Problem 912: Sort an Array

Problem Information

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

Problem Description

Given an array of integers

nums

, sort the array in ascending order and return it.

You must solve the problem

without using any built-in

functions in

O(nlog(n))

time complexity and with the smallest space complexity possible.

Example 1:

Input:

nums = [5,2,3,1]

Output:

[1,2,3,5]

Explanation:

After sorting the array, the positions of some numbers are not changed (for example, 2 and 3), while the positions of other numbers are changed (for example, 1 and 5).

Example 2:

Input:

nums = [5,1,1,2,0,0]

Output:

[0,0,1,1,2,5]

Explanation:

Note that the values of nums are not necessarily unique.

Constraints:

1 <= nums.length <= 5 * 10

4

-5 * 10

4

<= nums[i] <= 5 * 10

4

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

```
};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Sort an Array
* Difficulty: Medium
* Tags: array, sort, queue, heap
*
* 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> sortArray(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Sort an Array
* Difficulty: Medium
* Tags: array, sort, queue, heap
*
* 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[] sortArray(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Sort an Array
Difficulty: Medium
Tags: array, sort, queue, heap

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

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Sort an Array
* Difficulty: Medium
* Tags: array, sort, queue, heap
*
* 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 sortArray = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Sort an Array
* Difficulty: Medium
* Tags: array, sort, queue, heap
*
* 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 sortArray(nums: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Sort an Array
* Difficulty: Medium
* Tags: array, sort, queue, heap
*
* 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[] SortArray(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Sort an Array
* Difficulty: Medium
* Tags: array, sort, queue, heap
*
* 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* sortArray(int* nums, int numsSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Sort an Array
// Difficulty: Medium
// Tags: array, sort, queue, heap
//
// 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 sortArray(nums []int) []int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Sort an Array
// Difficulty: Medium
// Tags: array, sort, queue, heap
//
// 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(nums: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
```

```
end
```

Erlang Solution:

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

Racket Solution:

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