Problem 977: Squares of a Sorted Array

Problem Information

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

Problem Description

Given an integer array

nums

sorted in

non-decreasing

order, return

an array of

the squares of each number

sorted in non-decreasing order

.

Example 1:

Input:

nums = [-4,-1,0,3,10]

Output:

[0,1,9,16,100]

Explanation:

After squaring, the array becomes [16,1,0,9,100]. After sorting, it becomes [0,1,9,16,100].

Example 2:

Input:

nums = [-7,-3,2,3,11]

Output:

[4,9,9,49,121]

Constraints:

1 <= nums.length <=

10

4

-10

4

<= nums[i] <= 10

4

nums

is sorted in

non-decreasing

order.

Follow up:

Squaring each element and sorting the new array is very trivial, could you find an

O(n)

solution using a different approach?

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

```
class Solution {
fun sortedSquares(nums: IntArray): IntArray {
}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Squares of a Sorted Array
* Difficulty: Easy
* Tags: array, 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> sortedSquares(vector<int>& nums) {

}
};
```

Java Solution:

```
/**
* Problem: Squares of a Sorted Array
* Difficulty: Easy
* Tags: array, 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[] sortedSquares(int[] nums) {

}
}
```

Python3 Solution:

```
"""
Problem: Squares of a Sorted Array
Difficulty: Easy
Tags: array, 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 sortedSquares(self, nums: List[int]) -> List[int]:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Squares of a Sorted Array
* Difficulty: Easy
* Tags: array, 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 sortedSquares = function(nums) {

};
```

TypeScript Solution:

```
/**
* Problem: Squares of a Sorted Array
* Difficulty: Easy
* Tags: array, 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 sortedSquares(nums: number[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Squares of a Sorted Array
* Difficulty: Easy
* Tags: array, 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[] SortedSquares(int[] nums) {

}
}
```

C Solution:

```
/*
* Problem: Squares of a Sorted Array
* Difficulty: Easy
* Tags: array, 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* sortedSquares(int* nums, int numsSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Squares of a Sorted Array
// Difficulty: Easy
// Tags: array, 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 sortedSquares(nums []int) []int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Squares of a Sorted Array
// Difficulty: Easy
// Tags: array, 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 sorted_squares(nums: Vec<i32>) -> Vec<i32> {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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