Problem 2679: Sum in a Matrix

Problem Information

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

Problem Description

You are given a

0-indexed

2D integer array

nums

. Initially, your score is

0

. Perform the following operations until the matrix becomes empty:

From each row in the matrix, select the largest number and remove it. In the case of a tie, it does not matter which number is chosen.

Identify the highest number amongst all those removed in step 1. Add that number to your

score

.

Return

the final

score

.

Example 1:

Input:

nums = [[7,2,1],[6,4,2],[6,5,3],[3,2,1]]

Output:

15

Explanation:

In the first operation, we remove 7, 6, 6, and 3. We then add 7 to our score. Next, we remove 2, 4, 5, and 2. We add 5 to our score. Lastly, we remove 1, 2, 3, and 1. We add 3 to our score. Thus, our final score is 7 + 5 + 3 = 15.

Example 2:

Input:

nums = [[1]]

Output:

1

Explanation:

We remove 1 and add it to the answer. We return 1.

Constraints:

1 <= nums.length <= 300

1 <= nums[i].length <= 500

0 <= nums[i][j] <= 10

3

Code Snippets

C++:

```
class Solution {
public:
int matrixSum(vector<vector<int>>& nums) {

}
};
```

Java:

```
class Solution {
public int matrixSum(int[][] nums) {

}
}
```

Python3:

```
class Solution:
def matrixSum(self, nums: List[List[int]]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function matrixSum(nums: number[][]): number {

};
```

C#:

```
public class Solution {
public int MatrixSum(int[][] nums) {

}
}
```

C:

```
int matrixSum(int** nums, int numsSize, int* numsColSize) {

}
```

Go:

```
func matrixSum(nums [][]int) int {

}
```

Kotlin:

```
class Solution {
fun matrixSum(nums: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func matrixSum(_ nums: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn matrix_sum(nums: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} nums
# @return {Integer}
def matrix_sum(nums)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $nums
* @return Integer
*/
function matrixSum($nums) {

}
}
```

Dart:

```
class Solution {
int matrixSum(List<List<int>> nums) {

}
}
```

Scala:

```
object Solution {
def matrixSum(nums: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec matrix_sum(nums :: [[integer]]) :: integer
def matrix_sum(nums) do

end
end
```

Erlang:

```
-spec matrix_sum(Nums :: [[integer()]]) -> integer().
matrix_sum(Nums) ->
.
```

Racket:

```
(define/contract (matrix-sum nums)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Sum in a Matrix
* Difficulty: Medium
* 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 matrixSum(vector<vector<int>>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Sum in a Matrix
* Difficulty: Medium
* 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 matrixSum(int[][] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Sum in a Matrix
Difficulty: Medium
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 matrixSum(self, nums: List[List[int]]) -> int:
# TODO: Implement optimized solution
```

```

pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Sum in a Matrix
* Difficulty: Medium
* 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[][]} nums
* @return {number}
*/
var matrixSum = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Sum in a Matrix
* Difficulty: Medium
* 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 matrixSum(nums: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Sum in a Matrix
* Difficulty: Medium
* 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 MatrixSum(int[][] nums) {

}
}
```

C Solution:

```
/*
* Problem: Sum in a Matrix
* Difficulty: Medium
* 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 matrixSum(int** nums, int numsSize, int* numsColSize) {

}
```

Go Solution:

```
// Problem: Sum in a Matrix
// Difficulty: Medium
// 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 matrixSum(nums [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun matrixSum(nums: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func matrixSum(_ nums: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Sum in a Matrix
// Difficulty: Medium
// 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 matrix_sum(nums: Vec<Vec<i32>>) -> i32 {

}

}
```

Ruby Solution:

```
# @param {Integer[][]} nums
# @return {Integer}
def matrix_sum(nums)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $nums
* @return Integer
*/
function matrixSum($nums) {

}
}
```

Dart Solution:

```
class Solution {
int matrixSum(List<List<int>> nums) {

}
}
```

Scala Solution:

```
object Solution {
def matrixSum(nums: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec matrix_sum(nums :: [[integer]]) :: integer
def matrix_sum(nums) do

end
end
```

Erlang Solution:

```
-spec matrix_sum(Nums :: [[integer()]]) -> integer().
matrix_sum(Nums) ->
.
```

Racket Solution:

```
(define/contract (matrix-sum nums)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```