Problem 2908: Minimum Sum of Mountain Triplets I

Problem Information

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

Problem Description

You are given a

0-indexed

array

nums

of integers.

A triplet of indices

(i, j, k)

is a

mountain

if:

i < j < k

nums[i] < nums[j]

and

nums[k] < nums[j]

Return

the

minimum possible sum

of a mountain triplet of

nums

.

If no such triplet exists, return

-1

.

Example 1:

Input:

nums = [8,6,1,5,3]

Output:

9

Explanation:

Triplet (2, 3, 4) is a mountain triplet of sum 9 since: - 2 < 3 < 4 - nums[2] < nums[3] and nums[4] < nums[3] And the sum of this triplet is nums[2] + nums[3] + nums[4] = 9. It can be shown that there are no mountain triplets with a sum of less than 9.

Example 2:

Input:

nums = [5,4,8,7,10,2]

Output:

13

Explanation:

Triplet (1, 3, 5) is a mountain triplet of sum 13 since: - 1 < 3 < 5 - nums[1] < nums[3] and nums[5] < nums[3] And the sum of this triplet is nums[1] + nums[3] + nums[5] = 13. It can be shown that there are no mountain triplets with a sum of less than 13.

Example 3:

Input:

nums = [6,5,4,3,4,5]

Output:

-1

Explanation:

It can be shown that there are no mountain triplets in nums.

Constraints:

3 <= nums.length <= 50

1 <= nums[i] <= 50

Code Snippets

C++:

```
class Solution {
public:
```

```
int minimumSum(vector<int>& nums) {

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

```
end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Minimum Sum of Mountain Triplets I
* Difficulty: Easy
* 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 minimumSum(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Sum of Mountain Triplets I
```

```
* Difficulty: Easy
* 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 minimumSum(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Sum of Mountain Triplets I
Difficulty: Easy
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 minimumSum(self, nums: List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Sum of Mountain Triplets I
* Difficulty: Easy
* 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 minimumSum = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Sum of Mountain Triplets I
* Difficulty: Easy
* 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 minimumSum(nums: number[]): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Sum of Mountain Triplets I
* Difficulty: Easy
* 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 MinimumSum(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Sum of Mountain Triplets I
* Difficulty: Easy
* 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 minimumSum(int* nums, int numsSize) {

}
```

Go Solution:

```
// Problem: Minimum Sum of Mountain Triplets I
// Difficulty: Easy
// 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 minimumSum(nums []int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Sum of Mountain Triplets I
// Difficulty: Easy
// 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 minimum_sum(nums: Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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