Problem 2518: Number of Great Partitions

Problem Information

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

Problem Description

You are given an array

nums

consisting of

positive

integers and an integer

k

.

Partition

the array into two ordered

groups

such that each element is in exactly

one

group. A partition is called great if the sum

of elements of each group is greater than or equal to

k

.

Return

the number of

distinct

great partitions

. Since the answer may be too large, return it

modulo

10

9

+ 7

.

Two partitions are considered distinct if some element

nums[i]

is in different groups in the two partitions.

Example 1:

Input:

nums = [1,2,3,4], k = 4

Output:

6

Explanation:

The great partitions are: ([1,2,3], [4]), ([1,3], [2,4]), ([1,4], [2,3]), ([2,3], [1,4]), ([2,4], [1,3]) and ([4], [1,2,3]).

Example 2:

Input:

nums = [3,3,3], k = 4

Output:

0

Explanation:

There are no great partitions for this array.

Example 3:

Input:

nums = [6,6], k = 2

Output:

2

Explanation:

We can either put nums[0] in the first partition or in the second partition. The great partitions will be ([6], [6]) and ([6], [6]).

Constraints:

1 <= nums.length, k <= 1000

1 <= nums[i] <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int countPartitions(vector<int>& nums, int k) {

}
};
```

Java:

```
class Solution {
public int countPartitions(int[] nums, int k) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function countPartitions(nums: number[], k: number): number {

};
```

C#:

```
public class Solution {
public int CountPartitions(int[] nums, int k) {

}
}
```

C:

```
int countPartitions(int* nums, int numsSize, int k) {

}
```

Go:

```
func countPartitions(nums []int, k int) int {

}
```

Kotlin:

```
class Solution {
fun countPartitions(nums: IntArray, k: Int): Int {

}
}
```

Swift:

```
class Solution {
func countPartitions(_ nums: [Int], _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn count_partitions(nums: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer}
def count_partitions(nums, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer
*/
function countPartitions($nums, $k) {

}
}
```

Dart:

```
class Solution {
int countPartitions(List<int> nums, int k) {

}
}
```

Scala:

```
object Solution {
def countPartitions(nums: Array[Int], k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec count_partitions(nums :: [integer], k :: integer) :: integer
def count_partitions(nums, k) do

end
end
```

Erlang:

```
-spec count_partitions(Nums :: [integer()], K :: integer()) -> integer().
count_partitions(Nums, K) ->
.
```

Racket:

```
(define/contract (count-partitions nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Great Partitions
* Difficulty: Hard
* Tags: array, dp
*
* 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 countPartitions(vector<int>& nums, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Great Partitions
* Difficulty: Hard
* Tags: array, dp
*
* 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 countPartitions(int[] nums, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Great Partitions
Difficulty: Hard
Tags: array, dp

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 countPartitions(self, nums: List[int], k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Number of Great Partitions
* Difficulty: Hard
* Tags: array, dp
*
* 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} k
* @return {number}
*/
var countPartitions = function(nums, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Great Partitions
* Difficulty: Hard
* Tags: array, dp
*
* 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 countPartitions(nums: number[], k: number): number {

};
```

C# Solution:

```
/*
* Problem: Number of Great Partitions
* Difficulty: Hard
* Tags: array, dp
*
* 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 CountPartitions(int[] nums, int k) {

}
}
```

C Solution:

```
/*
* Problem: Number of Great Partitions
* Difficulty: Hard
* Tags: array, dp
*
* 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 countPartitions(int* nums, int numsSize, int k) {

}
```

Go Solution:

```
// Problem: Number of Great Partitions
// Difficulty: Hard
// Tags: array, dp
//
// 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 countPartitions(nums []int, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun countPartitions(nums: IntArray, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func countPartitions(_ nums: [Int], _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Number of Great Partitions
// Difficulty: Hard
// Tags: array, dp
```

```
//
// 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_partitions(nums: Vec<i32>, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} k
# @return {Integer}
def count_partitions(nums, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $k
* @return Integer
*/
function countPartitions($nums, $k) {

}
}
```

Dart Solution:

```
class Solution {
int countPartitions(List<int> nums, int k) {

}
}
```

Scala Solution:

```
object Solution {
def countPartitions(nums: Array[Int], k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_partitions(nums :: [integer], k :: integer) :: integer
def count_partitions(nums, k) do

end
end
```

Erlang Solution:

```
-spec count_partitions(Nums :: [integer()], K :: integer()) -> integer().
count_partitions(Nums, K) ->
.
```

Racket Solution:

```
(define/contract (count-partitions nums k)
(-> (listof exact-integer?) exact-integer? exact-integer?)
)
```