Problem 3631: Sort Threats by Severity and Exploitability

Problem Information

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

Problem Description

You are given a 2D integer array

threats

, where each

threats[i] = [ID

i

, sev

i

, exp

i

]

ID

i

: Unique identifier of the threat.

sev

i

: Indicates the severity of the threat.

exp

i

: Indicates the exploitability of the threat.

The

score

of a threat

i

is defined as:

score = 2 × sev

i

+ exp

i

Your task is to return

threats

sorted in

descending

order of

score

.

If multiple threats have the same score, sort them by

ascending ID

.

Example 1:

Input:

threats = [[101,2,3],[102,3,2],[103,3,3]]

Output:

[[103,3,3],[102,3,2],[101,2,3]]

Explanation:

Threat

ID

sev

exp

Score = 2 × sev + exp

threats[0]

101

2

3

2 × 2 + 3 = 7

threats[1]

102

3

2

2 × 3 + 2 = 8

threats[2]

103

3

3

2 × 3 + 3 = 9

Sorted Order:

[[103, 3, 3], [102, 3, 2], [101, 2, 3]]

Example 2:

Input:

threats = [[101,4,1],[103,1,5],[102,1,5]]

Output:

[[101,4,1],[102,1,5],[103,1,5]]

Explanation:

Threat	ID	sev	exp	Score = 2 × sev + exp
threats[0]	101	4	1	2 × 4 + 1 = 9
threats[1]	103	1	5	2 × 1 + 5 = 7
threats[2]	102	1	5	

2 × 1 + 5 = 7

threats[1]

and

threats[2]

have same score, thus sort them by ascending ID.

Sorted Order:

[[101, 4, 1], [102, 1, 5], [103, 1, 5]]

Constraints:

1 <= threats.length <= 10

5

threats[i] == [ID

i

, sev

i

, exp

i

]

1 <= ID

i

<= 10

6

1 <= sev

i

<= 10

9

1 <= exp

i

<= 10

9

All

ID

i

are

unique

Code Snippets

C++:

```
class Solution {
public:
vector<vector<int>> sortThreats(vector<vector<int>>& threats) {
}
};
```

Java:

```
class Solution {
public int[][] sortThreats(int[][] threats) {

}
}
```

Python3:

```
class Solution:
def sortThreats(self, threats: List[List[int]]) -> List[List[int]]:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function sortThreats(threats: number[][]): number[][] {

};
```

C#:

```
public class Solution {
public int[][] SortThreats(int[][] threats) {

}
}
```

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** sortThreats(int** threats, int threatsSize, int* threatsColSize, int*
returnSize, int** returnColumnSizes) {

}
```

Go:

```
func sortThreats(threats [][]int) [][]int {

}
```

Kotlin:

```
class Solution {
fun sortThreats(threats: Array<IntArray>): Array<IntArray> {

}
}
```

Swift:

```
class Solution {
func sortThreats(_ threats: [[Int]]) -> [[Int]] {

}
}
```

Rust:

```
impl Solution {
pub fn sort_threats(threats: Vec<Vec<i32>>) -> Vec<Vec<i32>> {

}
}
```

Ruby:

```
# @param {Integer[][]} threats
# @return {Integer[][]}
def sort_threats(threats)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $threats
* @return Integer[][]
*/
function sortThreats($threats) {

}
}
```

Dart:

```
class Solution {
List<List<int>> sortThreats(List<List<int>> threats) {

}
}
```

Scala:

```
object Solution {
def sortThreats(threats: Array[Array[Int]]): Array[Array[Int]] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec sort_threats(threats :: [[integer]]) :: [[integer]]
def sort_threats(threats) do

end
end
```

Erlang:

```
-spec sort_threats(Threats :: [[integer()]]) -> [[integer()]].
sort_threats(Threats) ->
.
```

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Sort Threats by Severity and Exploitability
* 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>> sortThreats(vector<vector<int>>& threats) {

}
};
```

Java Solution:

```
/**
* Problem: Sort Threats by Severity and Exploitability
* 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[][] sortThreats(int[][] threats) {

}
}
```

Python3 Solution:

```
"""
Problem: Sort Threats by Severity and Exploitability
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 sortThreats(self, threats: List[List[int]]) -> List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Sort Threats by Severity and Exploitability
* 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[][]} threats
* @return {number[][]}
*/
var sortThreats = function(threats) {

};
```

TypeScript Solution:

```
/**
* Problem: Sort Threats by Severity and Exploitability
* 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 sortThreats(threats: number[][]): number[][] {

};
```

C# Solution:

```
/*
* Problem: Sort Threats by Severity and Exploitability
* 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[][] SortThreats(int[][] threats) {

}
}
```

C Solution:

```
/*
* Problem: Sort Threats by Severity and Exploitability
* 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** sortThreats(int** threats, int threatsSize, int* threatsColSize, int* returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Sort Threats by Severity and Exploitability
// 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 sortThreats(threats [][]int) [][]int {

}
```

Kotlin Solution:

```
class Solution {
fun sortThreats(threats: Array<IntArray>): Array<IntArray> {

}
}
```

Swift Solution:

```
class Solution {
func sortThreats(_ threats: [[Int]]) -> [[Int]] {

}
}
```

Rust Solution:

```
// Problem: Sort Threats by Severity and Exploitability
// 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_threats(threats: Vec<Vec<i32>>) -> Vec<Vec<i32>> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} threats
# @return {Integer[][]}
def sort_threats(threats)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $threats
* @return Integer[][]
*/
function sortThreats($threats) {

}
}
```

Dart Solution:

```
class Solution {
List<List<int>> sortThreats(List<List<int>> threats) {

}
}
```

Scala Solution:

```
object Solution {
def sortThreats(threats: Array[Array[Int]]): Array[Array[Int]] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec sort_threats(threats :: [[integer]]) :: [[integer]]
def sort_threats(threats) do

end
end
```

Erlang Solution:

```
-spec sort_threats(Threats :: [[integer()]]) -> [[integer()]].
sort_threats(Threats) ->
.
```

Racket Solution:

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