Problem 3010: Divide an Array Into Subarrays With Minimum Cost I

Problem Information

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

Problem Description

You are given an array of integers

nums

of length

n

.

The

cost

of an array is the value of its

first

element. For example, the cost of

[1,2,3]

is

1

while the cost of

[3,4,1]

is

3

.

You need to divide

nums

into

3

disjoint contiguous

subarrays

.

Return

the

minimum

possible

sum

of the cost of these subarrays

.

Example 1:

Input:

nums = [1,2,3,12]

Output:

6

Explanation:

The best possible way to form 3 subarrays is: [1], [2], and [3,12] at a total cost of 1 + 2 + 3 = 6. The other possible ways to form 3 subarrays are: - [1], [2,3], and [12] at a total cost of 1 + 2 + 12 = 15. - [1,2], [3], and [12] at a total cost of 1 + 3 + 12 = 16.

Example 2:

Input:

nums = [5,4,3]

Output:

12

Explanation:

The best possible way to form 3 subarrays is: [5], [4], and [3] at a total cost of 5 + 4 + 3 = 12. It can be shown that 12 is the minimum cost achievable.

Example 3:

Input:

nums = [10,3,1,1]

Output:

12

Explanation:

The best possible way to form 3 subarrays is: [10,3], [1], and [1] at a total cost of 10 + 1 + 1 = 12. It can be shown that 12 is the minimum cost achievable.

Constraints:

3 <= n <= 50

1 <= nums[i] <= 50

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
```

```
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Divide an Array Into Subarrays With Minimum Cost I
* Difficulty: Easy
* Tags: array, sort
*
* 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 minimumCost(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Divide an Array Into Subarrays With Minimum Cost I
* Difficulty: Easy
* Tags: array, sort
*
* 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 minimumCost(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Divide an Array Into Subarrays With Minimum Cost I
Difficulty: Easy
Tags: array, sort
```

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Divide an Array Into Subarrays With Minimum Cost I
* Difficulty: Easy
* Tags: array, sort
*
* 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 minimumCost = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Divide an Array Into Subarrays With Minimum Cost I
* Difficulty: Easy
* Tags: array, sort
*
* 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 minimumCost(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Divide an Array Into Subarrays With Minimum Cost I
* Difficulty: Easy
* Tags: array, sort
*
* 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 MinimumCost(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Divide an Array Into Subarrays With Minimum Cost I
* Difficulty: Easy
* Tags: array, sort
*
* 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 minimumCost(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Divide an Array Into Subarrays With Minimum Cost I
// Difficulty: Easy
// Tags: array, sort
//
// 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 minimumCost(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Divide an Array Into Subarrays With Minimum Cost I
// Difficulty: Easy
// Tags: array, sort
```

```
//
// 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 minimum_cost(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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