Problem 922: Sort Array By Parity II

Problem Information

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

Problem Description

Given an array of integers

nums

, half of the integers in

nums

are

odd

, and the other half are

even

.

Sort the array so that whenever

nums[i]

is odd,

i

is

odd

, and whenever

nums[i]

is even,

i

is

even

.

Return

any answer array that satisfies this condition

.

Example 1:

Input:

nums = [4,2,5,7]

Output:

[4,5,2,7]

Explanation:

[4,7,2,5], [2,5,4,7], [2,7,4,5] would also have been accepted.

Example 2:

Input:

nums = [2,3]

Output:

[2,3]

Constraints:

2 <= nums.length <= 2 * 10

4

nums.length

is even.

Half of the integers in

nums

are even.

0 <= nums[i] <= 1000

Follow Up:

Could you solve it in-place?

Code Snippets

C++:

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

```
}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
```

```
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (sort-array-by-parity-ii nums)
(-> (listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Sort Array By Parity II
* 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> sortArrayByParityII(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Sort Array By Parity II
* 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[] sortArrayByParityII(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Sort Array By Parity II
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 sortArrayByParityII(self, nums: List[int]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Sort Array By Parity II
* 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 sortArrayByParityII = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Sort Array By Parity II
* 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 sortArrayByParityII(nums: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Sort Array By Parity II
* 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[] SortArrayByParityII(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Sort Array By Parity II
* 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* sortArrayByParityII(int* nums, int numsSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Sort Array By Parity II
// 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 sortArrayByParityII(nums []int) []int {
```

```
}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Sort Array By Parity II
// 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 sort_array_by_parity_ii(nums: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (sort-array-by-parity-ii nums)
(-> (listof exact-integer?) (listof exact-integer?))
)
```