Problem 2574: Left and Right Sum Differences

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

nums

of size

n

.

Define two arrays

leftSum

and

rightSum

where:

leftSum[i]

is the sum of elements to the left of the index

i

in the array

nums

. If there is no such element,

leftSum[i] = 0

.

rightSum[i]

is the sum of elements to the right of the index

i

in the array

nums

. If there is no such element,

rightSum[i] = 0

.

Return an integer array

answer

of size

n

where

answer[i] = |leftSum[i] - rightSum[i]|

.

Example 1:

Input:

nums = [10,4,8,3]

Output:

[15,1,11,22]

Explanation:

The array leftSum is [0,10,14,22] and the array rightSum is [15,11,3,0]. The array answer is [|0 - 15|,|10 - 11|,|14 - 3|,|22 - 0|] = [15,1,11,22].

Example 2:

Input:

nums = [1]

Output:

[0]

Explanation:

The array leftSum is [0] and the array rightSum is [0]. The array answer is [|0 - 0|] = [0].

Constraints:

1 <= nums.length <= 1000

1 <= nums[i] <= 10

5

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

```
var leftRightDifference = function(nums) {

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* leftRightDifference(int* nums, int numsSize, int* returnSize) {

}
```

Go:

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

}
```

Kotlin:

```
class Solution {
fun leftRightDifference(nums: IntArray): IntArray {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (left-right-difference nums)
(-> (listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Left and Right Sum Differences
* 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:
vector<int> leftRightDifference(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Left and Right Sum Differences
* 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[] leftRightDifference(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Left and Right Sum Differences
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 leftRightDifference(self, nums: List[int]) -> List[int]:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Left and Right Sum Differences
* 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 leftRightDifference = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Left and Right Sum Differences
* 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 leftRightDifference(nums: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Left and Right Sum Differences
* 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[] LeftRightDifference(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Left and Right Sum Differences
* 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
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* leftRightDifference(int* nums, int numsSize, int* returnSize) {
```

```
}
```

Go Solution:

```
// Problem: Left and Right Sum Differences
// 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 leftRightDifference(nums []int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun leftRightDifference(nums: IntArray): IntArray {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Left and Right Sum Differences
// 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 left_right_difference(nums: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

```
}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (left-right-difference nums)
(-> (listof exact-integer?) (listof exact-integer?))
)
```