leetcode 1574. Shortest Subarray to be Removed to Make Array Sorted

Approach #1: Double Pointers [Accepted]

Intuition

For each index i, find the minimum index j (j >= i) such that after remove the subarray arr[i...j], the remaining elements in arr are non-decreasing.

Algorithm

For each index i, if elements in subarray arr[0...i-1] are non-decreasing, then i could be the possible start index of the removed subarray.

Then, for each possible start index i, we need to find mininum index j (j >= i), such that elements in subarray arr[j+1...n-1] are non-decreasing and arr[j+1] >= arr[i-1].

As index i increases, the optimal value of index j cannot decreases. So we can apply a double pointer method to find the optimal j for every possible start index i.

C++:

```
class Solution {
public:
    int findLengthOfShortestSubarray(vector<int>& arr) {
        if((int)arr.size() == 1)
            return 0;

        int j = (int)arr.size() - 2;
        while(j >= 0 && arr[j] <= arr[j+1])
            --j;

        if(j < 0)
            return 0;
        int ans = (int)arr.size();
        for(int i = 0; i < (int)arr.size() && (i <= 1 || arr[i-2] <= arr[i-1]);
++i)
        {
            while(i > 0 && j < (int)arr.size() - 1 && arr[i-1] > arr[j+1])
                ++j;
            ans = min(ans, j - i + 1);
        }
        return ans;
    }
};
```

Java:

```
class Solution {
    public int findLengthOfShortestSubarray(int[] arr) {
        if(arr.length == 1)
            return 0;

        int j = arr.length - 2;
        while(j >= 0 && arr[j] <= arr[j+1])
            --j;

        if(j < 0)
            return 0;
        int ans = arr.length;
        for(int i = 0; i < arr.length && (i <= 1 || arr[i-2] <= arr[i-1]); ++i)
        {
            while(i > 0 && j < (int)arr.length - 1 && arr[i-1] > arr[j+1])
                ++j;
            ans = Math.min(ans, j - i + 1);
        }
        return ans;
    }
}
```

Python:

```
class Solution(object):
    def findLengthOfShortestSubarray(self, arr):
        if len(arr) == 1:
            return 0

        j = len(arr) - 2
        while j >= 0 and arr[j] <= arr[j+1]:
            j -= 1

        if j < 0:
            return 0
        ans = len(arr)
        i = 0
        while (i < len(arr)) and (i <= 1 or arr[i-2] <= arr[i-1]):
            while i > 0 and j < len(arr) - 1 and arr[i-1] > arr[j+1]:
                j += 1
            ans = min(ans, j - i + 1)
            i += 1
        return ans
```

Complexity Analysis:

- Time Complexity: *O(N)*, where *N* is the length of arr.
- Space Complexity: *O(1)*, constant space for two pointers.