Problem 2206: Divide Array Into Equal Pairs

Problem Information

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

Problem Description

You are given an integer array

nums

consisting of

2 * n

integers.

You need to divide

nums

into

n

pairs such that:

Each element belongs to

exactly one

pair.

The elements present in a pair are

equal

.

Return

true

if nums can be divided into

n

pairs, otherwise return

false

.

Example 1:

Input:

nums = [3,2,3,2,2,2]

Output:

true

Explanation:

There are 6 elements in nums, so they should be divided into 6 / 2 = 3 pairs. If nums is divided into the pairs (2, 2), (3, 3), and (2, 2), it will satisfy all the conditions.

Example 2:

Input:

nums = [1,2,3,4]

Output:

false

Explanation:

There is no way to divide nums into 4 / 2 = 2 pairs such that the pairs satisfy every condition.

Constraints:

nums.length == 2 * n

1 <= n <= 500

1 <= nums[i] <= 500

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public boolean divideArray(int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function divideArray(nums: number[]): boolean {

};
```

C#:

```
public class Solution {
public bool DivideArray(int[] nums) {

}
}
```

C:

```
bool divideArray(int* nums, int numsSize) {

}
```

Go:

```
func divideArray(nums []int) bool {

}
```

Kotlin:

```
class Solution {
fun divideArray(nums: IntArray): Boolean {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
bool divideArray(List<int> nums) {

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (divide-array nums)
(-> (listof exact-integer?) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Divide Array Into Equal Pairs
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
bool divideArray(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Divide Array Into Equal Pairs
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public boolean divideArray(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Divide Array Into Equal Pairs
Difficulty: Easy
Tags: array, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Divide Array Into Equal Pairs
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

TypeScript Solution:

```
/**
* Problem: Divide Array Into Equal Pairs
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function divideArray(nums: number[]): boolean {

};
```

C# Solution:

```
/*
* Problem: Divide Array Into Equal Pairs
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public bool DivideArray(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Divide Array Into Equal Pairs
* Difficulty: Easy
* Tags: array, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

bool divideArray(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Divide Array Into Equal Pairs
// Difficulty: Easy
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func divideArray(nums []int) bool {

}
```

Kotlin Solution:

```
class Solution {
fun divideArray(nums: IntArray): Boolean {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Divide Array Into Equal Pairs
// Difficulty: Easy
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn divide_array(nums: Vec<i32>) -> bool {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
bool divideArray(List<int> nums) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (divide-array nums)
(-> (listof exact-integer?) boolean?)
)
```