Problem 1331: Rank Transform of an Array

Problem Information

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

Problem Description

Given an array of integers

arr

, replace each element with its rank.

The rank represents how large the element is. The rank has the following rules:

Rank is an integer starting from 1.

The larger the element, the larger the rank. If two elements are equal, their rank must be the same.

Rank should be as small as possible.

Example 1:

Input:

arr = [40,10,20,30]

Output:

[4,1,2,3]

Explanation

: 40 is the largest element. 10 is the smallest. 20 is the second smallest. 30 is the third smallest.

Example 2:

Input:

arr = [100,100,100]

Output:

[1,1,1]

Explanation

: Same elements share the same rank.

Example 3:

Input:

arr = [37,12,28,9,100,56,80,5,12]

Output:

[5,3,4,2,8,6,7,1,3]

Constraints:

0 <= arr.length <= 10

5

-10

9

<= arr[i] <= 10

9

Code Snippets

C++:

```
class Solution {
public:
vector<int> arrayRankTransform(vector<int>& arr) {

}
};
```

Java:

```
class Solution {
public int[] arrayRankTransform(int[] arr) {

}
}
```

Python3:

```
class Solution:
def arrayRankTransform(self, arr: List[int]) -> List[int]:
```

Python:

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

JavaScript:

```
/**
* @param {number[]} arr
* @return {number[]}
*/
```

```
var arrayRankTransform = function(arr) {

};
```

TypeScript:

```
function arrayRankTransform(arr: number[]): number[] {

};
```

C#:

```
public class Solution {
public int[] ArrayRankTransform(int[] arr) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* arrayRankTransform(int* arr, int arrSize, int* returnSize) {

}
```

Go:

```
func arrayRankTransform(arr []int) []int {

}
```

Kotlin:

```
class Solution {
fun arrayRankTransform(arr: IntArray): IntArray {

}
}
```

Swift:

```
class Solution {
func arrayRankTransform(_ arr: [Int]) -> [Int] {

}
}
```

Rust:

```
impl Solution {
pub fn array_rank_transform(arr: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[]} arr
# @return {Integer[]}
def array_rank_transform(arr)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $arr
* @return Integer[]
*/
function arrayRankTransform($arr) {

}
}
```

Dart:

```
class Solution {
List<int> arrayRankTransform(List<int> arr) {

}
}
```

Scala:

```
object Solution {
def arrayRankTransform(arr: Array[Int]): Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec array_rank_transform(arr :: [integer]) :: [integer]
def array_rank_transform(arr) do

end
end
```

Erlang:

```
-spec array_rank_transform(Arr :: [integer()]) -> [integer()].
array_rank_transform(Arr) ->
.
```

Racket:

```
(define/contract (array-rank-transform arr)
(-> (listof exact-integer?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Rank Transform of an Array
* Difficulty: Easy
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/
```

```
class Solution {
public:
vector<int> arrayRankTransform(vector<int>& arr) {

}
};
```

Java Solution:

```
/**
* Problem: Rank Transform of an Array
* Difficulty: Easy
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int[] arrayRankTransform(int[] arr) {

}
}
```

Python3 Solution:

```
"""
Problem: Rank Transform of an Array
Difficulty: Easy
Tags: array, hash, sort

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def arrayRankTransform(self, arr: List[int]) -> List[int]:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Rank Transform of an Array
* Difficulty: Easy
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* @param {number[]} arr
* @return {number[]}
*/
var arrayRankTransform = function(arr) {

};
```

TypeScript Solution:

```
/**
* Problem: Rank Transform of an Array
* Difficulty: Easy
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
```

```
*/

function arrayRankTransform(arr: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Rank Transform of an Array
* Difficulty: Easy
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int[] ArrayRankTransform(int[] arr) {

}
}
```

C Solution:

```
/*
* Problem: Rank Transform of an Array
* Difficulty: Easy
* Tags: array, hash, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* arrayRankTransform(int* arr, int arrSize, int* returnSize) {

}
```

Go Solution:

```go
// Problem: Rank Transform of an Array
// Difficulty: Easy
// Tags: array, hash, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func arrayRankTransform(arr []int) []int {

}
```

Kotlin Solution:

```kotlin
class Solution {
fun arrayRankTransform(arr: IntArray): IntArray {

}
}
```

Swift Solution:

```swift
class Solution {
func arrayRankTransform(_ arr: [Int]) -> [Int] {

}
}
```

Rust Solution:

```rust
// Problem: Rank Transform of an Array
// Difficulty: Easy
// Tags: array, hash, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map
```

```rust
impl Solution {
pub fn array_rank_transform(arr: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

```ruby
# @param {Integer[]} arr
# @return {Integer[]}
def array_rank_transform(arr)

end
```

PHP Solution:

```php
class Solution {

/**
* @param Integer[] $arr
* @return Integer[]
*/
function arrayRankTransform($arr) {

}
}
```

Dart Solution:

```dart
class Solution {
List<int> arrayRankTransform(List<int> arr) {

}
}
```

Scala Solution:

```scala
object Solution {
def arrayRankTransform(arr: Array[Int]): Array[Int] = {
}
}
```

Elixir Solution:

```
defmodule Solution do
@spec array_rank_transform(arr :: [integer]) :: [integer]
def array_rank_transform(arr) do

end
end
```

Erlang Solution:

```
-spec array_rank_transform(Arr :: [integer()]) -> [integer()].
array_rank_transform(Arr) ->
.
```

Racket Solution:

```
(define/contract (array-rank-transform arr)
(-> (listof exact-integer?) (listof exact-integer?))
)
```