Shortest Subarray To Be Removed To Make Array Sorted

solved by	Senan
Platform	LeetCode
difficulty	Medium
Serial	1574
tags	Stack
language	C++
solved on	@15/11/2024
link	https://leetcode.com/problems/shortest-subarray-to-be-removed-to-make-array-sorted/description/
Completion	✓

Intuition

To make an array non-decreasing, the minimum number of elements to be removed can be determined by keeping the largest sorted subarrays on the left and right and potentially merging them. The goal is to identify these subarrays efficiently and compute the smallest number of removals required.

Approach

1. **Identify Left Sorted Subarray:**
   - Traverse from the start of the array to find the first unsorted element. This gives the largest sorted prefix (`left`).
2. **Identify Right Sorted Subarray:**
   - Traverse from the end of the array to find the first unsorted element when going backward. This gives the largest sorted suffix (`right`).
3. **Check Initial Cases:**
   - If the entire array is already sorted, return `0` (no elements need to be removed).
4. **Compute Minimum Removals:**
   - Initially, the minimum removals are either removing all elements after the left prefix or before the right suffix (`min(n-left-1, right)`).
5. **Merge Left and Right Subarrays:**
   - Use two pointers (`i` from the left prefix and `j` from the right suffix) to check if elements from the two subarrays can merge into a sorted order.
   - Update the `answer` by minimizing the number of elements between the pointers that need to be removed (`j - i - 1`).
6. **Return the Result:**
   - The variable `answer` holds the minimum number of elements to be removed.

Complexity

Time Complexity:

- **O(n):**
  - Traversing the array to identify `left` and `right` takes O(n).
  - The two-pointer merging process also takes O(n) in total.

Space Complexity:

- **O(1):**
  - No additional space is used, apart from a few variables.

Code

```
class Solution {
public:
    int findLengthOfShortestSubarray(vector<int>& arr) {
        int n = arr.size();
        int left = 0, right = n - 1;

        while (left + 1 < n && arr[left] <= arr[left + 1]) left++;
        if (left == right) return 0;

        while (right > left && arr[right - 1] <= arr[right]) right--;

        int answer = min(n - left - 1, right);

        for (int i = 0, j = right; i <= left && j < n; ) {
            if (arr[i] <= arr[j]) {
                answer = min(answer, j - i - 1);
                i++;
            } else {
                j++;
            }
        }

        return answer;
    }
};
```