Problem 3073: Maximum Increasing Triplet Value

Problem Information

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

Problem Description

Given an array

nums

, return

the

maximum value

of a triplet

(i, j, k)

such that

i < j < k

and

nums[i] < nums[j] < nums[k]

.

The

value

of a triplet

(i, j, k)

is

nums[i] - nums[j] + nums[k]

.

Example 1:

Input:

nums = [5,6,9]

Output:

8

Explanation:

We only have one choice for an increasing triplet and that is choosing all three elements. The value of this triplet would be

5 - 6 + 9 = 8

.

Example 2:

Input:

nums = [1,5,3,6]

Output:

4

Explanation:

There are only two increasing triplets:

(0, 1, 3)

: The value of this triplet is

nums[0] - nums[1] + nums[3] = 1 - 5 + 6 = 2

.

(0, 2, 3)

: The value of this triplet is

nums[0] - nums[2] + nums[3] = 1 - 3 + 6 = 4

.

Thus the answer would be

4

.

Constraints:

3 <= nums.length <= 10

5

1 <= nums[i] <= 10

9

The input is generated such that at least one triplet meets the given condition.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

```
var maximumTripletValue = function(nums) {

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (maximum-triplet-value nums)
(-> (listof exact-integer?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Increasing Triplet Value
* Difficulty: Medium
* Tags: array
*
* 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 maximumTripletValue(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Increasing Triplet Value
* Difficulty: Medium
* Tags: array
*
* 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 maximumTripletValue(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Increasing Triplet Value
Difficulty: Medium
Tags: array

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Increasing Triplet Value
* Difficulty: Medium
* Tags: array
*
* 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 maximumTripletValue = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Increasing Triplet Value
* Difficulty: Medium
* Tags: array
*
* 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 maximumTripletValue(nums: number[]): number {



};
```

C# Solution:

```
/*
* Problem: Maximum Increasing Triplet Value
* Difficulty: Medium
* Tags: array
*
* 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 MaximumTripletValue(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Increasing Triplet Value
* Difficulty: Medium
* Tags: array
*
* 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 maximumTripletValue(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Maximum Increasing Triplet Value
// Difficulty: Medium
// Tags: array
//
// 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 maximumTripletValue(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Increasing Triplet Value
// Difficulty: Medium
// Tags: array
//
// 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 maximum_triplet_value(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

```
end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (maximum-triplet-value nums)
(-> (listof exact-integer?) exact-integer?)
)
```