Problem 360: Sort Transformed Array

Problem Information

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

Problem Description

Given a

sorted

integer array

nums

and three integers

a

,

b

and

c

, apply a quadratic function of the form

f(x) = ax

2

+ bx + c

to each element

nums[i]

in the array, and return

the array in a sorted order

.

Example 1:

Input:

nums = [-4,-2,2,4], a = 1, b = 3, c = 5

Output:

[3,9,15,33]

Example 2:

Input:

nums = [-4,-2,2,4], a = -1, b = 3, c = 5

Output:

[-23,-5,1,7]

Constraints:

1 <= nums.length <= 200

-100 <= nums[i], a, b, c <= 100

nums

is sorted in

ascending

order.

Follow up:

Could you solve it in

O(n)

time?

Code Snippets

C++:

```
class Solution {
public:
vector<int> sortTransformedArray(vector<int>& nums, int a, int b, int c) {

}
};
```

Java:

```
class Solution {
public int[] sortTransformedArray(int[] nums, int a, int b, int c) {

}
}
```

Python3:

```
class Solution:
def sortTransformedArray(self, nums: List[int], a: int, b: int, c: int) ->
List[int]:
```

Python:

```
class Solution(object):
def sortTransformedArray(self, nums, a, b, c):
"""
:type nums: List[int]
:type a: int
:type b: int
:type c: int
:rtype: List[int]
"""
```

JavaScript:

```
/**
* @param {number[]} nums
* @param {number} a
* @param {number} b
* @param {number} c
* @return {number[]}
*/
var sortTransformedArray = function(nums, a, b, c) {

};
```

TypeScript:

```
function sortTransformedArray(nums: number[], a: number, b: number, c:
number): number[] {

};
```

C#:

```
public class Solution {
public int[] SortTransformedArray(int[] nums, int a, int b, int c) {

}
}
```

C:

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

}
```

Go:

```
func sortTransformedArray(nums []int, a int, b int, c int) []int {

}
```

Kotlin:

```
class Solution {
fun sortTransformedArray(nums: IntArray, a: Int, b: Int, c: Int): IntArray {

}
}
```

Swift:

```
class Solution {
func sortTransformedArray(_ nums: [Int], _ a: Int, _ b: Int, _ c: Int) ->
[Int] {

}
}
```

Rust:

```
impl Solution {
pub fn sort_transformed_array(nums: Vec<i32>, a: i32, b: i32, c: i32) ->
Vec<i32> {

}
}
```

Ruby:

```
# @param {Integer[]} nums
# @param {Integer} a
# @param {Integer} b
# @param {Integer} c
# @return {Integer[]}
def sort_transformed_array(nums, a, b, c)

end
```

PHP:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $a
* @param Integer $b
* @param Integer $c
* @return Integer[]
*/
function sortTransformedArray($nums, $a, $b, $c) {

}
}
```

Dart:

```
class Solution {
List<int> sortTransformedArray(List<int> nums, int a, int b, int c) {

}
}
```

Scala:

```
object Solution {
def sortTransformedArray(nums: Array[Int], a: Int, b: Int, c: Int):
Array[Int] = {

}
}
```

Elixir:

```
defmodule Solution do
@spec sort_transformed_array(nums :: [integer], a :: integer, b :: integer, c
:: integer) :: [integer]
def sort_transformed_array(nums, a, b, c) do

end
end
```

Erlang:

```
-spec sort_transformed_array(Nums :: [integer()], A :: integer(), B ::
integer(), C :: integer()) -> [integer()].
sort_transformed_array(Nums, A, B, C) ->
.
```

Racket:

```
(define/contract (sort-transformed-array nums a b c)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?
(listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Sort Transformed Array
* Difficulty: Medium
* Tags: array, math, 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> sortTransformedArray(vector<int>& nums, int a, int b, int c) {

}
```

```
};
```

Java Solution:

```
/**
* Problem: Sort Transformed Array
* Difficulty: Medium
* Tags: array, math, 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[] sortTransformedArray(int[] nums, int a, int b, int c) {

}
}
```

Python3 Solution:

```
"""
Problem: Sort Transformed Array
Difficulty: Medium
Tags: array, math, 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 sortTransformedArray(self, nums: List[int], a: int, b: int, c: int) ->
List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def sortTransformedArray(self, nums, a, b, c):
"""
:type nums: List[int]
:type a: int
:type b: int
:type c: int
:rtype: List[int]
"""
```

JavaScript Solution:

```
/**
* Problem: Sort Transformed Array
* Difficulty: Medium
* Tags: array, math, 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
* @param {number} a
* @param {number} b
* @param {number} c
* @return {number[]}
*/
var sortTransformedArray = function(nums, a, b, c) {

};
```

TypeScript Solution:

```
/**
* Problem: Sort Transformed Array
* Difficulty: Medium
* Tags: array, math, 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 sortTransformedArray(nums: number[], a: number, b: number, c:
number): number[] {

};
```

C# Solution:

```
/*
* Problem: Sort Transformed Array
* Difficulty: Medium
* Tags: array, math, 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[] SortTransformedArray(int[] nums, int a, int b, int c) {

}
}
```

C Solution:

```
/*
* Problem: Sort Transformed Array
* Difficulty: Medium
* Tags: array, math, 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* sortTransformedArray(int* nums, int numsSize, int a, int b, int c, int*
returnSize) {

}
```

Go Solution:

```
// Problem: Sort Transformed Array
// Difficulty: Medium
// Tags: array, math, 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 sortTransformedArray(nums []int, a int, b int, c int) []int {

}
```

Kotlin Solution:

```
class Solution {
fun sortTransformedArray(nums: IntArray, a: Int, b: Int, c: Int): IntArray {

}
}
```

Swift Solution:

```
class Solution {
func sortTransformedArray(_ nums: [Int], _ a: Int, _ b: Int, _ c: Int) ->
[Int] {

}
}
```

Rust Solution:

```
// Problem: Sort Transformed Array
// Difficulty: Medium
// Tags: array, math, 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 sort_transformed_array(nums: Vec<i32>, a: i32, b: i32, c: i32) ->
Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {Integer[]} nums
# @param {Integer} a
# @param {Integer} b
# @param {Integer} c
# @return {Integer[]}
def sort_transformed_array(nums, a, b, c)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[] $nums
* @param Integer $a
* @param Integer $b
* @param Integer $c
* @return Integer[]
*/
function sortTransformedArray($nums, $a, $b, $c) {

}
}
```

Dart Solution:

```
class Solution {
List<int> sortTransformedArray(List<int> nums, int a, int b, int c) {

}
}
```

Scala Solution:

```
object Solution {
def sortTransformedArray(nums: Array[Int], a: Int, b: Int, c: Int):
Array[Int] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec sort_transformed_array(nums :: [integer], a :: integer, b :: integer, c
:: integer) :: [integer]
def sort_transformed_array(nums, a, b, c) do

end
end
```

Erlang Solution:

```
-spec sort_transformed_array(Nums :: [integer()], A :: integer(), B ::
integer(), C :: integer()) -> [integer()].
sort_transformed_array(Nums, A, B, C) ->
.
```

Racket Solution:

```
(define/contract (sort-transformed-array nums a b c)
(-> (listof exact-integer?) exact-integer? exact-integer? exact-integer?
(listof exact-integer?))
)
```