Problem 1330: Reverse Subarray To Maximize Array Value

Problem Information

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

Problem Description

You are given an integer array

nums

. The

value

of this array is defined as the sum of

|nums[i] - nums[i + 1]|

for all

0 <= i < nums.length - 1

.

You are allowed to select any subarray of the given array and reverse it. You can perform this operation

only once

.

Find maximum possible value of the final array.

Example 1:

Input:

nums = [2,3,1,5,4]

Output:

10

Explanation:

By reversing the subarray [3,1,5] the array becomes [2,5,1,3,4] whose value is 10.

Example 2:

Input:

nums = [2,4,9,24,2,1,10]

Output:

68

Constraints:

2 <= nums.length <= 3 * 10

4

-10

5

<= nums[i] <= 10

5

The answer is guaranteed to fit in a 32-bit integer.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {number[]} nums
* @return {number}
*/
```

```
var maxValueAfterReverse = function(nums) {

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

```
}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (max-value-after-reverse nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Reverse Subarray To Maximize Array Value
* Difficulty: Hard
* Tags: array, greedy, math
*
* 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 maxValueAfterReverse(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Reverse Subarray To Maximize Array Value
* Difficulty: Hard
* Tags: array, greedy, math
*
* 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 maxValueAfterReverse(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Reverse Subarray To Maximize Array Value
Difficulty: Hard
Tags: array, greedy, math

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Reverse Subarray To Maximize Array Value
* Difficulty: Hard
* Tags: array, greedy, math
*
* 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 maxValueAfterReverse = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Reverse Subarray To Maximize Array Value
* Difficulty: Hard
* Tags: array, greedy, math
*
* 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 maxValueAfterReverse(nums: number[]): number {
```

```
};
```

C# Solution:

```
/*
* Problem: Reverse Subarray To Maximize Array Value
* Difficulty: Hard
* Tags: array, greedy, math
*
* 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 MaxValueAfterReverse(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Reverse Subarray To Maximize Array Value
* Difficulty: Hard
* Tags: array, greedy, math
*
* 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 maxValueAfterReverse(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Reverse Subarray To Maximize Array Value
// Difficulty: Hard
```

```
// Tags: array, greedy, math
//
// 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 maxValueAfterReverse(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Reverse Subarray To Maximize Array Value
// Difficulty: Hard
// Tags: array, greedy, math
//
// 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 max_value_after_reverse(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

```
end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (max-value-after-reverse nums)
(-> (listof exact-integer?) exact-integer?)
)
```