1574. Shortest Subarray to be Removed to Make Array Sorted

Solution by Prasanna Bartakke(Prasanna648)

Brute Force Approach

Remove every possible subarray and check whether the remaining array is in sorted order. The solution exceeds the time limit.

Code (Python3)
Link : Solution 1

```
class Solution:
    def findLengthOfShortestSubarray(self, arr: List[int]) -> int:
        n = len(arr)
        ans = float("inf")
        for i in range(n):
            for j in range(i, n+1):
                if self.issorted(arr[:i] + arr[j:]):
                    ans = min(ans, j - i)

        return ans
    def issorted(self, arr):
        l = len(arr)
        for i in range(1, l):
            if arr[i] < arr[i-1]:
                return False
        return True
```

Time Complexity -> O(n^3)

Sliding Window Approach

From left to right, check the longest possible subarray starting at index 0 which is in non decreasing order. This subarray ends at p1. If p1 is the end of the array(p1 = n - 1), the entire array is in non decreasing order and we do not have to delete any subarray. So answer will be zero in this case. Otherwise we have to remove the elements from p1 to n.

Let us use this array to visualise the operation: 1 2 3 4 3 2 3 4 5 3 2 3 4 5 6
Green elements remain and the red ones have to be deleted.

After the first operation, p1 = 3.
1 2 3 4 3 2 3 4 5 3 2 3 4 5 6

Now from right to left, check the longest possible non decreasing array starting at index p2 (initially p2 = n-1) and ending at the last element. From p2 to n, the array is in sorted order. Remove the subarray from 0 to p2-1.

After second operation, p2 = 10
1 2 3 4 3 2 3 4 5 3 2 3 4 5 6

The required number of elements to be removed is minimum of n – p1 – 1 and p2.
ans = min(n – p1 – 1, p2)

As we can see, in the sequence 1 2 3 4 3 2 3 4 5 3 2 3 4 5 6, we do not need to remove the entire left subarray, we can keep the elements from position 0 to 1(1,2) so that the sequence remains in sorted as shown below.
1 2 3 4 3 2 3 4 5 3 2 3 4 5 6

To find this we use the sliding window approach. Let the left and right pointers of the window be 0 and p2 initially. We have to remove certain elements between left and right.

We do the following till left pointer <= p1 and right pointer < n:

Case 1 : If arr[left] <= arr[right] we can keep the element at left in the array, so we increment left pointer.

Case 2: Else we cannot keep the element at left pointer, so we increment right pointer to make the sequence increasing.

ans = min(ans, right – left - 1)

Case 1:
left = 0
right = 10
1 2 3 4 3 2 3 4 5 3 2 3 4 5 6
As we can see arr[left] <= arr[right] so left can be included
1 2 3 4 3 2 3 4 5 3 2 3 4 5 6

Case 2:
left = 2
right = 10
1 2 3 4 3 2 3 4 5 3 2 3 4 5 6
arr[left] > arr[right]

increment right till a[left] <= arr[right]
1 2 3 4 3 2 3 4 5 3 2 3 4 5 6

Code (Python3)
Link : Solution 2

```
class Solution:
    def findLengthOfShortestSubarray(self, arr: List[int]) -> int:
        n = len(arr)
        ans = float("inf")
        p1 = 0
        while p1 < n - 1 and arr[p1] <= arr[p1 + 1]:
            p1 += 1
        # no need to remove anything
        if p1 == n - 1:
            return 0

        p2 = n - 1
        while p2 > 0 and arr[p2] >= arr[p2-1]:
            p2 -= 1

        ans = min(n - p1 - 1, p2)

        #sliding window
        left = 0
        right = p2

        while left <= p1 and right < n:
            if arr[left] <= arr[right]:
                left += 1
            else:
                right += 1
            ans = min(ans, right - left)

        return ans
```

Time Complexity :
We visit each element not more than 3 times.
Therefore complexity = O(3*n) = **O(n)**

Space Complexity : O(1)
Constant extra space is used.