Problem 2615: Sum of Distances

Problem Information

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

Problem Description

You are given a

0-indexed

integer array

nums

. There exists an array

arr

of length

nums.length

, where

arr[i]

is the sum of

|i - j|

over all

j

such that

nums[j] == nums[i]

and

j != i

. If there is no such

j

, set

arr[i]

to be

0

.

Return

the array

arr

.

Example 1:

Input:

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

Output:

[5,0,3,4,0]

Explanation:

When i = 0, nums[0] == nums[2] and nums[0] == nums[3]. Therefore, arr[0] = |0 - 2| + |0 - 3| = 5. When i = 1, arr[1] = 0 because there is no other index with value 3. When i = 2, nums[2] == nums[0] and nums[2] == nums[3]. Therefore, arr[2] = |2 - 0| + |2 - 3| = 3. When i = 3, nums[3] == nums[0] and nums[3] == nums[2]. Therefore, arr[3] = |3 - 0| + |3 - 2| = 4. When i = 4, arr[4] = 0 because there is no other index with value 2.

Example 2:

Input:

nums = [0,5,3]

Output:

[0,0,0]

Explanation:

Since each element in nums is distinct, arr[i] = 0 for all i.

Constraints:

1 <= nums.length <= 10

5

0 <= nums[i] <= 10

9

Note:

This question is the same as

2121: Intervals Between Identical Elements.

Code Snippets

C++:

```cpp
class Solution {
public:
vector<long long> distance(vector<int>& nums) {

}
};
```

Java:

```java
class Solution {
public long[] distance(int[] nums) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public long[] Distance(int[] nums) {

}
}
```

C:

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

}
```

Go:

```
func distance(nums []int) []int64 {

}
```

Kotlin:

```
class Solution {
fun distance(nums: IntArray): LongArray {

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn distance(nums: Vec<i32>) -> Vec<i64> {

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Sum of Distances
* Difficulty: Medium
* Tags: array, hash
*
* 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<long long> distance(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Sum of Distances
* Difficulty: Medium
* Tags: array, hash
*
* 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 long[] distance(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Sum of Distances
Difficulty: Medium
Tags: array, hash

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

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Sum of Distances
* Difficulty: Medium
* Tags: array, hash
*
* 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[]} nums
* @return {number[]}
*/
var distance = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Sum of Distances
* Difficulty: Medium
* Tags: array, hash
*
* 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 distance(nums: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Sum of Distances
* Difficulty: Medium
* Tags: array, hash
*
* 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 long[] Distance(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Sum of Distances
* Difficulty: Medium
* Tags: array, hash
*
* 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().
*/
long long* distance(int* nums, int numsSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Sum of Distances
// Difficulty: Medium
// Tags: array, hash
//
// 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 distance(nums []int) []int64 {

}
```

Kotlin Solution:

```
class Solution {
fun distance(nums: IntArray): LongArray {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Sum of Distances
// Difficulty: Medium
// Tags: array, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map
```

```
impl Solution {
pub fn distance(nums: Vec<i32>) -> Vec<i64> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

```
object Solution {
def distance(nums: Array[Int]): Array[Long] = {
}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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