Problem 2216: Minimum Deletions to Make Array Beautiful

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

nums

. The array

nums

is

beautiful

if:

nums.length

is even.

nums[i] != nums[i + 1]

for all

i % 2 == 0

.

Note that an empty array is considered beautiful.

You can delete any number of elements from

nums

. When you delete an element, all the elements to the right of the deleted element will be

shifted one unit to the left

to fill the gap created and all the elements to the left of the deleted element will remain

unchanged

.

Return

the

minimum

number of elements to delete from

nums

to make it

beautiful.

Example 1:

Input:

nums = [1,1,2,3,5]

Output:

1

Explanation:

You can delete either

nums[0]

or

nums[1]

to make

nums

= [1,2,3,5] which is beautiful. It can be proven you need at least 1 deletion to make

nums

beautiful.

Example 2:

Input:

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

Output:

2

Explanation:

You can delete

nums[0]

and

nums[5]

to make nums = [1,2,2,3] which is beautiful. It can be proven you need at least 2 deletions to make nums beautiful.

Constraints:

1 <= nums.length <= 10

5

0 <= nums[i] <= 10

5

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @return Integer
```

```
*/
function minDeletion($nums) {

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (min-deletion nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Deletions to Make Array Beautiful
* Difficulty: Medium
* Tags: array, greedy, stack
*
* 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 minDeletion(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Deletions to Make Array Beautiful
* Difficulty: Medium
* Tags: array, greedy, stack
*
* 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 minDeletion(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Deletions to Make Array Beautiful
Difficulty: Medium
Tags: array, greedy, stack

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Deletions to Make Array Beautiful
* Difficulty: Medium
* Tags: array, greedy, stack
*
* 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 minDeletion = function(nums) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Minimum Deletions to Make Array Beautiful
* Difficulty: Medium
* Tags: array, greedy, stack
*
* 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 minDeletion(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Deletions to Make Array Beautiful
* Difficulty: Medium
* Tags: array, greedy, stack
*
* 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 MinDeletion(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Deletions to Make Array Beautiful
* Difficulty: Medium
```

```
* Tags: array, greedy, stack
*
* 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 minDeletion(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Minimum Deletions to Make Array Beautiful
// Difficulty: Medium
// Tags: array, greedy, stack
//
// 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 minDeletion(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Deletions to Make Array Beautiful
// Difficulty: Medium
// Tags: array, greedy, stack
//
// 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 min_deletion(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

```
}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (min-deletion nums)
(-> (listof exact-integer?) exact-integer?)
)
```