Problem 3379: Transformed Array

Problem Information

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

Problem Description

You are given an integer array

nums

that represents a circular array. Your task is to create a new array

result

of the

same

size, following these rules:

For each index

i

(where

0 <= i < nums.length

), perform the following

independent

actions:

If

nums[i] > 0

: Start at index

i

and move

nums[i]

steps to the

right

in the circular array. Set

result[i]

to the value of the index where you land.

If

nums[i] < 0

: Start at index

i

and move

abs(nums[i])

steps to the

left

in the circular array. Set

result[i]

to the value of the index where you land.

If

nums[i] == 0

: Set

result[i]

to

nums[i]

.

Return the new array

result

.

Note:

Since

nums

is circular, moving past the last element wraps around to the beginning, and moving before the first element wraps back to the end.

Example 1:

Input:

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

Output:

[1,1,1,3]

Explanation:

For

nums[0]

that is equal to 3, If we move 3 steps to right, we reach

nums[3]

. So

result[0]

should be 1.

For

nums[1]

that is equal to -2, If we move 2 steps to left, we reach

nums[3]

. So

result[1]

should be 1.

For

nums[2]

that is equal to 1, If we move 1 step to right, we reach

nums[3]

. So

result[2]

should be 1.

For

nums[3]

that is equal to 1, If we move 1 step to right, we reach

nums[0]

. So

result[3]

should be 3.

Example 2:

Input:

nums = [-1,4,-1]

Output:

[-1,-1,4]

Explanation:

For

nums[0]

that is equal to -1, If we move 1 step to left, we reach

nums[2]

. So

result[0]

should be -1.

For

nums[1]

that is equal to 4, If we move 4 steps to right, we reach

nums[2]

. So

result[1]

should be -1.

For

nums[2]

that is equal to -1, If we move 1 step to left, we reach

nums[1]

. So

result[2]

should be 4.

Constraints:

1 <= nums.length <= 100

-100 <= nums[i] <= 100

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

```
class Solution {
fun constructTransformedArray(nums: IntArray): IntArray {
}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (construct-transformed-array nums)
(-> (listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Transformed Array
* 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> constructTransformedArray(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Transformed Array
* 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[] constructTransformedArray(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Transformed Array
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 constructTransformedArray(self, nums: List[int]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Transformed Array
* 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 constructTransformedArray = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Transformed Array
* 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 constructTransformedArray(nums: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Transformed Array
* 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[] ConstructTransformedArray(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Transformed Array
* 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* constructTransformedArray(int* nums, int numsSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Transformed Array
// 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 constructTransformedArray(nums []int) []int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Transformed Array
// 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 construct_transformed_array(nums: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```racket
(define/contract (construct-transformed-array nums)
(-> (listof exact-integer?) (listof exact-integer?))
)
```