Problem 1968: Array With Elements Not Equal to Average of Neighbors

Problem Information

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

Problem Description

You are given a

0-indexed

array

nums

of

distinct

integers. You want to rearrange the elements in the array such that every element in the rearranged array is

not

equal to the

average

of its neighbors.

More formally, the rearranged array should have the property such that for every

i

in the range

1 <= i < nums.length - 1

,

(nums[i-1] + nums[i+1]) / 2

is

not

equal to

nums[i]

.

Return

any

rearrangement of

nums

that meets the requirements

.

Example 1:

Input:

nums = [1,2,3,4,5]

Output:

[1,2,4,5,3]

Explanation:

When i=1, nums[i] = 2, and the average of its neighbors is (1+4) / 2 = 2.5. When i=2, nums[i] = 4, and the average of its neighbors is (2+5) / 2 = 3.5. When i=3, nums[i] = 5, and the average of its neighbors is (4+3) / 2 = 3.5.

Example 2:

Input:

nums = [6,2,0,9,7]

Output:

[9,7,6,2,0]

Explanation:

When i=1, nums[i] = 7, and the average of its neighbors is (9+6) / 2 = 7.5. When i=2, nums[i] = 6, and the average of its neighbors is (7+2) / 2 = 4.5. When i=3, nums[i] = 2, and the average of its neighbors is (6+0) / 2 = 3. Note that the original array [6,2,0,9,7] also satisfies the conditions.

Constraints:

3 <= nums.length <= 10

5

0 <= nums[i] <= 10

5

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

```
};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Array With Elements Not Equal to Average of Neighbors
* Difficulty: Medium
* Tags: array, greedy, 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> rearrangeArray(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Array With Elements Not Equal to Average of Neighbors
* Difficulty: Medium
* Tags: array, greedy, 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[] rearrangeArray(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Array With Elements Not Equal to Average of Neighbors
Difficulty: Medium
Tags: array, greedy, 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 rearrangeArray(self, nums: List[int]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Array With Elements Not Equal to Average of Neighbors
* Difficulty: Medium
* Tags: array, greedy, 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 rearrangeArray = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Array With Elements Not Equal to Average of Neighbors
* Difficulty: Medium
* Tags: array, greedy, 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 rearrangeArray(nums: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Array With Elements Not Equal to Average of Neighbors
* Difficulty: Medium
* Tags: array, greedy, 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[] RearrangeArray(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Array With Elements Not Equal to Average of Neighbors
* Difficulty: Medium
* Tags: array, greedy, 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* rearrangeArray(int* nums, int numsSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Array With Elements Not Equal to Average of Neighbors
// Difficulty: Medium
// Tags: array, greedy, 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 rearrangeArray(nums []int) []int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Array With Elements Not Equal to Average of Neighbors
// Difficulty: Medium
// Tags: array, greedy, 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 rearrange_array(nums: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
```

```
end
```

Erlang Solution:

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

Racket Solution:

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