Problem 3667: Sort Array By Absolute Value

Problem Information

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

Problem Description

You are given an integer array

nums

.

Rearrange elements of

nums

in

non-decreasing

order of their absolute value.

Return

any

rearranged array that satisfies this condition.

Note

: The absolute value of an integer x is defined as:

x

if

x >= 0

-x

if

x < 0

Example 1:

Input:

nums = [3,-1,-4,1,5]

Output:

[-1,1,3,-4,5]

Explanation:

The absolute values of elements in

nums

are 3, 1, 4, 1, 5 respectively.

Rearranging them in increasing order, we get 1, 1, 3, 4, 5.

This corresponds to

[-1, 1, 3, -4, 5]

. Another possible rearrangement is

[1, -1, 3, -4, 5].

Example 2:

Input:

nums = [-100,100]

Output:

[-100,100]

Explanation:

The absolute values of elements in

nums

are 100, 100 respectively.

Rearranging them in increasing order, we get 100, 100.

This corresponds to

[-100, 100]

. Another possible rearrangement is

[100, -100]

.

Constraints:

1 <= nums.length <= 100

-100 <= nums[i] <= 100

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
```

```
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (sort-by-absolute-value nums)
(-> (listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Sort Array By Absolute Value
* Difficulty: Easy
* Tags: array, math, 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> sortByAbsoluteValue(vector<int>& nums) {
```

```
}
};
```

Java Solution:

```
/**
* Problem: Sort Array By Absolute Value
* Difficulty: Easy
* Tags: array, math, 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[] sortByAbsoluteValue(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Sort Array By Absolute Value
Difficulty: Easy
Tags: array, math, 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 sortByAbsoluteValue(self, nums: List[int]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Sort Array By Absolute Value
* Difficulty: Easy
* Tags: array, math, 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 sortByAbsoluteValue = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Sort Array By Absolute Value
* Difficulty: Easy
* Tags: array, math, 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 sortByAbsoluteValue(nums: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Sort Array By Absolute Value
* Difficulty: Easy
* Tags: array, math, 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[] SortByAbsoluteValue(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Sort Array By Absolute Value
* Difficulty: Easy
* Tags: array, math, 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* sortByAbsoluteValue(int* nums, int numsSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Sort Array By Absolute Value
// Difficulty: Easy
// Tags: array, math, 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 sortByAbsoluteValue(nums []int) []int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Sort Array By Absolute Value
// Difficulty: Easy
// Tags: array, math, 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_by_absolute_value(nums: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

```
end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (sort-by-absolute-value nums)
(-> (listof exact-integer?) (listof exact-integer?))
)
```