Problem 3467: Transform Array by Parity

Problem Information

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

Problem Description

You are given an integer array

nums

. Transform

nums

by performing the following operations in the

exact

order specified:

Replace each even number with 0.

Replace each odd numbers with 1.

Sort the modified array in

non-decreasing

order.

Return the resulting array after performing these operations.

Example 1:

Input:

nums = [4,3,2,1]

Output:

[0,0,1,1]

Explanation:

Replace the even numbers (4 and 2) with 0 and the odd numbers (3 and 1) with 1. Now,

nums = [0, 1, 0, 1]

.

After sorting

nums

in non-descending order,

nums = [0, 0, 1, 1]

.

Example 2:

Input:

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

Output:

[0,0,1,1,1]

Explanation:

Replace the even numbers (4 and 2) with 0 and the odd numbers (1, 5 and 1) with 1. Now,

nums = [1, 1, 1, 0, 0]

.

After sorting

nums

in non-descending order,

nums = [0, 0, 1, 1, 1]

.

Constraints:

1 <= nums.length <= 100

1 <= nums[i] <= 1000

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

```
end
```

PHP:

```php
class Solution {

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

}
}
```

Dart:

```dart
class Solution {
List<int> transformArray(List<int> nums) {

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Transform Array by Parity
* Difficulty: Easy
* Tags: array, sort
*
* 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> transformArray(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Transform Array by Parity
* Difficulty: Easy
* Tags: array, sort
*
* 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[] transformArray(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Transform Array by Parity
Difficulty: Easy
Tags: array, sort

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Transform Array by Parity
* Difficulty: Easy
* Tags: array, sort
*
```

```
* 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 transformArray = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Transform Array by Parity
* Difficulty: Easy
* Tags: array, sort
*
* 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 transformArray(nums: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Transform Array by Parity
* Difficulty: Easy
* Tags: array, sort
*
* 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[] TransformArray(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Transform Array by Parity
* Difficulty: Easy
* Tags: array, sort
*
* 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* transformArray(int* nums, int numsSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Transform Array by Parity
// Difficulty: Easy
// Tags: array, sort
//
// 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 transformArray(nums []int) []int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Transform Array by Parity
// Difficulty: Easy
// Tags: array, sort
//
// 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 transform_array(nums: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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