Problem 3690: Split and Merge Array Transformation

Problem Information

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

Problem Description

You are given two integer arrays

nums1

and

nums2

, each of length

n

. You may perform the following

split-and-merge operation

on

nums1

any number of times:

Choose a subarray

nums1[L..R]

.

Remove that subarray, leaving the prefix

nums1[0..L-1]

(empty if

L = 0

) and the suffix

nums1[R+1..n-1]

(empty if

R = n - 1

).

Re-insert the removed subarray (in its original order) at

any

position in the remaining array (i.e., between any two elements, at the very start, or at the very end).

Return the

minimum

number of

split-and-merge operations

needed to transform

nums1

into

nums2

.

Example 1:

Input:

nums1 = [3,1,2], nums2 = [1,2,3]

Output:

1

Explanation:

Split out the subarray

[3]

(

L = 0

,

R = 0

); the remaining array is

[1,2]

.

Insert

[3]

at the end; the array becomes

[1,2,3]

.

Example 2:

Input:

nums1 =

[1,1,2,3,4,5]

, nums2 =

[5,4,3,2,1,1]

Output:

3

Explanation:

Remove

[1,1,2]

at indices

0 - 2

; remaining is

[3,4,5]

; insert

[1,1,2]

at position

2

, resulting in

[3,4,1,1,2,5]

.

Remove

[4,1,1]

at indices

1 - 3

; remaining is

[3,2,5]

; insert

[4,1,1]

at position

3

, resulting in

[3,2,5,4,1,1]

.

Remove

[3,2]

at indices

0 - 1

; remaining is

[5,4,1,1]

; insert

[3,2]

at position

2

, resulting in

[5,4,3,2,1,1]

.

Constraints:

2 <= n == nums1.length == nums2.length <= 6

-10

5

<= nums1[i], nums2[i] <= 10

5

nums2

is a

permutation

of

nums1

.

Code Snippets

C++:

```
class Solution {
public:
int minSplitMerge(vector<int>& nums1, vector<int>& nums2) {

}
};
```

Java:

```
class Solution {
public int minSplitMerge(int[] nums1, int[] nums2) {

}
}
```

Python3:

```
class Solution:
def minSplitMerge(self, nums1: List[int], nums2: List[int]) -> int:
```

Python:

```
class Solution(object):
def minSplitMerge(self, nums1, nums2):
"""
:type nums1: List[int]
```

```
:type nums2: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[]} nums1
* @param {number[]} nums2
* @return {number}
*/
var minSplitMerge = function(nums1, nums2) {

};
```

TypeScript:

```
function minSplitMerge(nums1: number[], nums2: number[]): number {

};
```

C#:

```
public class Solution {
public int MinSplitMerge(int[] nums1, int[] nums2) {

}
}
```

C:

```
int minSplitMerge(int* nums1, int nums1Size, int* nums2, int nums2Size) {

}
```

Go:

```
func minSplitMerge(nums1 []int, nums2 []int) int {

}
```

Kotlin:

```
class Solution {
fun minSplitMerge(nums1: IntArray, nums2: IntArray): Int {

}
}
```

Swift:

```
class Solution {
func minSplitMerge(_ nums1: [Int], _ nums2: [Int]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_split_merge(nums1: Vec<i32>, nums2: Vec<i32>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[]} nums1
# @param {Integer[]} nums2
# @return {Integer}
def min_split_merge(nums1, nums2)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums1
* @param Integer[] $nums2
* @return Integer
*/
function minSplitMerge($nums1, $nums2) {

}
```

```
}
```

Dart:

```
class Solution {
int minSplitMerge(List<int> nums1, List<int> nums2) {

}
}
```

Scala:

```
object Solution {
def minSplitMerge(nums1: Array[Int], nums2: Array[Int]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_split_merge(nums1 :: [integer], nums2 :: [integer]) :: integer
def min_split_merge(nums1, nums2) do

end
end
```

Erlang:

```
-spec min_split_merge(Nums1 :: [integer()], Nums2 :: [integer()]) ->
integer().
min_split_merge(Nums1, Nums2) ->
.
```

Racket:

```
(define/contract (min-split-merge nums1 nums2)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Split and Merge Array Transformation
* Difficulty: Medium
* Tags: array, hash, search
*
* 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:
int minSplitMerge(vector<int>& nums1, vector<int>& nums2) {

}
};
```

Java Solution:

```
/**
* Problem: Split and Merge Array Transformation
* Difficulty: Medium
* Tags: array, hash, search
*
* 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 int minSplitMerge(int[] nums1, int[] nums2) {

}
}
```

Python3 Solution:

```
"""
Problem: Split and Merge Array Transformation
```

```
Difficulty: Medium
Tags: array, hash, search

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 minSplitMerge(self, nums1: List[int], nums2: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minSplitMerge(self, nums1, nums2):
"""
:type nums1: List[int]
:type nums2: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Split and Merge Array Transformation
* Difficulty: Medium
* Tags: array, hash, search
*
* 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[]} nums1
* @param {number[]} nums2
* @return {number}
*/
var minSplitMerge = function(nums1, nums2) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Split and Merge Array Transformation
* Difficulty: Medium
* Tags: array, hash, search
*
* 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 minSplitMerge(nums1: number[], nums2: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Split and Merge Array Transformation
* Difficulty: Medium
* Tags: array, hash, search
*
* 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 int MinSplitMerge(int[] nums1, int[] nums2) {

}
}
```

C Solution:

```
/*
* Problem: Split and Merge Array Transformation
```

```
* Difficulty: Medium
* Tags: array, hash, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

int minSplitMerge(int* nums1, int nums1Size, int* nums2, int nums2Size) {

}
```

Go Solution:

```
// Problem: Split and Merge Array Transformation
// Difficulty: Medium
// Tags: array, hash, search
//
// 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 minSplitMerge(nums1 []int, nums2 []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minSplitMerge(nums1: IntArray, nums2: IntArray): Int {

}
}
```

Swift Solution:

```
class Solution {
func minSplitMerge(_ nums1: [Int], _ nums2: [Int]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Split and Merge Array Transformation
// Difficulty: Medium
// Tags: array, hash, search
//
// 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 min_split_merge(nums1: Vec<i32>, nums2: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums1
# @param {Integer[]} nums2
# @return {Integer}
def min_split_merge(nums1, nums2)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums1
* @param Integer[] $nums2
* @return Integer
*/
function minSplitMerge($nums1, $nums2) {

}
}
```

Dart Solution:

```
class Solution {
int minSplitMerge(List<int> nums1, List<int> nums2) {

}
}
```

Scala Solution:

```
object Solution {
def minSplitMerge(nums1: Array[Int], nums2: Array[Int]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_split_merge(nums1 :: [integer], nums2 :: [integer]) :: integer
def min_split_merge(nums1, nums2) do

end
end
```

Erlang Solution:

```
-spec min_split_merge(Nums1 :: [integer()], Nums2 :: [integer()]) ->
integer().
min_split_merge(Nums1, Nums2) ->
.
```

Racket Solution:

```
(define/contract (min-split-merge nums1 nums2)
(-> (listof exact-integer?) (listof exact-integer?) exact-integer?)
)
```