Problem 2741: Special Permutations

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

nums

containing

n

distinct

positive integers. A permutation of

nums

is called special if:

For all indexes

0 <= i < n - 1

, either

nums[i] % nums[i+1] == 0

or

nums[i+1] % nums[i] == 0

.

Return

the total number of special permutations.

As the answer could be large, return it

modulo

10

9

+ 7

.

Example 1:

Input:

nums = [2,3,6]

Output:

2

Explanation:

[3,6,2] and [2,6,3] are the two special permutations of nums.

Example 2:

Input:

nums = [1,4,3]

Output:

2

Explanation:

[3,1,4] and [4,1,3] are the two special permutations of nums.

Constraints:

2 <= nums.length <= 14

1 <= nums[i] <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int specialPerm(vector<int>& nums) {

}
};
```

Java:

```
class Solution {
public int specialPerm(int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function specialPerm(nums: number[]): number {

};
```

C#:

```
public class Solution {
public int SpecialPerm(int[] nums) {

}
}
```

C:

```
int specialPerm(int* nums, int numsSize) {

}
```

Go:

```
func specialPerm(nums []int) int {

}
```

Kotlin:

```
class Solution {
fun specialPerm(nums: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func specialPerm(_ nums: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn special_perm(nums: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @return {Integer}
def special_perm(nums)

end
```

PHP:

```
class Solution {

/**
```

```
* @param Integer[] $nums
* @return Integer
*/
function specialPerm($nums) {

}
}
```

Dart:

```
class Solution {
int specialPerm(List<int> nums) {

}
}
```

Scala:

```
object Solution {
def specialPerm(nums: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec special_perm(nums :: [integer]) :: integer
def special_perm(nums) do

end
end
```

Erlang:

```
-spec special_perm(Nums :: [integer()]) -> integer().
special_perm(Nums) ->
.
```

Racket:

```
(define/contract (special-perm nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Special Permutations
* Difficulty: Medium
* 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 specialPerm(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Special Permutations
* Difficulty: Medium
* 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 specialPerm(int[] nums) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Special Permutations
Difficulty: Medium
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 specialPerm(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Special Permutations
* Difficulty: Medium
* 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
* @return {number}
*/
var specialPerm = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Special Permutations
* Difficulty: Medium
* 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 specialPerm(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Special Permutations
* Difficulty: Medium
* 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 SpecialPerm(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Special Permutations
* Difficulty: Medium
* 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 specialPerm(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Special Permutations
// Difficulty: Medium
// 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 specialPerm(nums []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun specialPerm(nums: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func specialPerm(_ nums: [Int]) -> Int {
```

```
}
}
```

Rust Solution:

```
// Problem: Special Permutations
// Difficulty: Medium
// 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 special_perm(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @return {Integer}
def special_perm(nums)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
*/
function specialPerm($nums) {

}
}
```

Dart Solution:

```
class Solution {
int specialPerm(List<int> nums) {

}
}
```

Scala Solution:

```
object Solution {
def specialPerm(nums: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec special_perm(nums :: [integer]) :: integer
def special_perm(nums) do

end
end
```

Erlang Solution:

```
-spec special_perm(Nums :: [integer()]) -> integer().
special_perm(Nums) ->
.
```

Racket Solution:

```
(define/contract (special-perm nums)
(-> (listof exact-integer?) exact-integer?)
)
```