Problem 2161: Partition Array According to Given Pivot

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

nums

and an integer

pivot

. Rearrange

nums

such that the following conditions are satisfied:

Every element less than

pivot

appears

before

every element greater than

pivot

.

Every element equal to

pivot

appears

in between

the elements less than and greater than

pivot

.

The

relative order

of the elements less than

pivot

and the elements greater than

pivot

is maintained.

More formally, consider every

p

i

,

p

j

where

p

i

is the new position of the

i

th

element and

p

j

is the new position of the

j

th

element. If

i < j

and

both

elements are smaller (

or larger

) than

pivot

, then

p

i

< p

j

.

Return

nums

after the rearrangement.

Example 1:

Input:

nums = [9,12,5,10,14,3,10], pivot = 10

Output:

[9,5,3,10,10,12,14]

Explanation:

The elements 9, 5, and 3 are less than the pivot so they are on the left side of the array. The elements 12 and 14 are greater than the pivot so they are on the right side of the array. The relative ordering of the elements less than and greater than pivot is also maintained. [9, 5, 3] and [12, 14] are the respective orderings.

Example 2:

Input:

nums = [-3,4,3,2], pivot = 2

Output:

[-3,2,4,3]

Explanation:

The element -3 is less than the pivot so it is on the left side of the array. The elements 4 and 3 are greater than the pivot so they are on the right side of the array. The relative ordering of the elements less than and greater than pivot is also maintained. [-3] and [4, 3] are the respective orderings.

Constraints:

1 <= nums.length <= 10

5

-10

6

<= nums[i] <= 10

6

pivot

equals to an element of

nums

.

Code Snippets

C++:

```
class Solution {
public:
vector<int> pivotArray(vector<int>& nums, int pivot) {

}
};
```

Java:

```
class Solution {
public int[] pivotArray(int[] nums, int pivot) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function pivotArray(nums: number[], pivot: number): number[] {

};
```

C#:

```
public class Solution {
public int[] PivotArray(int[] nums, int pivot) {

}
}
```

C:

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

}
```

Go:

```
func pivotArray(nums []int, pivot int) []int {

}
```

Kotlin:

```
class Solution {
fun pivotArray(nums: IntArray, pivot: Int): IntArray {
```

```
}
}
```

Swift:

```
class Solution {
func pivotArray(_ nums: [Int], _ pivot: Int) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn pivot_array(nums: Vec<i32>, pivot: i32) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} pivot
# @return {Integer[]}
def pivot_array(nums, pivot)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $pivot
* @return Integer[]
*/
function pivotArray($nums, $pivot) {

}
}
```

Dart:

```
class Solution {
List<int> pivotArray(List<int> nums, int pivot) {

}
}
```

Scala:

```
object Solution {
def pivotArray(nums: Array[Int], pivot: Int): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec pivot_array(nums :: [integer], pivot :: integer) :: [integer]
def pivot_array(nums, pivot) do

end
end
```

Erlang:

```
-spec pivot_array(Nums :: [integer()], Pivot :: integer()) -> [integer()].
pivot_array(Nums, Pivot) ->
.
```

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Partition Array According to Given Pivot
* Difficulty: Medium
* Tags: array
*
* 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> pivotArray(vector<int>& nums, int pivot) {

}
};
```

Java Solution:

```
/**
* Problem: Partition Array According to Given Pivot
* Difficulty: Medium
* Tags: array
*
* 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[] pivotArray(int[] nums, int pivot) {

}
}
```

Python3 Solution:

```
"""
Problem: Partition Array According to Given Pivot
Difficulty: Medium
Tags: array
```

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Partition Array According to Given Pivot
* Difficulty: Medium
* Tags: array
*
* 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
* @param {number} pivot
* @return {number[]}
*/
var pivotArray = function(nums, pivot) {

};
```

TypeScript Solution:

```
/**
* Problem: Partition Array According to Given Pivot
* Difficulty: Medium
* Tags: array
*
* 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 pivotArray(nums: number[], pivot: number): number[] {

};
```

C# Solution:

```
/*
* Problem: Partition Array According to Given Pivot
* Difficulty: Medium
* Tags: array
*
* 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[] PivotArray(int[] nums, int pivot) {

}
}
```

C Solution:

```
/*
* Problem: Partition Array According to Given Pivot
* Difficulty: Medium
* Tags: array
*
* 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* pivotArray(int* nums, int numsSize, int pivot, int* returnSize) {

}
```

Go Solution:

```
// Problem: Partition Array According to Given Pivot
// Difficulty: Medium
// Tags: array
//
// 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 pivotArray(nums []int, pivot int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun pivotArray(nums: IntArray, pivot: Int): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func pivotArray(_ nums: [Int], _ pivot: Int) -> [Int] {

}
}
```

Rust Solution:

```
// Problem: Partition Array According to Given Pivot
// Difficulty: Medium
// Tags: array
//
// 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 pivot_array(nums: Vec<i32>, pivot: i32) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} pivot
# @return {Integer[]}
def pivot_array(nums, pivot)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $pivot
* @return Integer[]
*/
function pivotArray($nums, $pivot) {

}
}
```

Dart Solution:

```
class Solution {
List<int> pivotArray(List<int> nums, int pivot) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def pivotArray(nums: Array[Int], pivot: Int): Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec pivot_array(nums :: [integer], pivot :: integer) :: [integer]
def pivot_array(nums, pivot) do

end
end
```

Erlang Solution:

```
-spec pivot_array(Nums :: [integer()], Pivot :: integer()) -> [integer()].
pivot_array(Nums, Pivot) ->
.
```

Racket Solution:

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