315. Count of Smaller Numbers After Self

Question

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.
```

Thinking:

- Method 1: Slow, beats 18.69%

- find the right closed index whose value is same as current one.
- only need to calculate the value between these two and add the result of that index.

```
class Solution {
    public List<Integer> countSmaller(int[] nums) {
        Map<Integer, Integer> map = new HashMap<>();
        int[] arr = new int[nums.length];
        for(int i = nums.length - 1; i>= 0; i--){
            if(map.containsKey(nums[i])){
                arr[i] = map.get(nums[i]);
            }
            else{
                arr[i] = -1;
            }
            map.put(nums[i], i);
        }
        int[] result = new int[nums.length];
        for(int i = nums.length - 1; i >= 0; i--){
            int count = 0;
            if(arr[i] == -1){
                for(int j = i; j < nums.length; j++){
                    if(nums[i] > nums[j]) count++;
                }
                result[i] = count;
            }else{
                for(int j = i; j < arr[i]; j++){
                    if(nums[i] > nums[j]) count++;
                }
                result[i] = count + result[arr[i]];
            }
        }
        List<Integer> res = new LinkedList<>();
        for(int n : result) res.add(n);
        return res;
    }
}
```

- Method 2: Use index and frequency array. slow, beats 6.23%
  - Example[5,6,2,1]
    - We sort the array in ascending order[1,2,5,6]
    - we use another map to save the corresponding index: [2, 3, 1, 0], we need to remove the duplicate values.
    - we traversal the array from the end to the beginning, once we meet a

value, we add the frequency.

- Then we sum the appear number from the end to current index.

```
class Solution {
    public List<Integer> countSmaller(int[] nums) {
        LinkedList<Integer> result = new LinkedList<>();
        if(nums == null || nums.length == 0) return result;
        Map<Integer, Integer> map = new HashMap<>();
        int[] copy = Arrays.copyOf(nums, nums.length);
        Arrays.sort(nums);
        int count = 0;
        map.put(nums[nums.length - 1], count);
        for(int i = nums.length - 2; i >= 0; i--){
            if(nums[i] != nums[i + 1]){
                map.put(nums[i], ++count);
            }
        }
        int[] freq = new int[count+1];
        for(int i = copy.length - 1; i >= 0; i--){
            freq[map.get(copy[i])]++;
            int appear = 0;
            for(int j = count; j > map.get(copy[i]); j--){
                appear += freq[j];
            }
            result.addFirst(appear);
        }
        return result;
    }
}
```

- Method 3: BST, Beats 99.02%
  - We create a Node and look into this node:

```
    private static class Node{
    int val;
    int count; // Number of current value.
    int leftCount; // Number of left child Node for current node.
    Node left, right;
    public Node(int val){
        this.val = val;
        this.count = 1;
    }


} ```
```

  - For a right child node, the number of values smaller than current value

should be: node.leftCount + parent.count+ parent.leftCount, so here are three conditions:

a. given value equals to node.val: we need to add node.count: node.count++;

b. given value is smaller than current value:

- left is null, create a new node and return 0
- left is not null, return insert(node.left, val), means we recursively call the function.

c. given value is bigger than current value:

- right is null, create a new node and should return node.count + node.leftCount.
- right is not null, return node.count + node.leftcount + insert(node.right, val).

```
class Solution {
    private static class Node{
        int val;
        int count;
        int leftCount;
        Node left, right;
        public Node(int val){
            this.val = val;
            this.count = 1;
        }
    }
    public List<Integer> countSmaller(int[] nums) {
        LinkedList<Integer> result = new LinkedList<>();
        if(nums == null || nums.length == 0) return result;
        Node root = new Node(nums[nums.length - 1]);
        result.add(0);
        for(int i = nums.length - 2; i >= 0; i--){
            result.addFirst(insert(root, nums[i]));
        }
        return result;
    }
    private int insert(Node node, int val){
        if(val == node.val){
            node.count ++;
            return node.leftCount;
        }else if(val < node.val){
            node.leftCount++;
            if(node.left == null){
                node.left = new Node(val);
                return 0;
            }else{
```

```
                    return insert(node.left, val);
                }
            }else{
                if(node.right == null){
                    node.right = new Node(val);
                    return node.count + node.leftCount;
                }else return node.count + node.leftCount + insert(node.right
            }
        }
    }
```