Problem 2369: Check if There is a Valid Partition For The Array

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

nums

. You have to partition the array into one or more

contiguous

subarrays.

We call a partition of the array

valid

if each of the obtained subarrays satisfies

one

of the following conditions:

The subarray consists of

exactly

2,

equal elements. For example, the subarray

[2,2]

is good.

The subarray consists of

exactly

3,

equal elements. For example, the subarray

[4,4,4]

is good.

The subarray consists of

exactly

3

consecutive increasing elements, that is, the difference between adjacent elements is

1

. For example, the subarray

[3,4,5]

is good, but the subarray

[1,3,5]

is not.

Return

true

if the array has

at least

one valid partition

. Otherwise, return

false

.

Example 1:

Input:

nums = [4,4,4,5,6]

Output:

true

Explanation:

The array can be partitioned into the subarrays [4,4] and [4,5,6]. This partition is valid, so we return true.

Example 2:

Input:

nums = [1,1,1,2]

Output:

false

Explanation:

There is no valid partition for this array.

Constraints:

2 <= nums.length <= 10

5

1 <= nums[i] <= 10

6

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

/**
```

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (valid-partition nums)
(-> (listof exact-integer?) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Check if There is a Valid Partition For The Array
* 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:
bool validPartition(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Check if There is a Valid Partition For The Array
* 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 boolean validPartition(int[] nums) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Check if There is a Valid Partition For The Array
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 validPartition(self, nums: List[int]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Check if There is a Valid Partition For The Array
* 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 {boolean}
*/
var validPartition = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Check if There is a Valid Partition For The Array
* 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 validPartition(nums: number[]): boolean {

};
```

C# Solution:

```
/*
* Problem: Check if There is a Valid Partition For The Array
* 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 bool ValidPartition(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Check if There is a Valid Partition For The Array
* 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
*/

bool validPartition(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Check if There is a Valid Partition For The Array
// 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 validPartition(nums []int) bool {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

```
}
}
```

Rust Solution:

```
// Problem: Check if There is a Valid Partition For The Array
// 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 valid_partition(nums: Vec<i32>) -> bool {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (valid-partition nums)
(-> (listof exact-integer?) boolean?)
)
```