Problem 3469: Find Minimum Cost to Remove Array Elements

Problem Information

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

Problem Description

You are given an integer array

nums

. Your task is to remove

all elements

from the array by performing one of the following operations at each step until

nums

is empty:

Choose any two elements from the first three elements of

nums

and remove them. The cost of this operation is the

maximum

of the two elements removed.

If fewer than three elements remain in

nums

, remove all the remaining elements in a single operation. The cost of this operation is the

maximum

of the remaining elements.

Return the

minimum

cost required to remove all the elements.

Example 1:

Input:

nums = [6,2,8,4]

Output:

12

Explanation:

Initially,

nums = [6, 2, 8, 4]

.

In the first operation, remove

nums[0] = 6

and

nums[2] = 8

with a cost of

max(6, 8) = 8

. Now,

nums = [2, 4]

.

In the second operation, remove the remaining elements with a cost of

max(2, 4) = 4

.

The cost to remove all elements is

8 + 4 = 12

. This is the minimum cost to remove all elements in

nums

. Hence, the output is 12.

Example 2:

Input:

nums = [2,1,3,3]

Output:

5

Explanation:

Initially,

nums = [2, 1, 3, 3]

.

In the first operation, remove

nums[0] = 2

and

nums[1] = 1

with a cost of

max(2, 1) = 2

. Now,

nums = [3, 3]

.

In the second operation remove the remaining elements with a cost of

max(3, 3) = 3

.

The cost to remove all elements is

2 + 3 = 5

. This is the minimum cost to remove all elements in

nums

. Hence, the output is 5.

Constraints:

1 <= nums.length <= 1000

1 <= nums[i] <= 10

6

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

```
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Find Minimum Cost to Remove Array Elements
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Find Minimum Cost to Remove Array Elements
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
}
```

Python3 Solution:

```
"""
Problem: Find Minimum Cost to Remove Array Elements
Difficulty: Medium
Tags: array, dp
```

```
Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find Minimum Cost to Remove Array Elements
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Find Minimum Cost to Remove Array Elements
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

C# Solution:

```
/*
* Problem: Find Minimum Cost to Remove Array Elements
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
}
```

C Solution:

```
/*
* Problem: Find Minimum Cost to Remove Array Elements
* Difficulty: Medium
* Tags: array, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
```

```
*/

int minCost(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Find Minimum Cost to Remove Array Elements
// Difficulty: Medium
// Tags: array, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func minCost(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Find Minimum Cost to Remove Array Elements
// Difficulty: Medium
// Tags: array, dp
```

```
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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