Find the Minimum length Unsorted Subarray, sorting which makes the complete array sorted

Given an unsorted array arr[0..n-1] of size n, find the minimum length subarray arr[s..e] such that sorting this subarray makes the whole array sorted.

Examples:

1) If the input array is [10, 12, 20, 30, 25, 40, 32, 31, 35, 50, 60], your program should be able to find that the subarray lies between the indexes 3 and 8.

2) If the input array is [0, 1, 15, 25, 6, 7, 30, 40, 50], your program should be able to find that the subarray lies between the indexes 2 and 5.

Solution:
1) Find the candidate unsorted subarray
a) Scan from left to right and find the first element which is greater than the next element. Let *s* be the index of such an element. In the above example 1, *s* is 3 (index of 30).
b) Scan from right to left and find the first element (first in right to left order) which is smaller than the next element (next in right to left order). Let *e* be the index of such an element. In the above example 1, e is 7 (index of 31).

2) Check whether sorting the candidate unsorted subarray makes the complete array sorted or not. If not, then include more elements in the subarray.
a) Find the minimum and maximum values in *arr[s..e]*. Let minimum and maximum values be *min* and *max*. *min* and *max* for [30, 25, 40, 32, 31] are 25 and 40 respectively.
b) Find the first element (if there is any) in *arr[0..s-1]* which is greater than *min*, change *s* to index of this element. There is no such element in above example 1.
c) Find the last element (if there is any) in *arr[e+1..n-1]* which is smaller than max, change *e* to index of this element. In the above example 1, e is changed to 8 (index of 35).

3) return e - s + 1, which is the length of required sub-array.