Problem 3507: Minimum Pair Removal to Sort Array I

Problem Information

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

Problem Description

Given an array

nums

, you can perform the following operation any number of times:

Select the

adjacent

pair with the

minimum

sum in

nums

. If multiple such pairs exist, choose the leftmost one.

Replace the pair with their sum.

Return the

minimum number of operations

needed to make the array

non-decreasing

.

An array is said to be

non-decreasing

if each element is greater than or equal to its previous element (if it exists).

Example 1:

Input:

nums = [5,2,3,1]

Output:

2

Explanation:

The pair

(3,1)

has the minimum sum of 4. After replacement,

nums = [5,2,4]

.

The pair

(2,4)

has the minimum sum of 6. After replacement,

nums = [5,6]

.

The array

nums

became non-decreasing in two operations.

Example 2:

Input:

nums = [1,2,2]

Output:

0

Explanation:

The array

nums

is already sorted.

Constraints:

1 <= nums.length <= 50

-1000 <= nums[i] <= 1000

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (minimum-pair-removal nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Pair Removal to Sort Array I
* Difficulty: Easy
* Tags: array, hash, sort, linked_list, queue, heap
*
* 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 minimumPairRemoval(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Pair Removal to Sort Array I
* Difficulty: Easy
* Tags: array, hash, sort, linked_list, queue, heap
*
* 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 minimumPairRemoval(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Pair Removal to Sort Array I
Difficulty: Easy
Tags: array, hash, sort, linked_list, queue, heap

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

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Pair Removal to Sort Array I
* Difficulty: Easy
* Tags: array, hash, sort, linked_list, queue, heap
*
* 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 {number}
*/
var minimumPairRemoval = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Pair Removal to Sort Array I
* Difficulty: Easy
* Tags: array, hash, sort, linked_list, queue, heap
*
* 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 minimumPairRemoval(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Pair Removal to Sort Array I
* Difficulty: Easy
* Tags: array, hash, sort, linked_list, queue, heap
*
* 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 MinimumPairRemoval(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Pair Removal to Sort Array I
* Difficulty: Easy
* Tags: array, hash, sort, linked_list, queue, heap
*
* 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 minimumPairRemoval(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Minimum Pair Removal to Sort Array I
// Difficulty: Easy
// Tags: array, hash, sort, linked_list, queue, heap
//
// 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 minimumPairRemoval(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Pair Removal to Sort Array I
// Difficulty: Easy
// Tags: array, hash, sort, linked_list, queue, heap
//
// 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 minimum_pair_removal(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (minimum-pair-removal nums)
(-> (listof exact-integer?) exact-integer?)
)
```