# Problem 2856: Minimum Array Length After Pair Removals

## Problem Information

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

## Problem Description

Given an integer array

num

sorted in non-decreasing order.

You can perform the following operation any number of times:

Choose

two

indices,

i

and

j

, where

nums[i] < nums[j]

.

Then, remove the elements at indices

i

and

j

from

nums

. The remaining elements retain their original order, and the array is re-indexed.

Return the

minimum

length of

nums

after applying the operation zero or more times.

Example 1:

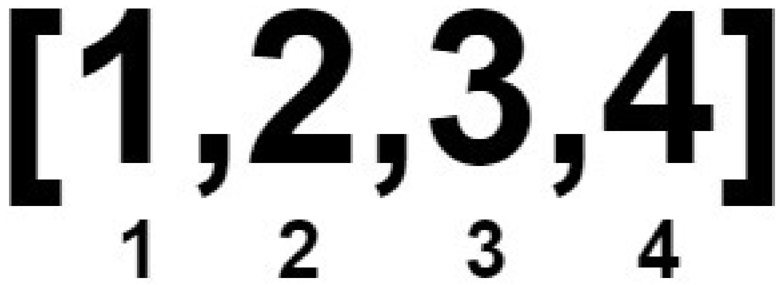
Input:

nums = [1,2,3,4]

Output:

0

Explanation:



Example 2:

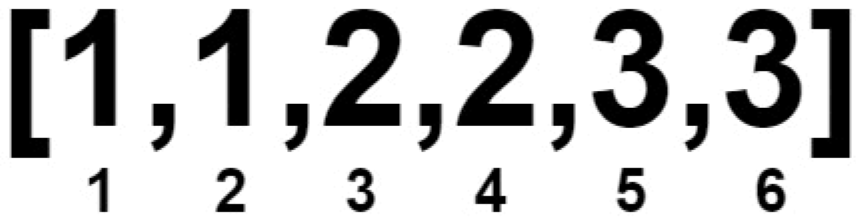
Input:

nums = [1,1,2,2,3,3]

Output:

0

Explanation:



Example 3:

Input:

nums = [1000000000,1000000000]

Output:

2

Explanation:

Since both numbers are equal, they cannot be removed.

Example 4:

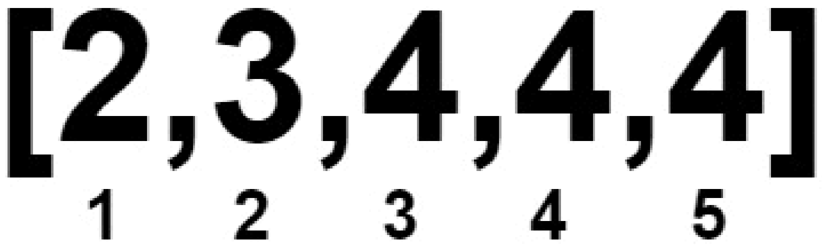
Input:

nums = [2,3,4,4,4]

Output:

1

Explanation:



Constraints:

1 <= nums.length <= 10

5

1 <= nums[i] <= 10

9

nums

is sorted in

non-decreasing

order.

## Code Snippets

### C++:

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

}
};
```

### Java:

```
class Solution {
public int minLengthAfterRemovals(List<Integer> nums) {

}
}
```

### Python3:

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

### Python:

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

### JavaScript:

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

};
```

### TypeScript:

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

};
```

### C#:

```
public class Solution {
public int MinLengthAfterRemovals(IList<int> nums) {

}
}
```

### C:

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

}
```

### Go:

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

}
```

### Kotlin:

```
class Solution {
fun minLengthAfterRemovals(nums: List<Int>): Int {

}
}
```

### Swift:

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

}

}
```

**Rust:**

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

}
}
```

**Ruby:**

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

end
```

**PHP:**

```
class Solution {

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

}
}
```

**Dart:**

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

}
}
```

**Scala:**

```
object Solution {
def minLengthAfterRemovals(nums: List[Int]): Int = {

}
}
```

### Elixir:

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

end
end
```

### Erlang:

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

### Racket:

```
(define/contract (min-length-after-removals nums)
(-> (listof exact-integer?) exact-integer?)
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Minimum Array Length After Pair Removals
* Difficulty: Medium
* Tags: array, greedy, hash, sort, 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 minLengthAfterRemovals(vector<int>& nums) {

}
};
```

### Java Solution:

```
/**
* Problem: Minimum Array Length After Pair Removals
* Difficulty: Medium
* Tags: array, greedy, hash, sort, 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 minLengthAfterRemovals(List<Integer> nums) {

}
}
```

### Python3 Solution:

```
"""
Problem: Minimum Array Length After Pair Removals
Difficulty: Medium
Tags: array, greedy, hash, sort, 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 minLengthAfterRemovals(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

### Python Solution:

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

### JavaScript Solution:

```
/**
* Problem: Minimum Array Length After Pair Removals
* Difficulty: Medium
* Tags: array, greedy, hash, sort, 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[]} nums
* @return {number}
*/
var minLengthAfterRemovals = function(nums) {

};
```

### TypeScript Solution:

```
/**
* Problem: Minimum Array Length After Pair Removals
* Difficulty: Medium
* Tags: array, greedy, hash, sort, 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 minLengthAfterRemovals(nums: number[]): number {


};
```

### C# Solution:

```
/*
* Problem: Minimum Array Length After Pair Removals
* Difficulty: Medium
* Tags: array, greedy, hash, sort, 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 MinLengthAfterRemovals(IList<int> nums) {

}
}
```

### C Solution:

```
/*
* Problem: Minimum Array Length After Pair Removals
* Difficulty: Medium
* Tags: array, greedy, hash, sort, 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 minLengthAfterRemovals(int* nums, int numsSize) {

}
```

### Go Solution:

```
// Problem: Minimum Array Length After Pair Removals
// Difficulty: Medium
// Tags: array, greedy, hash, sort, 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 minLengthAfterRemovals(nums []int) int {

}
```

### Kotlin Solution:

```
class Solution {
fun minLengthAfterRemovals(nums: List<Int>): Int {

}
}
```

### Swift Solution:

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

}
}
```

### Rust Solution:

```
// Problem: Minimum Array Length After Pair Removals
// Difficulty: Medium
// Tags: array, greedy, hash, sort, 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_length_after_removals(nums: Vec<i32>) -> i32 {

}
}
```

### Ruby Solution:

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

end
```

### PHP Solution:

```php
class Solution {

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

}
}
```

### Dart Solution:

```dart
class Solution {
int minLengthAfterRemovals(List<int> nums) {

}
}
```

### Scala Solution:

```scala
object Solution {
def minLengthAfterRemovals(nums: List[Int]): Int = {

}
}
```

### Elixir Solution:

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

```


end
end
```

## Erlang Solution:

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

## Racket Solution:

```
(define/contract (min-length-after-removals nums)
(-> (listof exact-integer?) exact-integer?)
)
```