Problem 915: Partition Array into Disjoint Intervals

Problem Information

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

Problem Description

Given an integer array

nums

, partition it into two (contiguous) subarrays

left

and

right

so that:

Every element in

left

is less than or equal to every element in

right

.

left

and

right

are non-empty.

left

has the smallest possible size.

Return

the length of

left

after such a partitioning

.

Test cases are generated such that partitioning exists.

Example 1:

Input:

nums = [5,0,3,8,6]

Output:

3

Explanation:

left = [5,0,3], right = [8,6]

Example 2:

Input:

nums = [1,1,1,0,6,12]

Output:

4

Explanation:

left = [1,1,1,0], right = [6,12]

Constraints:

2 <= nums.length <= 10

5

0 <= nums[i] <= 10

6

There is at least one valid answer for the given input.

Code Snippets

C++:

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

}
};
```

Java:

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

```
}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Partition Array into Disjoint Intervals
* Difficulty: Medium
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Partition Array into Disjoint Intervals
* Difficulty: Medium
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

```
}
}
```

Python3 Solution:

```
"""
Problem: Partition Array into Disjoint Intervals
Difficulty: Medium
Tags: array

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Partition Array into Disjoint Intervals
* Difficulty: Medium
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
```

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

};
```

TypeScript Solution:

```
/**
* Problem: Partition Array into Disjoint Intervals
* Difficulty: Medium
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function partitionDisjoint(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Partition Array into Disjoint Intervals
* Difficulty: Medium
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int PartitionDisjoint(int[] nums) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Partition Array into Disjoint Intervals
* Difficulty: Medium
* Tags: array
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

int partitionDisjoint(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Partition Array into Disjoint Intervals
// Difficulty: Medium
// Tags: array
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func partitionDisjoint(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Partition Array into Disjoint Intervals
// Difficulty: Medium
// Tags: array
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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