educative

Grokking the Coding Interview: Patterns for Coding Questions

13% completed

Search Course

Solution Review: Problem Challenge 3

We'll cover the following

Minimum Window Sort (medium)

Given an array, find the length of the smallest subarray in it which when sorted will sort the whole array.

Example 1:

```
Input: [1, 2, 5, 3, 7, 10, 9, 12]
Output: 5
Explanation: We need to sort only the subarray [5, 3, 7, 10, 9] to make the whole array sorted
```

Example 2:

```
Input: [1, 3, 2, 0, -1, 7, 10]
Output: 5
Explanation: We need to sort only the subarray [1, 3, 2, 0, -1] to make the whole array sorted
```

Example 3:

```
Input: [1, 2, 3]
Output: 0
Explanation: The array is already sorted
```

Example 4:

```
Input: [3, 2, 1]
Output: 3
Explanation: The whole array needs to be sorted.
```

Solution

As we know, once an array is sorted (in ascending order), the smallest number is at the beginning and the largest number is at the end of the array. So if we start from the beginning of the array to find the first element which is out of sorting order i.e., which is smaller than its previous element, and similarly from the end of array to find the first element which is bigger than its previous element, will sorting the subarray between these two numbers result in the whole array being sorted?

Let's try to understand this with Example-2 mentioned above. In the following array, what are the first numbers out of sorting order from the beginning and the end of the array:

```
    [1, 3, 2, 0, -1, 7, 10]
```

1. Starting from the beginning of the array the first number out of the sorting order is '2' as it is smaller than its previous element which is '3'.
2. Starting from the end of the array the first number out of the sorting order is '0' as it is bigger than its previous element which is '-1'

As you can see, sorting the numbers between '3' and '-1' will not sort the whole array. To see this, the following will be our original array after the sorted subarray:

```
    [1, -1, 0, 2, 3, 7, 10]
```

The problem here is that the smallest number of our subarray is '-1' which dictates that we need to include more numbers from the beginning of the array to make the whole array sorted. We will have a similar problem if the maximum of the subarray is bigger than some elements at the end of the array. To sort the whole array we need to include all such elements that are smaller than the biggest element of the subarray. So our final algorithm will look like:

1. From the beginning and end of the array, find the first elements that are out of the sorting order. The two elements will be our candidate subarray.
2. Find the maximum and minimum of this subarray.
3. Extend the subarray from beginning to include any number which is bigger than the minimum of the subarray.
4. Similarly, extend the subarray from the end to include any number which is smaller than the maximum

MW
Explore
Tracks
My Courses
Edpresso
Refer a Friend
Create

of the subarray.

Code

Here is what our algorithm will look like:

Java | Python3 | C++ | JS

```java
class ShortestWindowSort {

  public static int sort(int[] arr) {
    int low = 0, high = arr.length - 1;
    // find the first number out of sorting order from the beginning
    while (low < arr.length - 1 && arr[low] <= arr[low + 1])
      low++;

    if (low == arr.length - 1) // if the array is sorted
      return 0;

    // find the first number out of sorting order from the end
    while (high > 0 && arr[high] >= arr[high - 1])
      high--;

    // find the maximum and minimum of the subarray
    int subarrayMax = Integer.MIN_VALUE, subarrayMin = Integer.MAX_VALUE;
    for (int k = low; k <= high; k++) {
      subarrayMax = Math.max(subarrayMax, arr[k]);
      subarrayMin = Math.min(subarrayMin, arr[k]);
    }

    // extend the subarray to include any number which is bigger than the minimum of the subarray
    while (low > 0 && arr[low - 1] > subarrayMin)
      low--;
    // extend the subarray to include any number which is smaller than the maximum of the subarray
    while (high < arr.length - 1 && arr[high + 1] < subarrayMax)
      high++;
```

RUN SAVE RESET

Close

Output 2.275s

```
5
5
0
3
```

Time complexity

The time complexity of the above algorithm will be $O(N)$.

Space complexity

The algorithm runs in constant space $O(1)$.

✓ MARK AS COMPLETED

← Back
Problem Challenge 3

Next →
Introduction

Report an Issue