Problem 3640: Trionic Array II

Problem Information

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

Problem Description

You are given an integer array

nums

of length

n

.

A

trionic subarray

is a contiguous subarray

nums[l...r]

(with

0 <= l < r < n

) for which there exist indices

l < p < q < r

such that:

nums[l...p]

is

strictly

increasing,

nums[p...q]

is

strictly

decreasing,

nums[q...r]

is

strictly

increasing.

Return the

maximum

sum of any trionic subarray in

nums

.

Example 1:

Input:

nums = [0,-2,-1,-3,0,2,-1]

Output:

-4

Explanation:

Pick

l = 1

,

p = 2

,

q = 3

,

r = 5

:

nums[l...p] = nums[1...2] = [-2, -1]

is strictly increasing (

-2 < -1

).

nums[p...q] = nums[2...3] = [-1, -3]

is strictly decreasing (

-1 > -3

)

nums[q...r] = nums[3...5] = [-3, 0, 2]

is strictly increasing (

-3 < 0 < 2

).

Sum =

(-2) + (-1) + (-3) + 0 + 2 = -4

.

Example 2:

Input:

nums = [1,4,2,7]

Output:

14

Explanation:

Pick

l = 0

,

p = 1

,

q = 2

,

r = 3

:

nums[l...p] = nums[0...1] = [1, 4]

is strictly increasing (

1 < 4

).

nums[p...q] = nums[1...2] = [4, 2]

is strictly decreasing (

4 > 2

).

nums[q...r] = nums[2...3] = [2, 7]

is strictly increasing (

2 < 7

).

Sum =

1 + 4 + 2 + 7 = 14

.

Constraints:

4 <= n = nums.length <= 10

5

-10

9

<= nums[i] <= 10

9

It is guaranteed that at least one trionic subarray exists.

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public long maxSumTrionic(int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public long MaxSumTrionic(int[] nums) {

}
}
```

C:

```
long long maxSumTrionic(int* nums, int numsSize) {

}
```

Go:

```
func maxSumTrionic(nums []int) int64 {
```

```
}
```

Kotlin:

```
class Solution {
fun maxSumTrionic(nums: IntArray): Long {

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn max_sum_trionic(nums: Vec<i32>) -> i64 {

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

```
function maxSumTrionic($nums) {

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (max-sum-trionic nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Trionic Array II
* Difficulty: Hard
* 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:
long long maxSumTrionic(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Trionic Array II
* Difficulty: Hard
* 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 long maxSumTrionic(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Trionic Array II
Difficulty: Hard
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 maxSumTrionic(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Trionic Array II
* Difficulty: Hard
* 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 maxSumTrionic = function(nums) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Trionic Array II
* Difficulty: Hard
* 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 maxSumTrionic(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Trionic Array II
* Difficulty: Hard
* 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 long MaxSumTrionic(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Trionic Array II
* Difficulty: Hard
```

```
* 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
*/

long long maxSumTrionic(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Trionic Array II
// Difficulty: Hard
// 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 maxSumTrionic(nums []int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun maxSumTrionic(nums: IntArray): Long {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Trionic Array II
// Difficulty: Hard
// 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 max_sum_trionic(nums: Vec<i32>) -> i64 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

```
}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (max-sum-trionic nums)
(-> (listof exact-integer?) exact-integer?)
)
```