Problem 2902: Count of Sub-Multisets With Bounded Sum

Problem Information

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

Problem Description

You are given a

0-indexed

array

nums

of non-negative integers, and two integers

l

and

r

.

Return

the

count of sub-multisets

within

nums

where the sum of elements in each subset falls within the inclusive range of

[l, r]

.

Since the answer may be large, return it modulo

10

9

+ 7

.

A

sub-multiset

is an

unordered

collection of elements of the array in which a given value

x

can occur

0, 1, ..., occ[x]

times, where

occ[x]

is the number of occurrences of

x

in the array.

Note

that:

Two

sub-multisets

are the same if sorting both sub-multisets results in identical multisets.

The sum of an

empty

multiset is

0

.

Example 1:

Input:

nums = [1,2,2,3], l = 6, r = 6

Output:

1

Explanation:

The only subset of nums that has a sum of 6 is {1, 2, 3}.

Example 2:

Input:

nums = [2,1,4,2,7], l = 1, r = 5

Output:

7

Explanation:

The subsets of nums that have a sum within the range [1, 5] are {1}, {2}, {4}, {2, 2}, {1, 2}, {1, 4}, and {1, 2, 2}.

Example 3:

Input:

nums = [1,2,1,3,5,2], l = 3, r = 5

Output:

9

Explanation:

The subsets of nums that have a sum within the range [3, 5] are {3}, {5}, {1, 2}, {1, 3}, {2, 2}, {2, 3}, {1, 1, 2}, {1, 1, 3}, and {1, 2, 2}.

Constraints:

1 <= nums.length <= 2 * 10

4

0 <= nums[i] <= 2 * 10

4

Sum of

nums

does not exceed

2 * 10

4

.

0 <= l <= r <= 2 * 10

4

Code Snippets

C++:

```
class Solution {
public:
int countSubMultisets(vector<int>& nums, int l, int r) {

}
};
```

Java:

```
class Solution {
public int countSubMultisets(List<Integer> nums, int l, int r) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[]} nums
* @param {number} l
* @param {number} r
* @return {number}
*/
var countSubMultisets = function(nums, l, r) {

};
```

TypeScript:

```
function countSubMultisets(nums: number[], l: number, r: number): number {

};
```

C#:

```
public class Solution {
public int CountSubMultisets(IList<int> nums, int l, int r) {

}
}
```

C:

```
int countSubMultisets(int* nums, int numsSize, int l, int r) {

}
```

Go:

```
func countSubMultisets(nums []int, l int, r int) int {

}
```

Kotlin:

```
class Solution {
fun countSubMultisets(nums: List<Int>, l: Int, r: Int): Int {

}
}
```

Swift:

```
class Solution {
func countSubMultisets(_ nums: [Int], _ l: Int, _ r: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn count_sub_multisets(nums: Vec<i32>, l: i32, r: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} l
# @param {Integer} r
# @return {Integer}
def count_sub_multisets(nums, l, r)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $l
* @param Integer $r
* @return Integer
*/
function countSubMultisets($nums, $l, $r) {

}
}
```

Dart:

```
class Solution {
int countSubMultisets(List<int> nums, int l, int r) {

}
}
```

Scala:

```
object Solution {
def countSubMultisets(nums: List[Int], l: Int, r: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec count_sub_multisets(nums :: [integer], l :: integer, r :: integer) ::
integer
def count_sub_multisets(nums, l, r) do

end
end
```

Erlang:

```
-spec count_sub_multisets(Nums :: [integer()], L :: integer(), R ::
integer()) -> integer().
```

```
count_sub_multisets(Nums, L, R) ->
.
```

Racket:

```
(define/contract (count-sub-multisets nums l r)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count of Sub-Multisets With Bounded Sum
* Difficulty: Hard
* Tags: array, dp, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int countSubMultisets(vector<int>& nums, int l, int r) {

}
};
```

Java Solution:

```
/**
* Problem: Count of Sub-Multisets With Bounded Sum
* Difficulty: Hard
* Tags: array, dp, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
```

```
*/

class Solution {
public int countSubMultisets(List<Integer> nums, int l, int r) {

}
}
```

Python3 Solution:

```
"""
Problem: Count of Sub-Multisets With Bounded Sum
Difficulty: Hard
Tags: array, dp, hash, sort

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def countSubMultisets(self, nums: List[int], l: int, r: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count of Sub-Multisets With Bounded Sum
* Difficulty: Hard
```

```
* Tags: array, dp, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number[]} nums
* @param {number} l
* @param {number} r
* @return {number}
*/
var countSubMultisets = function(nums, l, r) {

};
```

TypeScript Solution:

```
/**
* Problem: Count of Sub-Multisets With Bounded Sum
* Difficulty: Hard
* Tags: array, dp, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function countSubMultisets(nums: number[], l: number, r: number): number {

};
```

C# Solution:

```
/*
* Problem: Count of Sub-Multisets With Bounded Sum
* Difficulty: Hard
* Tags: array, dp, hash, sort
*
* Approach: Use two pointers or sliding window technique
```

```
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int CountSubMultisets(IList<int> nums, int l, int r) {

}
}
```

C Solution:

```
/*
* Problem: Count of Sub-Multisets With Bounded Sum
* Difficulty: Hard
* Tags: array, dp, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int countSubMultisets(int* nums, int numsSize, int l, int r) {

}
```

Go Solution:

```
// Problem: Count of Sub-Multisets With Bounded Sum
// Difficulty: Hard
// Tags: array, dp, hash, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func countSubMultisets(nums []int, l int, r int) int {

}
```

Kotlin Solution:

```
class Solution {
fun countSubMultisets(nums: List<Int>, l: Int, r: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func countSubMultisets(_ nums: [Int], _ l: Int, _ r: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Count of Sub-Multisets With Bounded Sum
// Difficulty: Hard
// Tags: array, dp, hash, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn count_sub_multisets(nums: Vec<i32>, l: i32, r: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} l
# @param {Integer} r
# @return {Integer}
def count_sub_multisets(nums, l, r)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $l
* @param Integer $r
* @return Integer
*/
function countSubMultisets($nums, $l, $r) {

}
}
```

Dart Solution:

```
class Solution {
int countSubMultisets(List<int> nums, int l, int r) {

}
}
```

Scala Solution:

```
object Solution {
def countSubMultisets(nums: List[Int], l: Int, r: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_sub_multisets(nums :: [integer], l :: integer, r :: integer) ::
integer
def count_sub_multisets(nums, l, r) do

end
end
```

Erlang Solution:

```
-spec count_sub_multisets(Nums :: [integer()], L :: integer(), R ::
integer()) -> integer().
count_sub_multisets(Nums, L, R) ->
.
```

Racket Solution:

```
(define/contract (count-sub-multisets nums l r)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?)
)
```