See my latest update in repo [LeetCode](https://github.com/lzl124631x/LeetCode)

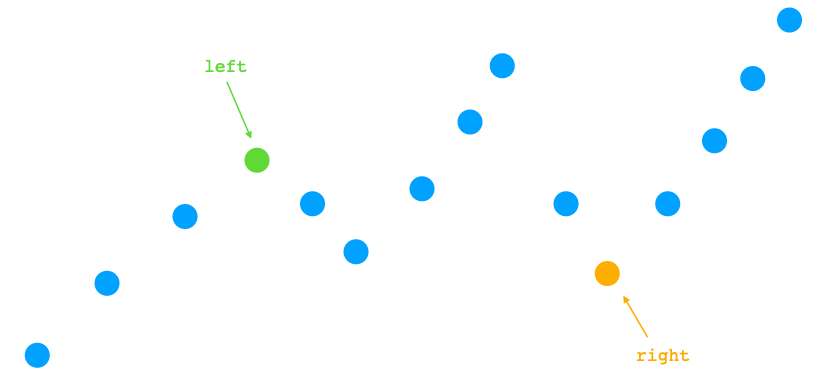
<https://leetcode.com/problems/shortest-subarray-to-be-removed-to-make-array-sorted/discuss/830480/C%2B%2B-O(N)-Sliding-window-Explanation-with-Illustrations>

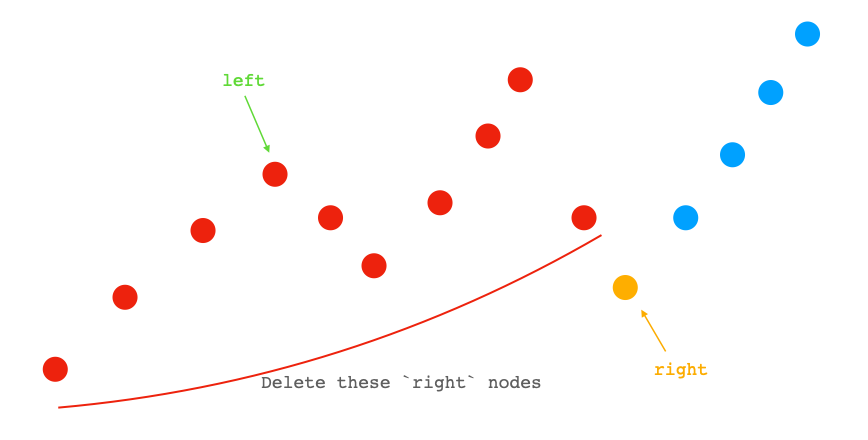
Solution 1.

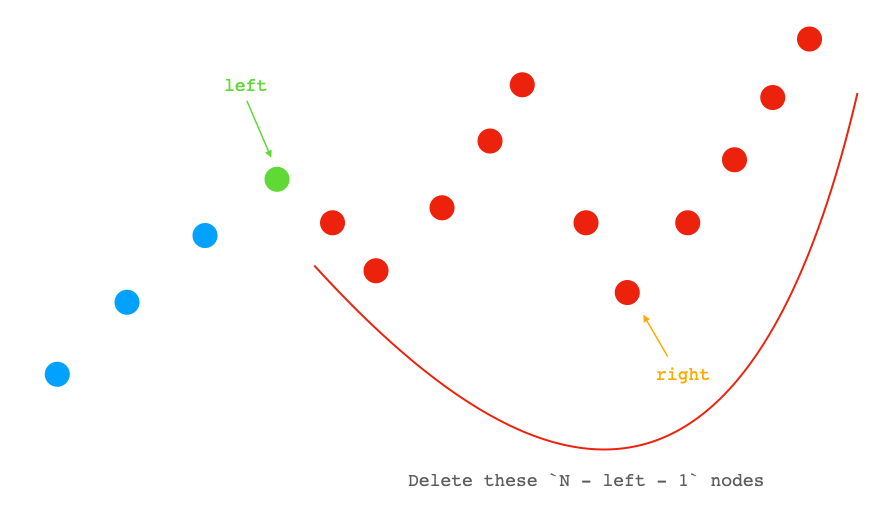
Scan from left to right, find the first index left that A[left] > A[left + 1].

If left == N - 1, this array is already non-descending, return 0.

Scan from right to left, find the first index right that A[right] < A[right - 1].



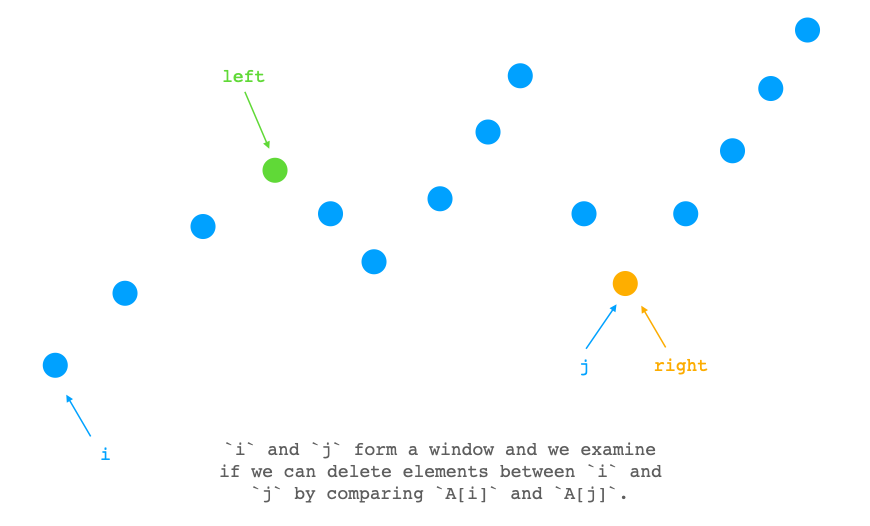
Now we loosely have two options, either deleting the left-side right nodes, or deleting the right-side N - left - 1 nodes.  




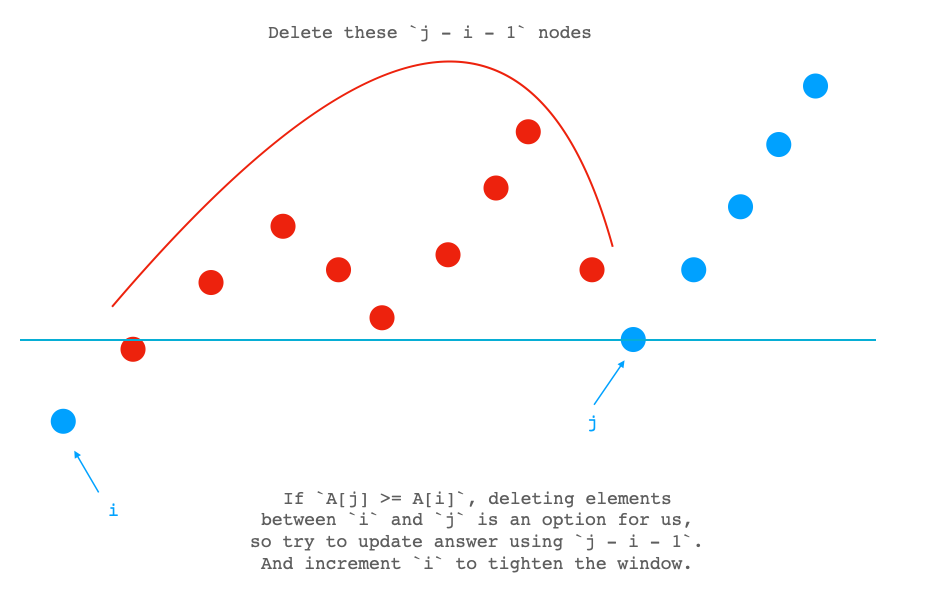
So the answer is at most min(N - left - 1, right).

Now we can use a sliding window / two pointers to get tighter result.

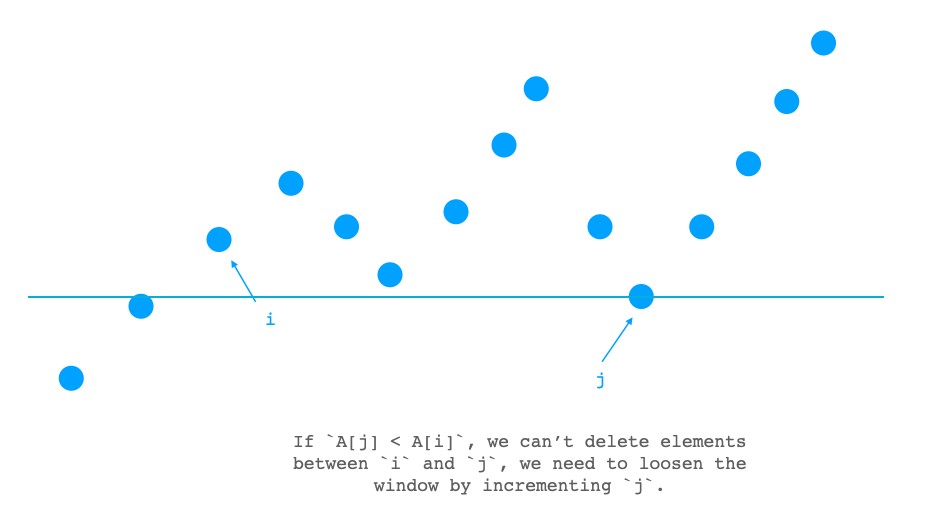
Let i = 0, j = right. And we examine if we can delete elements between i and j (exclusive) by comparing A[i] and A[j].



**Case 1:** A[j] >= A[i], we can delete elements inbetween, so we can try to update the answer using j - i - 1 and increment i to tighten the window.



**Case 2:** A[j] < A[i], we can't delete elements inbetween, so we increment j to loosen the window.



We loop until i > left or j == N. And the answer we get should be the minimal possible solution.

// OJ: https://leetcode.com/problems/shortest-subarray-to-be-removed-to-make-array-sorted/

// Author: github.com/lzl124631x

// Time: O(N)

// Space: O(1)

// Ref: https://leetcode.com/problems/shortest-subarray-to-be-removed-to-make-array-sorted/discuss/830416/Java-Increasing-From-Left-Right-and-Merge-O(n)

class Solution {

public:

int findLengthOfShortestSubarray(vector<int>& A) {

int N = A.size(), left = 0, right = N - 1;

while (left + 1 < N && A[left] <= A[left + 1]) ++left;

if (left == A.size() - 1) return 0;

while (right > left && A[right - 1] <= A[right]) --right;

int ans = min(N - left - 1, right), i = 0, j = right;

while (i <= left && j < N) {

if (A[j] >= A[i]) {

ans = min(ans, j - i - 1);

++i;

} else ++j;

}

return ans;

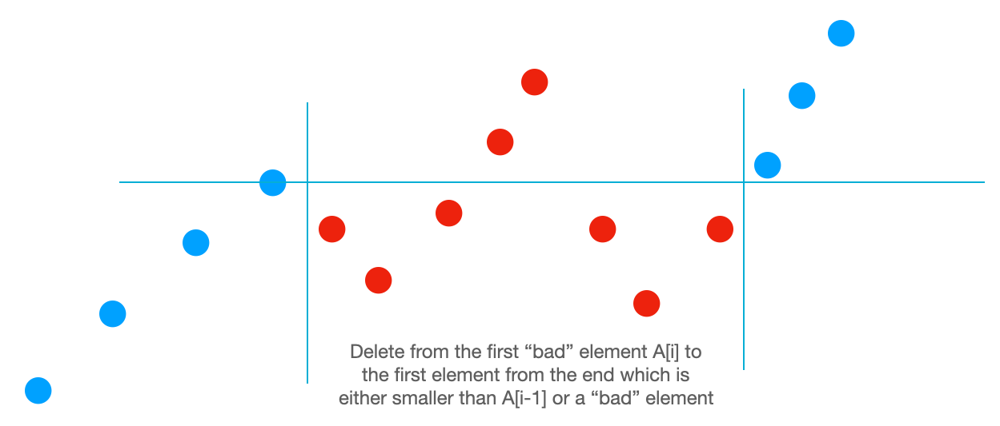
}

};

Original Approach (Incorrect)

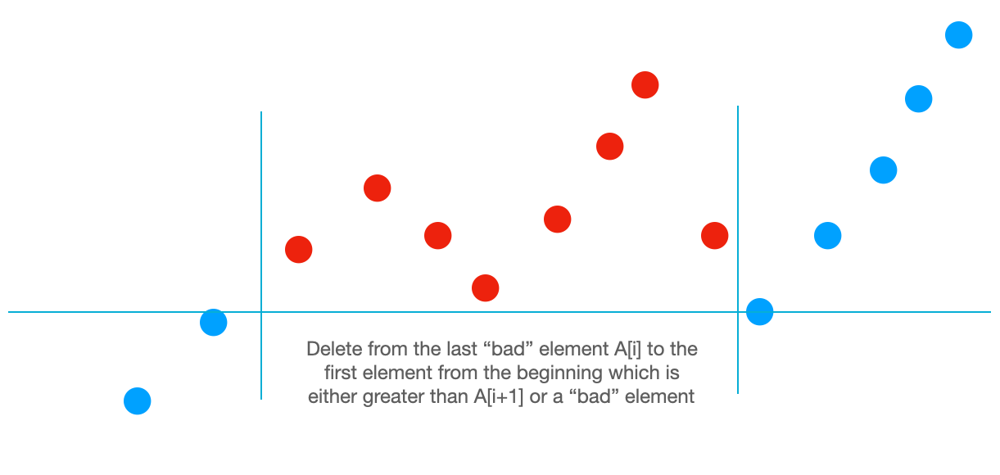
**Oops, sorry my original approach below is incorrect because it can't pass [1,2,3,10,0,7,8,9]. :(**

Case 1



We scan from left to right and stop at the first "bad" node where A[i] < A[i -1]. Now we have an option to delete all the nodes from A[i] to A[j] where A[j] is the first node from the end that is either smaller than A[i - 1] or A[j] > A[j + 1].

Case 2



We scan from right to left and stop at the first "bad" node where A[i] > A[i + 1]. Now we have an option to delete all the nodes from A[i] to A[j] where A[j] is the first node from the beginning that is either greater than A[i + 1] or A[j] < A[j - 1].

// OJ: https://leetcode.com/contest/biweekly-contest-34/problems/shortest-subarray-to-be-removed-to-make-array-sorted/

// Author: github.com/lzl124631x

// Time: O(N)

// Space: O(1)

class Solution {

public:

int findLengthOfShortestSubarray(vector<int>& A) {

int N = A.size(), ans = N - 1, i;

for (i = 1; i < N; ++i) {

if (A[i] >= A[i - 1]) continue;

int j = N - 1;

while (j > i && A[j] >= A[i - 1] && (j == N - 1 || A[j] <= A[j + 1])) --j;

ans = min(ans, j - i + 1);

break;

}

if (i == N) return 0;

for (i = N - 2; i >= 0; --i) {

if (A[i] <= A[i + 1]) continue;

int j = 0;

while (j < i && A[j] <= A[i + 1] && (j == 0 || A[j] >= A[j - 1])) ++j;

ans = min(ans, i - j + 1);

break;

}

return ans;

}

};