MCS – 253P ADVANCED PROGRAMMING AND PROBLEM SOLVING

HOMEWORK –7 (Count of Smaller numbers after self)

Aswin Sampath
saswin@uci.edu
(53844684)

Description | Editorial | Solutions (2K) | Submissions

315. Count of Smaller Numbers After Self

Hard 8.6K 231

Companies

Given an integer array `nums`, return *an integer array* `counts` *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:

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`

Understanding the Problem

The task involves taking an integer array nums and constructing an array counts, where counts[i] represents the number of smaller elements to the right of nums[i].

The algorithm needs to return an array where each element counts[i] represents the count of elements smaller than nums[i] occurring to its right in the array.

Identifying Edge Cases

Before analyzing the code and providing a solution, let's identify potential edge cases:

1. An empty input array: When the input array nums is empty, the output array counts will also be empty.
2. Single-element array: When the input array nums has only one element, the count of smaller elements to the right will be 0.

Effective Test Cases

To validate the solution, consider the following test cases:

A mixed array with positive and negative integers

- Input: nums = [5, -2, 6, 1, 0]
- Expected Output: [2, 0, 2, 1, 0]

An array with all elements equal:

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

An array with descending elements:

- Input: nums = [5, 4, 3, 2, 1]
- Expected Output: [4, 3, 2, 1, 0]

Algorithmic Solution

The C++ code solves the problem using a modified merge sort algorithm to count smaller elements to the right.

- It constructs a vector of pairs v, where each pair contains the number from the input array nums and its index.
- The merge function sorts the pairs in descending order and updates the count vector based on the elements' indices.
- The mergeSort function recursively divides the array and merges pairs while performing the count updates during merging.
- Finally, it performs a merge sort on the pairs in descending order, updates the count vector accordingly, and returns the count vector as the output.

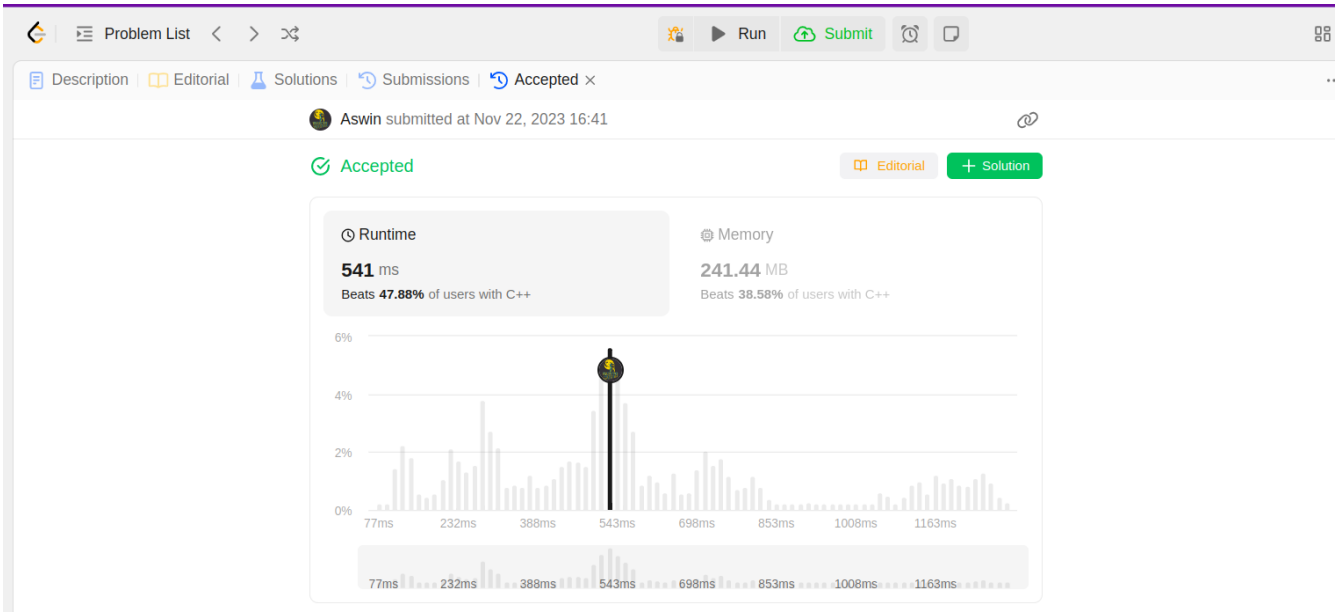
Time and Space Complexity Analysis

The time complexity for this algorithm is O(n log n), where n is the number of elements in the input array. The merge sort operation dominates the time complexity.

The space complexity is O(n) due to the usage of auxiliary vectors and recursion stack space.

Code:

```cpp
class Solution {
public:
    void merge(vector<int> &count, vector<pair<int, int> > &v, int left, int mid, int right) {
        vector<pair<int, int> > tmp(right-left+1);
        int i = left;
        int j = mid+1;
        int k = 0;

        while (i <= mid && j <= right) {
            // mind that we're sorting in descending order
            if (v[i].first <= v[j].first) {
                tmp[k++] = v[j++];
            }
            else {
                // only line responsible to update count, related to problem constraint,
                // remaining part is just regular mergeSort
                count[v[i].second] += right - j + 1;
                tmp[k++] = v[i++];
            }
        }
        while (i <= mid) {
            tmp[k++] = v[i++];
        }
        while (j <= right) {
            tmp[k++] = v[j++];
        }
        for (int i = left; i <= right; i++)
            v[i] = tmp[i-left];
    }

    void mergeSort(vector<int> &count, vector<pair<int, int> > &v, int left, int right) {
        if (left >= right)
            return;

        int mid = left + (right-left)/2;
        mergeSort(count, v, left, mid);
        mergeSort(count, v, mid+1, right);
        merge(count, v, left, mid, right);
    }

    vector<int> countSmaller(vector<int>& nums) {
        int N = nums.size();

        vector<pair<int, int> > v(N);
        for (int i = 0; i < N; i++)
            v[i] = make_pair(nums[i], i);

        vector<int> count(N, 0);
        // sorting in descending order
        mergeSort(count, v, 0, N-1);

        return count;
    }
};
```


Problem List
Run
Submit
Description
Editorial
Solutions
Submissions
Accepted
Aswin submitted at Nov 22, 2023 16:41
Accepted
Editorial
Solution
Runtime
541 ms
Beats 47.88% of users with C++
Memory
241.44 MB
Beats 38.58% of users with C++
6%
4%
2%
0%
77ms
232ms
388ms
543ms
698ms
853ms
1008ms
1163ms