Problem 2557: Maximum Number of Integers to Choose From a Range II

Problem Information

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

Problem Description

You are given an integer array

banned

and two integers

n

and

maxSum

. You are choosing some number of integers following the below rules:

The chosen integers have to be in the range

[1, n]

.

Each integer can be chosen

at most once

.

The chosen integers should not be in the array

banned

.

The sum of the chosen integers should not exceed

maxSum

.

Return

the

maximum

number of integers you can choose following the mentioned rules

.

Example 1:

Input:

banned = [1,4,6], n = 6, maxSum = 4

Output:

1

Explanation:

You can choose the integer 3. 3 is in the range [1, 6], and do not appear in banned. The sum of the chosen integers is 3, which does not exceed maxSum.

Example 2:

Input:

banned = [4,3,5,6], n = 7, maxSum = 18

Output:

3

Explanation:

You can choose the integers 1, 2, and 7. All these integers are in the range [1, 7], all do not appear in banned, and their sum is 10, which does not exceed maxSum.

Constraints:

1 <= banned.length <= 10

5

1 <= banned[i] <= n <= 10

9

1 <= maxSum <= 10

15

Code Snippets

C++:

```
class Solution {
public:
int maxCount(vector<int>& banned, int n, long long maxSum) {

}
};
```

Java:

```
class Solution {
public int maxCount(int[] banned, int n, long maxSum) {

}
}
```

Python3:

```
class Solution:
def maxCount(self, banned: List[int], n: int, maxSum: int) -> int:
```

Python:

```
class Solution(object):
def maxCount(self, banned, n, maxSum):
"""
:type banned: List[int]
:type n: int
:type maxSum: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} banned
* @param {number} n
* @param {number} maxSum
* @return {number}
*/
var maxCount = function(banned, n, maxSum) {

};
```

TypeScript:

```
function maxCount(banned: number[], n: number, maxSum: number): number {

};
```

C#:

```
public class Solution {
public int MaxCount(int[] banned, int n, long maxSum) {

}
}
```

C:

```
int maxCount(int* banned, int bannedSize, int n, long long maxSum) {

}
```

Go:

```
func maxCount(banned []int, n int, maxSum int64) int {

}
```

Kotlin:

```
class Solution {
fun maxCount(banned: IntArray, n: Int, maxSum: Long): Int {

}
}
```

Swift:

```
class Solution {
func maxCount(_ banned: [Int], _ n: Int, _ maxSum: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_count(banned: Vec<i32>, n: i32, max_sum: i64) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} banned
# @param {Integer} n
# @param {Integer} max_sum
# @return {Integer}
def max_count(banned, n, max_sum)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $banned
* @param Integer $n
* @param Integer $maxSum
* @return Integer
*/
function maxCount($banned, $n, $maxSum) {

}
}
```

Dart:

```
class Solution {
int maxCount(List<int> banned, int n, int maxSum) {

}
}
```

Scala:

```
object Solution {
def maxCount(banned: Array[Int], n: Int, maxSum: Long): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_count(banned :: [integer], n :: integer, max_sum :: integer) ::
```

```
integer
def max_count(banned, n, max_sum) do

end
end
```

Erlang:

```
-spec max_count(Banned :: [integer()], N :: integer(), MaxSum :: integer())
-> integer().
max_count(Banned, N, MaxSum) ->
.
```

Racket:

```
(define/contract (max-count banned n maxSum)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Number of Integers to Choose From a Range II
* Difficulty: Medium
* Tags: array, greedy, sort, search
*
* 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 maxCount(vector<int>& banned, int n, long long maxSum) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Number of Integers to Choose From a Range II
* Difficulty: Medium
* Tags: array, greedy, sort, search
*
* 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 maxCount(int[] banned, int n, long maxSum) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Number of Integers to Choose From a Range II
Difficulty: Medium
Tags: array, greedy, sort, search

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 maxCount(self, banned: List[int], n: int, maxSum: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maxCount(self, banned, n, maxSum):
"""
:type banned: List[int]
:type n: int
:type maxSum: int
:rtype: int
```

```
"""
```

JavaScript Solution:

```
/**
* Problem: Maximum Number of Integers to Choose From a Range II
* Difficulty: Medium
* Tags: array, greedy, sort, search
*
* 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[]} banned
* @param {number} n
* @param {number} maxSum
* @return {number}
*/
var maxCount = function(banned, n, maxSum) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Number of Integers to Choose From a Range II
* Difficulty: Medium
* Tags: array, greedy, sort, search
*
* 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 maxCount(banned: number[], n: number, maxSum: number): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Number of Integers to Choose From a Range II
* Difficulty: Medium
* Tags: array, greedy, sort, search
*
* 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 MaxCount(int[] banned, int n, long maxSum) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Number of Integers to Choose From a Range II
* Difficulty: Medium
* Tags: array, greedy, sort, search
*
* 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 maxCount(int* banned, int bannedSize, int n, long long maxSum) {

}
```

Go Solution:

```
// Problem: Maximum Number of Integers to Choose From a Range II
// Difficulty: Medium
// Tags: array, greedy, sort, search
//
// 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 maxCount(banned []int, n int, maxSum int64) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxCount(banned: IntArray, n: Int, maxSum: Long): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxCount(_ banned: [Int], _ n: Int, _ maxSum: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Number of Integers to Choose From a Range II
// Difficulty: Medium
// Tags: array, greedy, sort, search
//
// 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_count(banned: Vec<i32>, n: i32, max_sum: i64) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} banned
# @param {Integer} n
# @param {Integer} max_sum
```

```
# @return {Integer}
def max_count(banned, n, max_sum)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $banned
* @param Integer $n
* @param Integer $maxSum
* @return Integer
*/
function maxCount($banned, $n, $maxSum) {

}
}
```

Dart Solution:

```
class Solution {
int maxCount(List<int> banned, int n, int maxSum) {

}
}
```

Scala Solution:

```
object Solution {
def maxCount(banned: Array[Int], n: Int, maxSum: Long): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_count(banned :: [integer], n :: integer, max_sum :: integer) ::
integer
```

```
def max_count(banned, n, max_sum) do

end
end
```

Erlang Solution:

```
-spec max_count(Banned :: [integer()], N :: integer(), MaxSum :: integer())
-> integer().
max_count(Banned, N, MaxSum) ->
.
```

Racket Solution:

```
(define/contract (max-count banned n maxSum)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
)
```