>> grid) {

}
}
```

Scala:

```
object Solution {
def sortMatrix(grid: Array[Array[Int]]): Array[Array[Int]] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec sort_matrix(grid :: [[integer]]) :: [[integer]]
def sort_matrix(grid) do

end
end
```

Erlang:

```
-spec sort_matrix(Grid :: [[integer()]]) -> [[integer()]].
sort_matrix(Grid) ->
.
```

Racket:

```
(define/contract (sort-matrix grid)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Sort Matrix by Diagonals
* Difficulty: Medium
* Tags: array, sort
*
* 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:
vector<vector<int>> sortMatrix(vector<vector<int>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Sort Matrix by Diagonals
* Difficulty: Medium
* Tags: array, sort
*
* 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 int[][] sortMatrix(int[][] grid) {

}
}
```

Python3 Solution:

```
"""
Problem: Sort Matrix by Diagonals
Difficulty: Medium
Tags: array, sort

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 sortMatrix(self, grid: List[List[int]]) -> List[List[int]]:
# TODO: Implement optimized solution
```

```

pass
```

Python Solution:

```
class Solution(object):
def sortMatrix(self, grid):
"""
:type grid: List[List[int]]
:rtype: List[List[int]]
"""
```

JavaScript Solution:

```
/**
* Problem: Sort Matrix by Diagonals
* Difficulty: Medium
* Tags: array, sort
*
* 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[][]} grid
* @return {number[][]}
*/
var sortMatrix = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Sort Matrix by Diagonals
* Difficulty: Medium
* Tags: array, sort
*
* 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 sortMatrix(grid: number[][]): number[][] {

};
```

C# Solution:

```
/*
* Problem: Sort Matrix by Diagonals
* Difficulty: Medium
* Tags: array, sort
*
* 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 int[][] SortMatrix(int[][] grid) {

}
}
```

C Solution:

```
/*
* Problem: Sort Matrix by Diagonals
* Difficulty: Medium
* Tags: array, sort
*
* 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
*/

/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
```

```
*/
int** sortMatrix(int** grid, int gridSize, int* gridColSize, int* returnSize,
int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Sort Matrix by Diagonals
// Difficulty: Medium
// Tags: array, sort
//
// 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 sortMatrix(grid [][]int) [][]int {

}
```

Kotlin Solution:

```
class Solution {
fun sortMatrix(grid: Array<IntArray>): Array<IntArray> {

}
}
```

Swift Solution:

```
class Solution {
func sortMatrix(_ grid: [[Int]]) -> [[Int]] {

}
}
```

Rust Solution:

```
// Problem: Sort Matrix by Diagonals
// Difficulty: Medium
// Tags: array, sort
```

```
//
// 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 sort_matrix(grid: Vec<Vec<i32>>) -> Vec<Vec<i32>> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} grid
# @return {Integer[][]}
def sort_matrix(grid)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $grid
* @return Integer[][]
*/
function sortMatrix($grid) {

}
}
```

Dart Solution:

```
class Solution {
List<List<int>> sortMatrix(List<List<int>> grid) {

}
}
```

Scala Solution:

```
object Solution {
def sortMatrix(grid: Array[Array[Int]]): Array[Array[Int]] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec sort_matrix(grid :: [[integer]]) :: [[integer]]
def sort_matrix(grid) do

end
end
```

Erlang Solution:

```
-spec sort_matrix(Grid :: [[integer()]]) -> [[integer()]].
sort_matrix(Grid) ->
.
```

Racket Solution:

```
(define/contract (sort-matrix grid)
(-> (listof (listof exact-integer?)) (listof (listof exact-integer?)))
)
```