medium.com

Leetcode 315. Count of Smaller Numbers After Self - Mithlesh Kumar - Medium

Mithlesh Kumar

4-5 minutes

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:

Input: [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.

I have wasted 20hrs (almost 2day) about searching 100+ articles. But I now found it was easy.

Pre-Requisite: at least know Merge sort.

Well, the idea is to find the number of **inversion** at **each** index of array.

Note: **Inversion** (it is number of pairs such that `A[i]>A[j] for all j>i`).
or Simply number of elements which are less than `A[i]`

For example: `nums = [5 3 1 2 4]`, we can see here that
Elements less than `5` is `[3 1 2 4]`, total `count = 4`.
Elements less than `3` is `[1 2]`, total `count = 2`.
Elements less than `1` is `[ ]`, total `count = 0`.
Elements less than `2` is `[]`, total `count = 0`.
Elements less than `4` is `[]`, total `count = 0`.
So resultant vector of count is `[4 2 0 1 0]`

There are various approaches to solve this problem,
like you have seen solved using Binary Index Trees, Segment Trees,
But I am going to solve it using **merge sort**.

Why I am using merge sort, because
`While merging two sorted subarrays i.e., left & right, we check left or right element who is greater or lesser. then we merge. Right?`

We can exploit this property.
`-Suppose we are in merge step of merge sort.`
`-Then we can simply take count of elements (left[i]>right[j]),`
`I mean where left element is greater than right.`

Further we are just one step, when `left[i]<=right[j]`, then simply increment the count at respective index of `left[i]`.

This is code.
I have used a slight different version of merge sort, instead of

sorting array, I have sorted indices.

The code is self-explanatory.

```
class Solution {
public:
   vector<int> merge(vector<int>& left, vector<int>& right,
vector<int>& nums, vector<int>& res){
      int l = 0, lsize = left.size(),
         r = 0, rsize = right.size();
      int smaller = 0; // count number of smaller on right
      vector<int> index;
      while(l<lsize && r<rsize){
         if(nums[left[l]] > nums[right[r]]){
            smaller++;// increment count when left > right
            index.push_back(right[r++]);
         }
         else{
            res[left[l]] += smaller; // put when left <= right
            index.push_back(left[l++]);
         }
      }
      while(l<lsize){
         res[left[l]] += smaller;
         index.push_back(left[l++]);
      }
      while(r<rsize){
         index.push_back(right[r++]);
      }
               return index;
   }
```

```
    void merge_sort(vector<int>& index, vector<int>& nums,
vector<int>& res){
    int size = index.size();
    if(size<2) return;
    vector<int> left, right;
    left.assign(index.begin(), index.begin()+size/2);
    right.assign(index.begin()+size/2, index.end());
    merge_sort(left, nums, res);
    merge_sort(right, nums, res);
        // now comes the merge step;
    index = merge(left, right, nums, res);
    return;
  }

    vector<int> countSmaller(vector<int>& nums) {
    vector<int> res(nums.size(),0);
    vector<int> oldIdx(nums.size(),0);
    iota(oldIdx.begin(), oldIdx.end(), 0);
    merge_sort(oldIdx, nums, res);
    return res;
  }

  };
```

Note: Pardon for my English.

Thank you!

my leetcode id: **trade_off**