Problem 1243: Array Transformation

Problem Information

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

Problem Description

Given an initial array

arr

, every day you produce a new array using the array of the previous day.

On the

i

-th day, you do the following operations on the array of day

i-1

to produce the array of day

i

:

If an element is smaller than both its left neighbor and its right neighbor, then this element is incremented.

If an element is bigger than both its left neighbor and its right neighbor, then this element is decremented.

The first and last elements never change.

After some days, the array does not change. Return that final array.

Example 1:

Input:

arr = [6,2,3,4]

Output:

[6,3,3,4]

Explanation:

On the first day, the array is changed from [6,2,3,4] to [6,3,3,4]. No more operations can be done to this array.

Example 2:

Input:

arr = [1,6,3,4,3,5]

Output:

[1,4,4,4,4,5]

Explanation:

On the first day, the array is changed from [1,6,3,4,3,5] to [1,5,4,3,4,5]. On the second day, the array is changed from [1,5,4,3,4,5] to [1,4,4,4,4,5]. No more operations can be done to this array.

Constraints:

3 <= arr.length <= 100

1 <= arr[i] <= 100

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public List<Integer> transformArray(int[] arr) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

```
var transformArray = function(arr) {

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public IList<int> TransformArray(int[] arr) {

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

```
class Solution {
fun transformArray(arr: IntArray): List<Int> {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Scala:

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

}
}
```

Solutions

C++ Solution:

```
/*
* Problem: Array Transformation
* Difficulty: Easy
* 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> transformArray(vector<int>& arr) {

}
};
```

Java Solution:

```
/**
* Problem: Array Transformation
* Difficulty: Easy
* 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 List<Integer> transformArray(int[] arr) {

}
}
```

Python3 Solution:

```
"""
Problem: Array Transformation
Difficulty: Easy
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 transformArray(self, arr: List[int]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Array Transformation
* Difficulty: Easy
* 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[]} arr
* @return {number[]}
*/
```

```
var transformArray = function(arr) {

};
```

TypeScript Solution:

```
/**
* Problem: Array Transformation
* Difficulty: Easy
* 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 transformArray(arr: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Array Transformation
* Difficulty: Easy
* 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 IList<int> TransformArray(int[] arr) {

}
}
```

C Solution:

```
/*
* Problem: Array Transformation
* Difficulty: Easy
* 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* transformArray(int* arr, int arrSize, int* returnSize){

}
```

Go Solution:

```
// Problem: Array Transformation
// Difficulty: Easy
// 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 transformArray(arr []int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun transformArray(arr: IntArray): List<Int> {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Array Transformation
// Difficulty: Easy
// 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 transform_array(arr: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Scala Solution:

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

}
}
```