# Problem 2500: Delete Greatest Value in Each Row

## Problem Information

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

## Problem Description

You are given an

m x n

matrix

grid

consisting of positive integers.

Perform the following operation until

grid

becomes empty:

Delete the element with the greatest value from each row. If multiple such elements exist, delete any of them.

Add the maximum of deleted elements to the answer.

Note

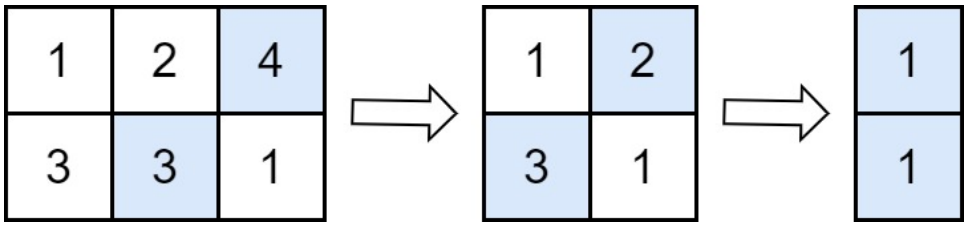
that the number of columns decreases by one after each operation.

Return

the answer after performing the operations described above

.

Example 1:




Input:

grid = [[1,2,4],[3,3,1]]

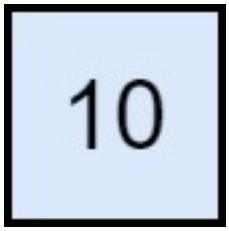
Output:

8

Explanation:

The diagram above shows the removed values in each step. - In the first operation, we remove 4 from the first row and 3 from the second row (notice that, there are two cells with value 3 and we can remove any of them). We add 4 to the answer. - In the second operation, we remove 2 from the first row and 3 from the second row. We add 3 to the answer. - In the third operation, we remove 1 from the first row and 1 from the second row. We add 1 to the answer. The final answer = 4 + 3 + 1 = 8.

Example 2:




Input:

grid = [[10]]

Output:

10

Explanation:

The diagram above shows the removed values in each step. - In the first operation, we remove 10 from the first row. We add 10 to the answer. The final answer = 10.

Constraints:

m == grid.length

n == grid[i].length

1 <= m, n <= 50

1 <= grid[i][j] <= 100

## Code Snippets

**C++:**

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

}
};
```

**Java:**

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

}
}
```

**Python3:**

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

**Python:**

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

**JavaScript:**

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

};
```

**TypeScript:**

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

};
```

**C#:**

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

}
}
```

**C:**

```
int deleteGreatestValue(int** grid, int gridSize, int* gridColSize) {

}
```

**Go:**

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

}
```

**Kotlin:**

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

}
}
```

**Swift:**

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

}
}
```

**Rust:**

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

}
}
```

**Ruby:**

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

end
```

**PHP:**

```
class Solution {

/**
```

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

}
}
```

**Dart:**

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

}
}
```

**Scala:**

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

}
}
```

**Elixir:**

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

end
end
```

**Erlang:**

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

**Racket:**

```
(define/contract (delete-greatest-value grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Delete Greatest Value in Each Row
* Difficulty: Easy
* Tags: array, sort, queue, heap
*
* 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 deleteGreatestValue(vector<vector<int>>& grid) {

}
};
```

### Java Solution:

```
/**
* Problem: Delete Greatest Value in Each Row
* Difficulty: Easy
* Tags: array, sort, queue, heap
*
* 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 deleteGreatestValue(int[][] grid) {

}
```

```
}
```

### Python3 Solution:

```
"""
Problem: Delete Greatest Value in Each Row
Difficulty: Easy
Tags: array, sort, queue, heap

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

### Python Solution:

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

### JavaScript Solution:

```
/**
* Problem: Delete Greatest Value in Each Row
* Difficulty: Easy
* Tags: array, sort, queue, heap
*
* 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 deleteGreatestValue = function(grid) {

};
```

### TypeScript Solution:

```
/**
* Problem: Delete Greatest Value in Each Row
* Difficulty: Easy
* Tags: array, sort, queue, heap
*
* 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 deleteGreatestValue(grid: number[][]): number {

};
```

### C# Solution:

```
/*
* Problem: Delete Greatest Value in Each Row
* Difficulty: Easy
* Tags: array, sort, queue, heap
*
* 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 DeleteGreatestValue(int[][] grid) {

}
}
```

### C Solution:

```
/*
* Problem: Delete Greatest Value in Each Row
* Difficulty: Easy
* Tags: array, sort, queue, heap
*
* 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
*/

int deleteGreatestValue(int** grid, int gridSize, int* gridColSize) {

}
```

### Go Solution:

```
// Problem: Delete Greatest Value in Each Row
// Difficulty: Easy
// Tags: array, sort, queue, heap
//
// 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 deleteGreatestValue(grid [][]int) int {

}
```

### Kotlin Solution:

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

}
}
```

### Swift Solution:

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

```
}
}
```

### Rust Solution:

```
// Problem: Delete Greatest Value in Each Row
// Difficulty: Easy
// Tags: array, sort, queue, heap
//
// 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 delete_greatest_value(grid: Vec<Vec<i32>>) -> i32 {

}
}
```

### Ruby Solution:

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

end
```

### PHP Solution:

```
class Solution {

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

}
}
```

### Dart Solution:

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

}
}
```

### Scala Solution:

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

}
}
```

### Elixir Solution:

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

end
end
```

### Erlang Solution:

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

### Racket Solution:

```
(define/contract (delete-greatest-value grid)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```