Squares of a Sorted Array

03 July 2022 09:41

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?

From <https://leetcode.com/explore/learn/card/fun-with-arrays/521/introduction/3240/>

First approach

```
class Solution {
public:
   vector<int> sortedSquares(vector<int>& nums) {
      int n=nums.size();
      for(int i=0;i<n;i++){
         nums[i]=nums[i]*nums[i];
      }
      sort(nums.begin(),nums.end());
      return nums;
   }
};
```

The most basic way to solve this solution is
->Run a loop
->Traverse each element of vector and multiple it same element and store at same position
->Sort the vector
Time Complexity - O(nlogn)
Space Complexity O(1)

Second approach

```
class Solution {
public:
   vector<int> sortedSquares(vector<int>& nums) {
      //we are given that vector is given in sorted ascending order
      int n=nums.size()-1;
      int s=0,e=n;
      vector<int>res(nums.size());
      for(int i=n;i>=0;i--){
         if(abs(nums[e])>abs(nums[s])){
            res[i]=nums[e]*nums[e];
            e--;
         }
         else{
            res[i]=nums[s]*nums[s];
            s++;
         }
      }
      return res;
   }
};
```

Since in Question it is given that vector is in ascending order

-4	-1	0	3	10

We will use Absolute function to change sign of -ve integers

4	1	0	3	10

Now take start and end pointers (start=0,end=len-1)
Create a result vector
->Run a loop
->check if end element>start element
    If true
    ->store the square of end element (i.e v[end]) since it is largest
    -> decrement end-- so it must be pointing to next element to
    Else
    ->store the square of start element (i.e v[start]) since it is largest
    ->increment start to point next element

Time Complexity- O(n)
Space complexity -O(n)