315. Count of Smaller Numbers After Self

Hard

You are given an integer array `nums` and you have to return a new `counts` array. The `counts` array has the property where `counts[i]` is the number of smaller elements to the right of `nums[i]`.

Example 1:

```
Input: nums = [5,2,6,1]
Output: [2,1,1,0]
Explanation:
To the right of 5 there are 2 smaller elements (2 and 1).
To the right of 2 there is only 1 smaller element (1).
To the right of 6 there is 1 smaller element (1).
To the right of 1 there is 0 smaller element.
```

Example 2:

```
Input: nums = [-1]
Output: [0]
```

Example 3:

```
Input: nums = [-1,-1]
Output: [0,0]
```

Constraints:

- `1 <= nums.length <=` 10^5
- -10^4 `<= nums[i] <=` 10^4

Related Topics

⌄

ArrayBinary SearchDivide and ConquerBinary Indexed TreeSegment TreeMerge SortOrdered Set

Similar Questions

⌄

Count of Range Sum
Hard
Queue Reconstruction by Height
Medium
Reverse Pairs
Hard
How Many Numbers Are Smaller Than the Current Number
Easy
Count Good Triplets in an Array
Hard

Quick Navigation

★ ★ ★ ★ ★

★
★
★
★
★

Average Rating: 4.51 (59 votes)

Premium

Solution

Overview

The problem is straightforward. For each `num` in `nums`, we need to obtain the number of smaller elements after `num`.

A straightforward approach is to use brute force with two for-loops. The first loop iterates over all `num` in `nums`, and the second loop iterates over all elements after `num`. However, this approach costs O(N^2)$O(N^2)$ and yields *Time Limit Exceed*, given that NN is the length of `nums`.

Luckily, there are two helpful data structures: segment tree and binary indexed tree, which are able to do the range query in logarithmic time.

Also, a solution based on Merge Sort is available.

Below, we will discuss each of the three approaches: *Segment Tree*, *Binary Indexed Tree*, and *Merge Sort*.

> After you finish, you can practice by solving some similar questions:
>
> - Reverse Pairs
> - Create Sorted Array through Instructions

Approach 1: Segment Tree

Intuition

> **Prerequisite: segment tree**
>
> If you are not familiar with segment trees, you should check out our Recursive Approach to segment trees tutorial before continuing.
>
> Also, here are some relevant applications for segment trees that you can practice on:
>
> - Range Sum Query - Mutable
> - Count of Range Sum
>
> For a full list, check out the segment tree Tag.

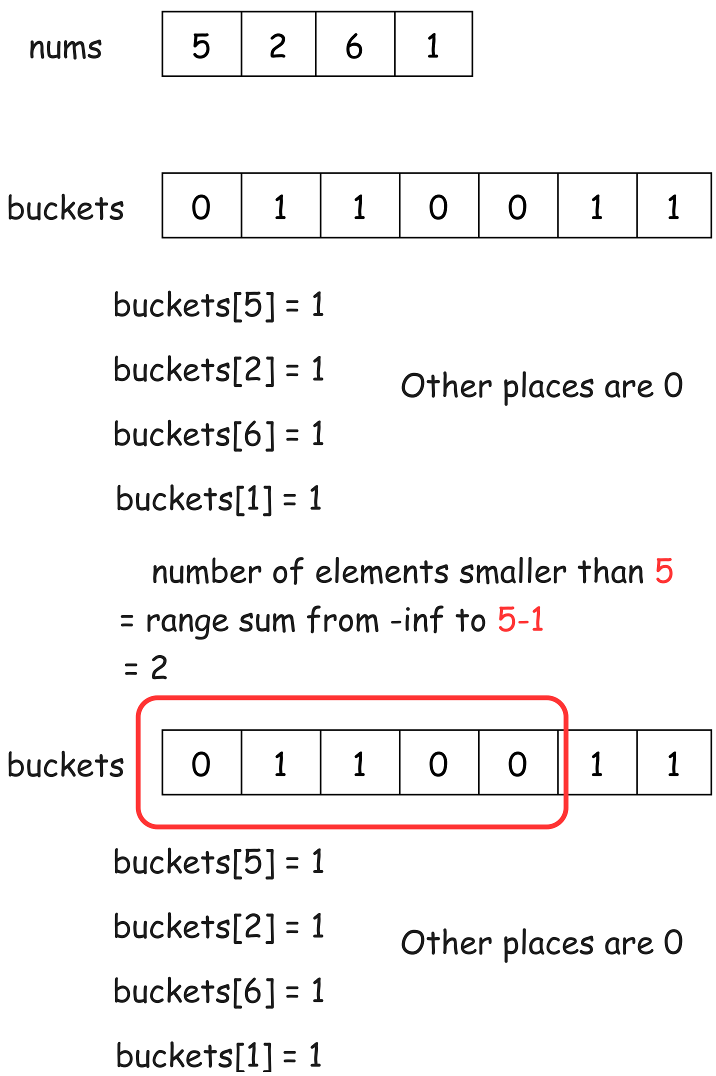
For a particular element in `nums`, located at index `i`, we want to count how many of the numbers on the right side of index `i` are smaller than `nums[i]`. Notice that the value of the smaller numbers must be in the range (-\infty, \text{nums[i]}-1]$(-\infty, \text{nums[i]} - 1]$.

Hence, if we can find the count of **each number** in the range (-\infty, \text{nums[i]}-1]$(-\infty, \text{nums[i]} - 1]$ on the right side of index `i`, then the answer will be the sum of those counts.

Therefore, for each index `i`, we need a query to find the sum of those counts. Recall that the segment tree and the binary indexed tree are two data structures that are generally helpful when solving range query problems.

Since we need counts of values, we can use an approach similar to [bucket sort](), where we have buckets of values and `buckets[value]` stores the count of `value`. For each value, we increment `buckets[value]` by 1. With this approach, the number of elements smaller than `nums[i]` is the range sum of (-\infty, \text{num}-1]$(-\infty, \text{num} - 1]$ in buckets.

With the help of a segment tree or binary indexed tree, we can perform the range sum query in logarithmic time.



With the given constraint `-10^4 <= nums[i] <= 10^4`, we can initialize buckets from `-10^4` to `10^4`.

Wait, there is a problem: Usually, we store buckets in an array, so the indices of buckets are non-negative. However, here we need to store some **negative** values. How can we resolve this problem?

There are two solutions:

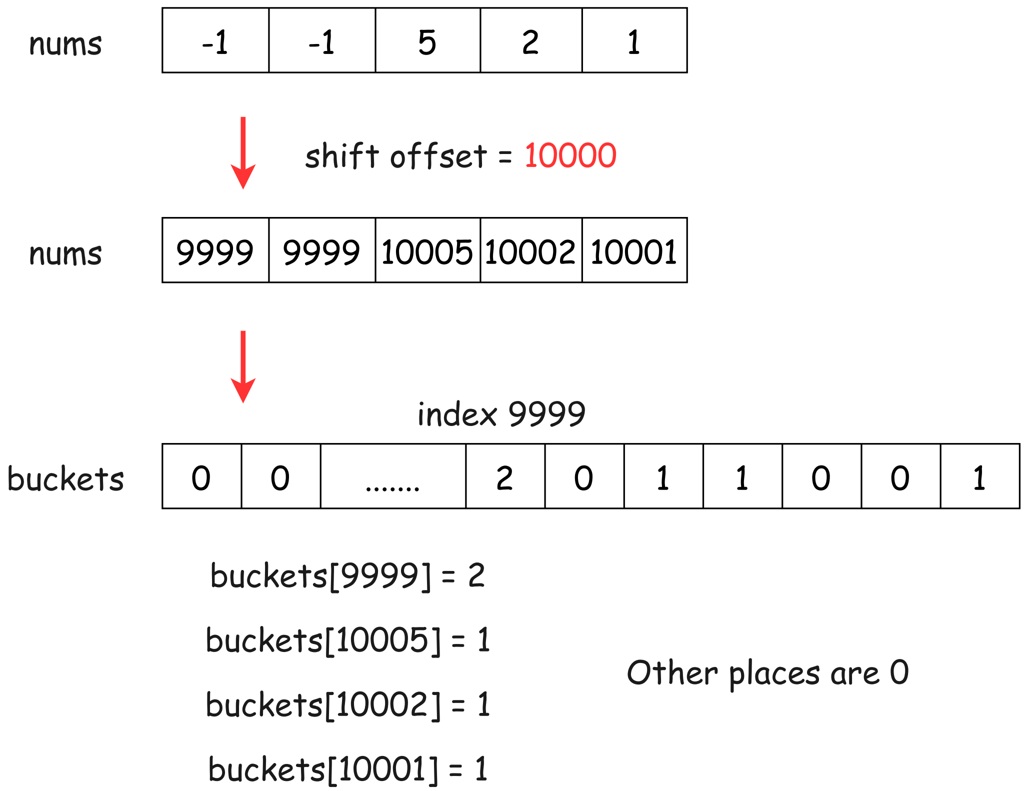
1. Use a map rather than an array.
2. Shift all numbers to non-negative.

Both solutions work, and here we have chosen the second one since it is easier to implement. Interested readers are welcome to try the first one on their own.

To shift all numbers to non-negative, we simply add a constant. Here we chose the constant `offset = 10^4` and increase each numl `offset`:

```
nums[i] = nums[i] + offset
```

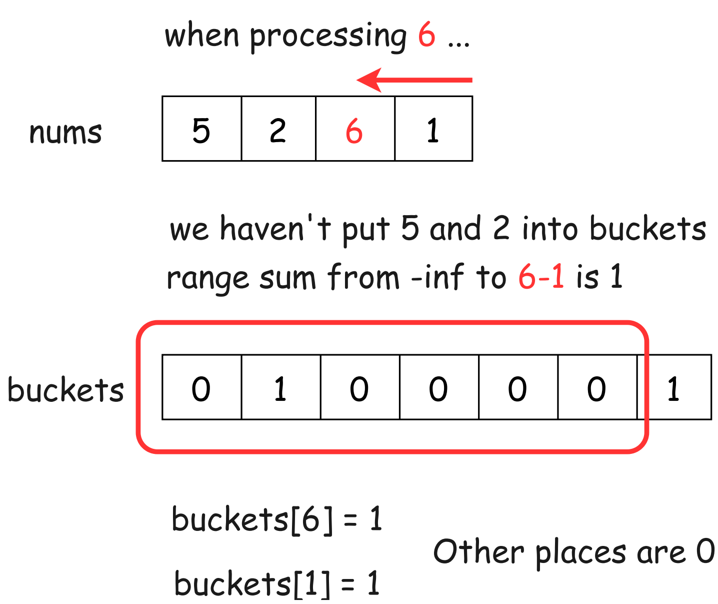
The smallest number `-10^4` becomes `0` under this shift.



Note that while querying a particular index, we only need to consider elements that are on the right side of the index. Therefore we need to make sure that when we query an index, say `i`, only elements from index `i+1` to the end of the array are present in the buckets.

To achieve this, we need to traverse `nums` from **right to left**, while performing range sum queries and updating the counts.

Similarly, with the help of a segment tree or binary indexed tree, we can perform the updates in logarithmic time.



(For convenience, the offset is not included in the above picture.)

Algorithm

- Implement the segment tree. Since the tree is initialized with all zeros, only `update` and `query` need to be implemented. Set o `10^4`.
- Iterate over each `num` in `nums` in reverse. For each `num`:

- Shift `num` to `num + offset`.
- Query the number of elements in the segment tree smaller than `num`.
- Update the count of `num` in the segment tree.

- Return the result.

Implementation

C++ | Java | Python3 | Copy

```
class Solution {
public:
    vector<int> countSmaller(vector<int>& nums) {
        int offset = 1e4;        // offset negative to non-negative
        int size = 2 * 1e4 + 1;  // total possible values in nums
        vector<int> tree(size * 2);
        vector<int> result;

        for (int i = nums.size() - 1; i >= 0; i--) {
            int smaller_count = query(0, nums[i] + offset, tree, size);
            result.push_back(smaller_count);
            update(nums[i] + offset, 1, tree, size);
        }
        reverse(result.begin(), result.end());
        return result;
    }

    // implement segment tree
    void update(int index, int value, vector<int>& tree, int size) {
        index += size;  // shift the index to the leaf
        // update from leaf to root
        tree[index] += value;
        while (index > 1) {
            index /= 2;
            tree[index] = tree[index * 2] + tree[index * 2 + 1];
        }
    }
```

Complexity Analysis

Let NN be the length of `nums` and MM be the difference between the maximum and minimum values in `nums`.

Note that for convenience, we fix M=2*10^4M = 2 ∗ 104 in the above implementations.

- Time Complexity: O(N\log(M))O(N log(M)).
  We need to iterate over `nums`. For each element, we spend O(\log(M))O(log(M)) to find the number of smaller elements after it, and spend O(\log(M))O(log(M)) time to update the counts. In total, we need O(N \cdot \log(M)) = O(N\log(M)) O(N · log(M)) = O(N log(M)) time.

- Space Complexity: O(M)O(M), since we need, at most, an array of size 2M+22M + 2 to store the segment tree.
  We need at most M+1M + 1 buckets, where the extra 11 is for the value 00. For the segment tree, we need twice the number of buckets, which is (M+1)\times 2 = 2M+2(M + 1) × 2 = 2M + 2.

Approach 2: Binary Indexed Tree (Fenwick Tree)

Intuition

> **Prerequisite: binary indexed tree**
>
> If you are not familiar with binary indexed tree (BIT), you should check relevant tutorials, such as [Range Sum Query 2D - Mutable]() before continuing.
>
> Also, here are some relevant applications for binary indexed trees that you can practice on:
>
> - [Range Sum Query - Mutable]()

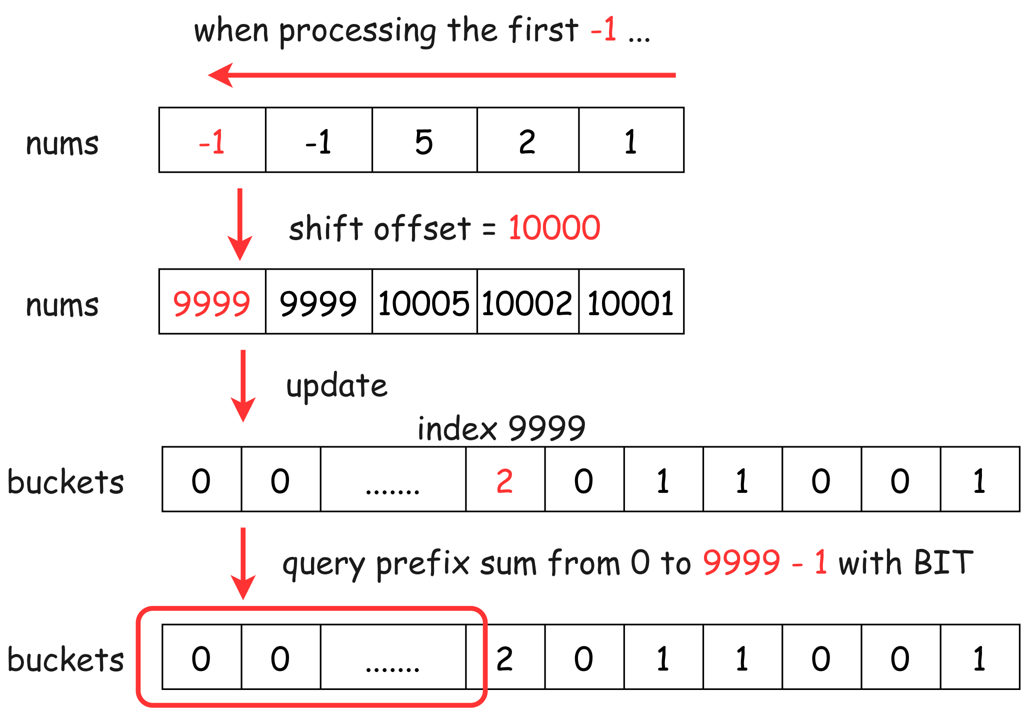
- Count of Range Sum

(Yes, many problems which can be solved by segment tree can also be solved by binary indexed tree.)

For a full list, you can check the binary indexed tree Tag.

Binary indexed tree is similar to segment tree. It allows us to perform a prefix query, such as prefix sum, in \loglog time. Can we transform this problem into a **prefix sum** problem?

Yes, using the same tricks that we used in approach 1, buckets and shift, we can transform the number of smaller elements into a prefix sum for the range [0, \text{num}+\text{offset}-1][0, num + offset – 1], where \text{offset}=10^4offset = 104.



Similarly, when querying, we need to traverse `nums` from right to left in order to ensure that only the elements to the right are in the buckets.

Algorithm

- Implement the binary indexed tree. Since the tree is initialized with all zeros, only `update` and `query` need to be implemented. Set `offset = 10^4`.
- Iterate over each `num` in `nums` in reverse. For each `num`:
  - Shift `num` to `num + offset`.
  - Query the number of elements in the BIT that are smaller than `num`.
  - Update the count of `num` in the BIT.
- Return the result.

Implementation

C++ Java Python3 Copy

```
            int smaller_count = query(nums[i] + offset, tree);
            result.push_back(smaller_count);
            update(nums[i] + offset, 1, tree, size);
        }
        reverse(result.begin(), result.end());
        return result;
    }

    // implement Binary Index Tree
    void update(int index, int value, vector<int>& tree, int size) {
        index++;  // index in BIT is 1 more than the original index
        while (index < size) {
            tree[index] += value;
            index += index & -index;
        }
    }

    int query(int index, vector<int>& tree) {
        // return sum of [0, index)
        int result = 0;
        while (index >= 1) {
            result += tree[index];
            index -= index & -index;
        }
        return result;
    }
};
```

Complexity Analysis

Let NN be the length of `nums` and MM be the difference between the maximum and minimum values in `nums`.

Note that for convenience, we fix M=2*10^4M = 2 ∗ 104 in the above implementations.

- Time Complexity: O(N\log(M))O(N log(M)).
  We need to iterate over `nums`. For each element, we spend O(\log(M))O(log(M)) to find the number of smaller elements after it, and spend O(\log(M))O(log(M)) time to update the counts. In total, we need O(N \cdot \log(M)) = O(N\log(M)) O(N · log(M)) = O(N log(M)) time.

- Space Complexity: O(M)O(M), since we need, at most, an array of size M+2M + 2 to store the BIT.
  We need at most M+1M + 1 buckets, where the extra 11 is for the value 00. The BIT requires an extra dummy node, so the size is (M+1)+1 = M+2(M + 1) + 1 = M + 2.

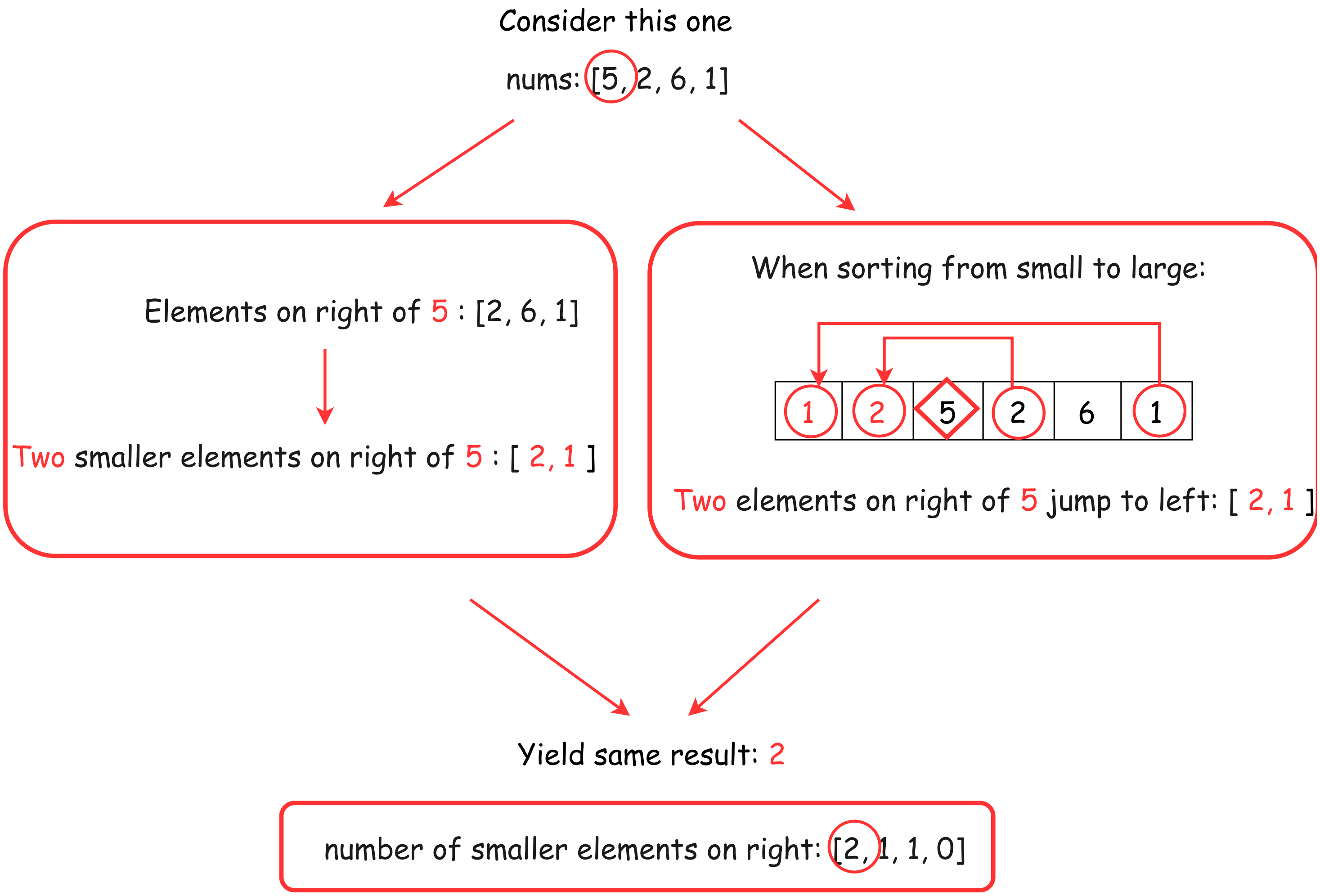
Approach 3: Merge Sort

Intuition

> **Prerequisite: Merge Sort**
>
> If you are not familiar with Merge Sort, you should check relevant tutorials before continuing.
>
> Also, here is a basic application of Merge Sort that you can practice on:
>
> - Sort an Array

To apply merge sort, one key observation is that:

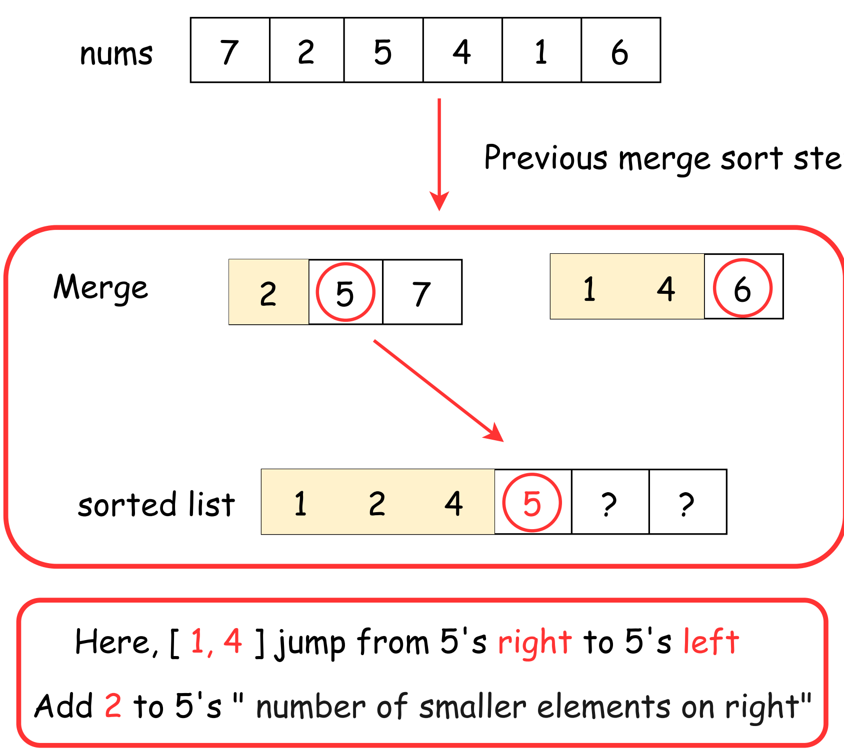
> The smaller elements on the right of a number will **jump from its right to its left** during the sorting process.



If we can record the numbers of those elements during sorting, then the problem is solved.

Can we modify the merge sort a little to meet our needs?

Consider when merging two sorted list



Yes! When we select an element `i` on the left array, we know that elements selected previously from the right array **jump** from `i`'s right to `i`'s left.

By adding the counts of those elements in every merge step, we get the total number of elements that jumped from `i`'s right to `i`'s left.

Algorithm

- Implement a merge sort function.
  - For each element `i`, the function records the number of elements jumping from `i`'s right to `i`'s left during the merge sort.
- Merge sort `nums`, store the number of elements jumping from right to left in `result`.
  - Alternatively, one can sort the *indices* with corresponding values in `nums`. That is to say, we are going to sort list `[0, 1, ..., n-1]` according to the comparator `nums[i]`. This helps to track the indices and update `result`. You can find additional details in the implementations below.
- Return `result`.

Implementation

C++ | Java | Python3 | Copy

```
class Solution {
public:
    vector<int> countSmaller(vector<int>& nums) {
        int n = nums.size();
        vector<int> result(n);
        vector<int> indices(n);  // record the index. we are going to sort this array
        for (int i = 0; i < n; i++) {
            indices[i] = i;
        }
        // sort indices with their corresponding values in nums, i.e., nums[indices[i]]
        mergeSort(indices, 0, n, result, nums);
        return result;
    }

    void mergeSort(vector<int>& indices, int left, int right, vector<int>& result,
                   vector<int>& nums) {
        if (right - left <= 1) {
            return;
        }
        int mid = (left + right) / 2;
        mergeSort(indices, left, mid, result, nums);
        mergeSort(indices, mid, right, result, nums);
        merge(indices, left, right, mid, result, nums);
    }

    void merge(vector<int>& indices, int left, int right, int mid, vector<int>& result,
               vector<int>& nums) {
```

Complexity Analysis

Let N be the length of `nums`.

- Time Complexity: $O(N \log(N))$. We need to perform a merge sort which takes $O(N \log(N))$ time. All other operations take at most $O(N)$ time.
- Space Complexity: $O(N)$, since we need a constant number of arrays of size $O(N)$.

[Report Article Issue]

Comments: 51

☑ Best ☐ Most Votes ☐ Newest to Oldest ☐ Oldest to Newest

Type comment here... (Markdown is supported)

Preview

Post

[yelun](#) ★462

June 4, 2021 8:22 AM

Read More

Dam... this one is hard

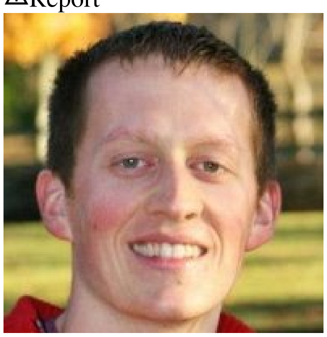
218

Show 6 replies

Reply

Share

Report



[b_clodius](#) ★808

June 26, 2021 6:02 PM

Read More

Such a simple problem statement but not an easy implementation. Time to review segment trees and binary index trees in depth!

73

Show 3 replies
Reply
Share
Report

[bennyRaichu](#)★168

June 30, 2021 10:04 AM

Read More

This is one of those questions that are simple enough to give you hope but are actually an omega pain in the butt

68

Reply
Share
Report

[abhi_1729](#)★45

June 29, 2021 3:21 PM

Read More

I am starting to appreciate merge sort a lot. There is a pattern to problems that use merge operations, but it's tricky to master.
I will put forward some of my learnings:

1. Divide and conquer style problems, where sorting data makes life easier. Can probably take advantage of merge operation
2. Binary search + merge style problems
3. Has a trivial O(N^2) solution but O(NLogN) is expected

41

Show 1 reply

Reply
Share
Report

ellait 81

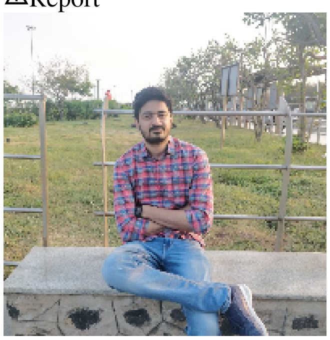
January 7, 2022 12:26 AM

Read More

Besides 'Easy', 'Medium' and 'Hard' they should have a new category called 'We really don't want to hire you'

26

Show 1 reply
Reply
Share
Report



user9981J 89

August 19, 2021 4:46 PM

Read More

Cannot understand the merge sort solution... playing with index rather than actual sorting

10

Reply
Share
Report

Luffy2020★313

July 4, 2021 12:22 AM

Read More

The beauty of Merge sort!

10

Reply
Share
Report

Derek_Y★6

Last Edit: June 5, 2021 12:43 PM

Read More

This is definately one of the best well organized solutions and the sample code is clean and readable.

16

Reply
Share
Report

SandSt0rm★9

Last Edit: July 17, 2021 2:54 PM

Read More

Solved by first sorting all elements and then iterating from left to right while using modified binary search to return the leftmost position in the sorted array:

```
def binarySearch(self, nums, target):
    lo = 0
    hi = len(nums) - 1
    res = -1
    while lo<=hi:
        mid = (lo+hi) // 2
        #print(lo, hi, mid)
        if nums[mid] > target:
            hi = mid - 1
```

```
        elif nums[mid] < target:
            lo = mid + 1
        else:
            res = mid
            hi = mid - 1
    return res
```

9

Show 7 replies
Reply
Share
Report

```
def countSmaller(self, nums: List[int]) -> List[int]:
    nums_sorted = sorted(nums)
ns)):
earch(nums_sorted, nums[i])
dx)
```

[jutsu](https://leetcode.com) ★33

May 22, 2021 10:18 PM

Read More

Is tree size being 2*size of array safe for the segment tree ?
I thought we need 4*size to represent the segment tree.

5

Show 4 replies
Reply
Share
Report

-
- 1
- 2
- 3
- 4
- 5
- 6
-

You don't have any submissions yet.

Problems
Pick One
Prev
315/2354
Next

i

C++

Autocomplete

i

{}

xxxxxxxxxx

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

    }
};
```

Console

Contribute

i

Run Code Submit

///

Type here...(Markdown is enabled)

×

Saved

All Problems

search problems

1

#1 Two Sum

Easy

✓#2 Add Two Numbers

Medium

✓#3 Longest Substring Without Repeating Characters

Medium

#4 Median of Two Sorted Arrays

Hard

✓#5 Longest Palindromic Substring

Medium

✓#6 Zigzag Conversion

Medium

✓#7 Reverse Integer

Medium

✓#8 String to Integer (atoi)
Medium
✓#9 Palindrome Number
Easy
#10 Regular Expression Matching
Hard
✓#11 Container With Most Water
Medium
✓#12 Integer to Roman
Medium
✓#13 Roman to Integer
Easy
✓#14 Longest Common Prefix
Easy
✓#15 3Sum
Medium
#16 3Sum Closest
Medium
✓#17 Letter Combinations of a Phone Number
Medium
✓#18 4Sum
Medium
✓#19 Remove Nth Node From End of List
Medium
✓#20 Valid Parentheses
Easy
✓#21 Merge Two Sorted Lists
Easy
✓#22 Generate Parentheses
Medium
✓#23 Merge k Sorted Lists
Hard
#24 Swap Nodes in Pairs
Medium
✓#25 Reverse Nodes in k-Group
Hard
✓#26 Remove Duplicates from Sorted Array
Easy
✓#27 Remove Element
Easy
✓#28 Implement strStr()
Easy
✓#29 Divide Two Integers
Medium
#30 Substring with Concatenation of All Words
Hard
✓#31 Next Permutation
Medium
#32 Longest Valid Parentheses
Hard
✓#33 Search in Rotated Sorted Array
Medium
#34 Find First and Last Position of Element in Sorted Array
Medium
✓#35 Search Insert Position
Easy
#36 Valid Sudoku
Medium
#37 Sudoku Solver
Hard
#38 Count and Say
Medium
✓#39 Combination Sum
Medium