Problem 996: Number of Squareful Arrays

Problem Information

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

Problem Description

An array is

squareful

if the sum of every pair of adjacent elements is a

perfect square

.

Given an integer array nums, return

the number of permutations of

nums

that are

squareful

.

Two permutations

perm1

and

perm2

are different if there is some index

i

such that

perm1[i] != perm2[i]

.

Example 1:

Input:

nums = [1,17,8]

Output:

2

Explanation:

[1,8,17] and [17,8,1] are the valid permutations.

Example 2:

Input:

nums = [2,2,2]

Output:

1

Constraints:

1 <= nums.length <= 12

0 <= nums[i] <= 10

9

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (num-squareful-perms nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Squareful Arrays
* Difficulty: Hard
* Tags: array, dp, math, hash
*
* 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 numSquarefulPerms(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Squareful Arrays
* Difficulty: Hard
* Tags: array, dp, math, hash
*
* 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 numSquarefulPerms(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Squareful Arrays
Difficulty: Hard
Tags: array, dp, math, hash

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

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Number of Squareful Arrays
* Difficulty: Hard
* Tags: array, dp, math, hash
*
* 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 numSquarefulPerms = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Squareful Arrays
* Difficulty: Hard
* Tags: array, dp, math, hash
*
* 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 numSquarefulPerms(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Number of Squareful Arrays
* Difficulty: Hard
* Tags: array, dp, math, hash
*
* 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 NumSquarefulPerms(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Number of Squareful Arrays
* Difficulty: Hard
* Tags: array, dp, math, hash
*
* 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 numSquarefulPerms(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Number of Squareful Arrays
// Difficulty: Hard
// Tags: array, dp, math, hash
//
// 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 numSquarefulPerms(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Number of Squareful Arrays
// Difficulty: Hard
// Tags: array, dp, math, hash
//
// 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 num_squareful_perms(nums: Vec<i32>) -> i32 {

}
```

```
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (num-squareful-perms nums)
(-> (listof exact-integer?) exact-integer?)
)
```