Problem 2164: Sort Even and Odd Indices Independently

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

nums

. Rearrange the values of

nums

according to the following rules:

Sort the values at

odd indices

of

nums

in

non-increasing

order.

For example, if

nums = [4,

1

,2,

3

]

before this step, it becomes

[4,

3

,2,

1

]

after. The values at odd indices

1

and

3

are sorted in non-increasing order.

Sort the values at

even indices

of

nums

in

non-decreasing

order.

For example, if

nums = [

4

,1,

2

,3]

before this step, it becomes

[

2

,1,

4

,3]

after. The values at even indices

0

and

2

are sorted in non-decreasing order.

Return

the array formed after rearranging the values of

nums

.

Example 1:

Input:

nums = [4,1,2,3]

Output:

[2,3,4,1]

Explanation:

First, we sort the values present at odd indices (1 and 3) in non-increasing order. So, nums changes from [4,

1

,2,

3

] to [4,

3

,2,

1

]. Next, we sort the values present at even indices (0 and 2) in non-decreasing order. So, nums changes from [

4

,1,

2

,3] to [

2

,3,

4

,1]. Thus, the array formed after rearranging the values is [2,3,4,1].

Example 2:

Input:

nums = [2,1]

Output:

[2,1]

Explanation:

Since there is exactly one odd index and one even index, no rearrangement of values takes place. The resultant array formed is [2,1], which is the same as the initial array.

Constraints:

1 <= nums.length <= 100

1 <= nums[i] <= 100

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
```

```
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (sort-even-odd nums)
(-> (listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Sort Even and Odd Indices Independently
* 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> sortEvenOdd(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Sort Even and Odd Indices Independently
* 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[] sortEvenOdd(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Sort Even and Odd Indices Independently
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 sortEvenOdd(self, nums: List[int]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Sort Even and Odd Indices Independently
* 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 sortEvenOdd = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Sort Even and Odd Indices Independently
* 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 sortEvenOdd(nums: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Sort Even and Odd Indices Independently
* 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[] SortEvenOdd(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Sort Even and Odd Indices Independently
* 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* sortEvenOdd(int* nums, int numsSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Sort Even and Odd Indices Independently
// 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 sortEvenOdd(nums []int) []int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Sort Even and Odd Indices Independently
// 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_even_odd(nums: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (sort-even-odd nums)
(-> (listof exact-integer?) (listof exact-integer?))
)
```