Max Chunks To Make Sorted II

This question is the same as "Max Chunks to Make Sorted" except the integers of the given array are not necessarily distinct, the input array could be up to length `2000`, and the elements could be up to `10**8`.

Given an array `arr` of integers (**not necessarily distinct**), we split the array into some number of "chunks" (partitions), and individually sort each chunk. After concatenating them, the result equals the sorted array.

What is the most number of chunks we could have made?

Example 1:

```
Input: arr = [5,4,3,2,1]
Output: 1
Explanation:
Splitting into two or more chunks will not return the required result.
For example, splitting into [5, 4], [3, 2, 1] will result in [4, 5, 1, 2, 3], which i
sn't sorted.
```

Example 2:

```
Input: arr = [2,1,3,4,4]
Output: 4
Explanation:
We can split into two chunks, such as [2, 1], [3, 4, 4].
However, splitting into [2, 1], [3], [4], [4] is the highest number of chunks possibl
e.
```

Note:

- `arr` will have length in range `[1, 2000]`.
- `arr[i]` will be an integer in range `[0, 10**8]`.

Solution 1

Algorithm: Iterate through the array, each time all elements to the left are smaller (or equal) to all elements to the right, there is a new chunck.
Use two arrays to store the left max and right min to achieve O(n) time complexity.
Space complexity is O(n) too.
This algorithm can be used to solve ver1 too.

```
class Solution {
    public int maxChunksToSorted(int[] arr) {
        int n = arr.length;
        int[] maxOfLeft = new int[n];
        int[] minOfRight = new int[n];

        maxOfLeft[0] = arr[0];
        for (int i = 1; i < n; i++) {
            maxOfLeft[i] = Math.max(maxOfLeft[i-1], arr[i]);
        }

        minOfRight[n - 1] = arr[n - 1];
        for (int i = n - 2; i >= 0; i--) {
            minOfRight[i] = Math.min(minOfRight[i + 1], arr[i]);
        }

        int res = 0;
        for (int i = 0; i < n - 1; i++) {
            if (maxOfLeft[i] <= minOfRight[i + 1]) res++;
        }

        return res + 1;
    }
}
```

written by shawngao original link here

Solution 2

Same as Max Chunks To Make Sorted (ver. 1). We just need to normalize the input array so it contains the indexes. We use the sorting for the normalization, and one trick here is to have a stable sort, so that if we have the same value, the index will be lowest for the value appearing first.

```
int maxChunksToSorted(vector<int>& v) {
    vector<int> arr(v.size());
    iota(arr.begin(), arr.end(), 0);
    sort(arr.begin(), arr.end(), [&v](int i1, int i2) {return v[i1] == v[i2] ? i1 <
i2 : v[i1] < v[i2]; });

    for (auto i = 0, max_i = 0, ch = 0; i <= arr.size(); ++i) {
      if (i == arr.size()) return ch;
      max_i = max(max_i, arr[i]);
      if (max_i == i) ++ch;
    }
}
```

written by votrubac original link here

Solution 3

```
def maxChunksToSorted(self, arr):
        res, c1, c2 = 0, collections.Counter(), collections.Counter()
        for a, b in zip(arr, sorted(arr)):
            c1[a] += 1
            c2[b] += 1
            res += c1 == c2
        return res
```

written by lee215 original link here