Problem 3645: Maximum Total from Optimal Activation Order

Problem Information

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

Problem Description

You are given two integer arrays

value

and

limit

, both of length

n

.

Initially, all elements are

inactive

. You may activate them in any order.

To activate an inactive element at index

i

, the number of

currently

active elements must be

strictly less

than

limit[i]

.

When you activate the element at index

i

, it adds

value[i]

to the

total

activation value (i.e., the sum of

value[i]

for all elements that have undergone activation operations).

After each activation, if the number of

currently

active elements becomes

x

, then

all

elements

j

with

limit[j] <= x

become

permanently

inactive, even if they are already active.

Return the

maximum

total

you can obtain by choosing the activation order optimally.

Example 1:

Input:

value = [3,5,8], limit = [2,1,3]

Output:

16

Explanation:

One optimal activation order is:

Step

Activated

i

value[i]

Active Before

i

Active After

i

Becomes Inactive

j

Inactive Elements

Total

1

1

5

0

1

j = 1

as

limit[1] = 1

[1]

5

2

0

3

0

1

-

[1]

8

3

2

8

1

2

j = 0

as

limit[0] = 2

[0, 1]

16

Thus, the maximum possible total is 16.

Example 2:

Input:

value = [4,2,6], limit = [1,1,1]

Output:

6

Explanation:

One optimal activation order is:

Step

Activated

i

value[i]

Active Before

i

Active After

i

Becomes Inactive

j

Inactive Elements

Total

1

2

6

0

1

j = 0, 1, 2

as

limit[j] = 1

[0, 1, 2]

6

Thus, the maximum possible total is 6.

Example 3:

Input:

value = [4,1,5,2], limit = [3,3,2,3]

Output:

12

Explanation:

One optimal activation order is:

Step

Activated

i

value[i]

Active Before

i

Active After

i

Becomes Inactive

j

Inactive Elements

Total

1

2

5

0

1

-

[ ]

5

2

0

4

1

2

j = 2

as

limit[2] = 2

[2]

9

3

1

1

1

2

-

[2]

10

4

3

2

2

3

j = 0, 1, 3

as

limit[j] = 3

[0, 1, 2, 3]

12

Thus, the maximum possible total is 12.

Constraints:

1 <= n == value.length == limit.length <= 10

5

1 <= value[i] <= 10

5

1 <= limit[i] <= n

Code Snippets

C++:

```
class Solution {
public:
long long maxTotal(vector<int>& value, vector<int>& limit) {

}
};
```

Java:

```
class Solution {
public long maxTotal(int[] value, int[] limit) {

}
}
```

Python3:

```
class Solution:
def maxTotal(self, value: List[int], limit: List[int]) -> int:
```

Python:

```
class Solution(object):
def maxTotal(self, value, limit):
"""
:type value: List[int]
:type limit: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} value
* @param {number[]} limit
* @return {number}
*/
var maxTotal = function(value, limit) {

};
```

TypeScript:

```
function maxTotal(value: number[], limit: number[]): number {

};
```

C#:

```
public class Solution {
public long MaxTotal(int[] value, int[] limit) {

}
}
```

C:

```
long long maxTotal(int* value, int valueSize, int* limit, int limitSize) {

}
```

Go:

```
func maxTotal(value []int, limit []int) int64 {

}
```

Kotlin:

```
class Solution {
fun maxTotal(value: IntArray, limit: IntArray): Long {

}
}
```

Swift:

```
class Solution {
func maxTotal(_ value: [Int], _ limit: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_total(value: Vec<i32>, limit: Vec<i32>) -> i64 {

}
}
```

Ruby:

```
# @param {Integer[]} value
# @param {Integer[]} limit
# @return {Integer}
def max_total(value, limit)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $value
* @param Integer[] $limit
* @return Integer
*/
function maxTotal($value, $limit) {

}
}
```

Dart:

```
class Solution {
int maxTotal(List<int> value, List<int> limit) {

}
}
```

Scala:

```
object Solution {
def maxTotal(value: Array[Int], limit: Array[Int]): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_total(value :: [integer], limit :: [integer]) :: integer
def max_total(value, limit) do

end
end
```

Erlang:

```
-spec max_total(Value :: [integer()], Limit :: [integer()]) -> integer().
max_total(Value, Limit) ->
.
```

Racket:

```
(define/contract (max-total value limit)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Total from Optimal Activation Order
* Difficulty: Medium
* Tags: array, greedy, 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:
long long maxTotal(vector<int>& value, vector<int>& limit) {
}
};
```

Java Solution:

```
/**
* Problem: Maximum Total from Optimal Activation Order
* Difficulty: Medium
* Tags: array, greedy, 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 long maxTotal(int[] value, int[] limit) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Total from Optimal Activation Order
Difficulty: Medium
Tags: array, greedy, 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 maxTotal(self, value: List[int], limit: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maxTotal(self, value, limit):
"""
:type value: List[int]
:type limit: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Total from Optimal Activation Order
* Difficulty: Medium
* Tags: array, greedy, 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[]} value
* @param {number[]} limit
* @return {number}
*/
var maxTotal = function(value, limit) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Total from Optimal Activation Order
* Difficulty: Medium
* Tags: array, greedy, 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 maxTotal(value: number[], limit: number[]): number {
};
```

C# Solution:

```
/*
* Problem: Maximum Total from Optimal Activation Order
* Difficulty: Medium
* Tags: array, greedy, 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 long MaxTotal(int[] value, int[] limit) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Total from Optimal Activation Order
* Difficulty: Medium
* Tags: array, greedy, 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
*/

long long maxTotal(int* value, int valueSize, int* limit, int limitSize) {

}
```

Go Solution:

```
// Problem: Maximum Total from Optimal Activation Order
// Difficulty: Medium
// Tags: array, greedy, 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 maxTotal(value []int, limit []int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun maxTotal(value: IntArray, limit: IntArray): Long {

}
}
```

Swift Solution:

```
class Solution {
func maxTotal(_ value: [Int], _ limit: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Total from Optimal Activation Order
// Difficulty: Medium
// Tags: array, greedy, 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 max_total(value: Vec<i32>, limit: Vec<i32>) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} value
# @param {Integer[]} limit
# @return {Integer}
def max_total(value, limit)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $value
* @param Integer[] $limit
* @return Integer
*/
function maxTotal($value, $limit) {

}
}
```

Dart Solution:

```
class Solution {
int maxTotal(List<int> value, List<int> limit) {

}
}
```

Scala Solution:

```
object Solution {
def maxTotal(value: Array[Int], limit: Array[Int]): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_total(value :: [integer], limit :: [integer]) :: integer
def max_total(value, limit) do

end
end
```

Erlang Solution:

```
-spec max_total(Value :: [integer()], Limit :: [integer()]) -> integer().
max_total(Value, Limit) ->
.
```

Racket Solution:

```
(define/contract (max-total value limit)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```