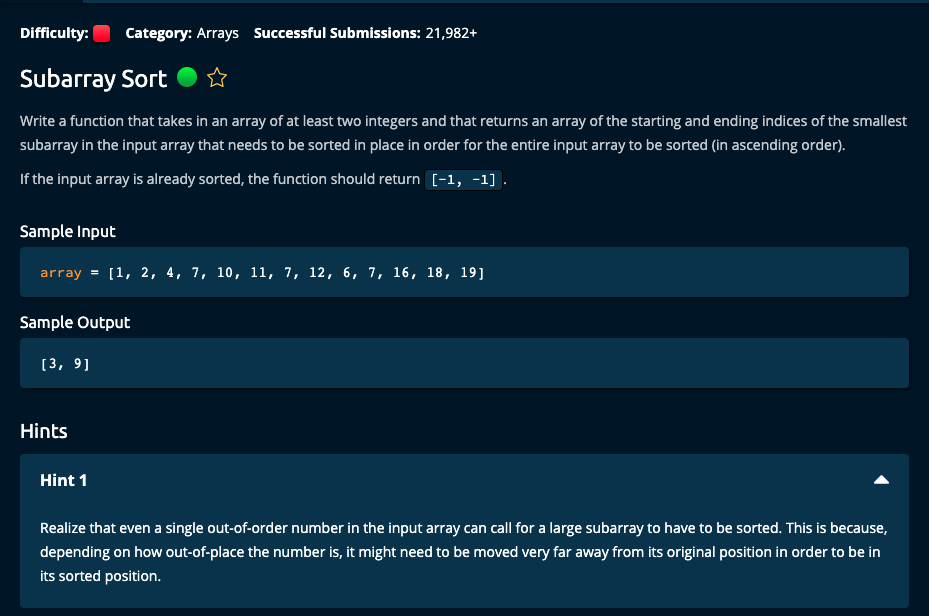
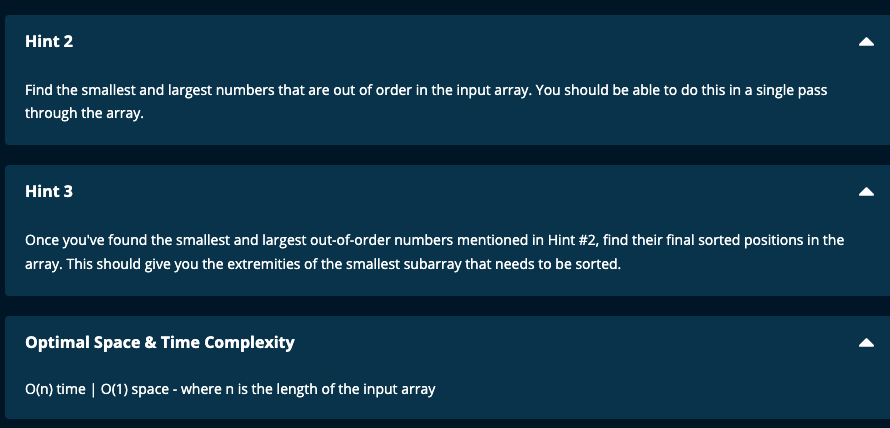
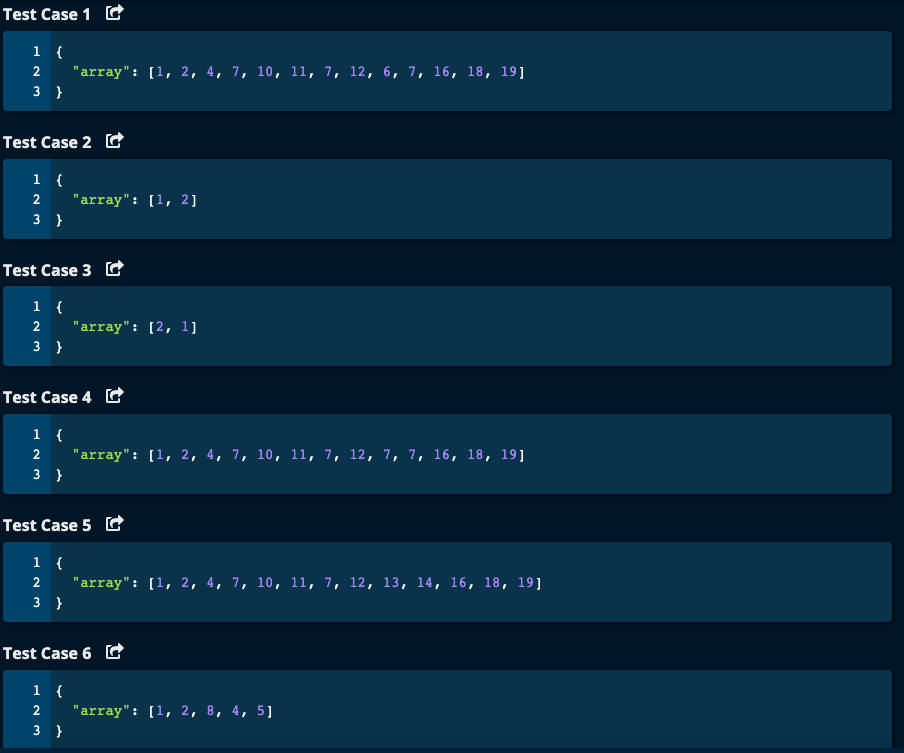
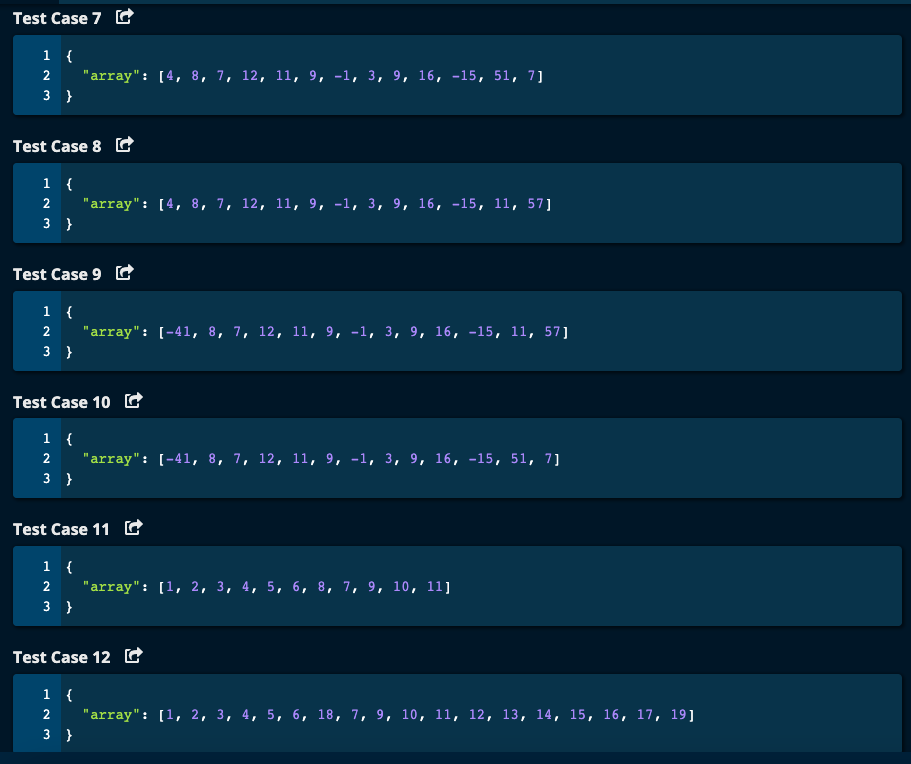
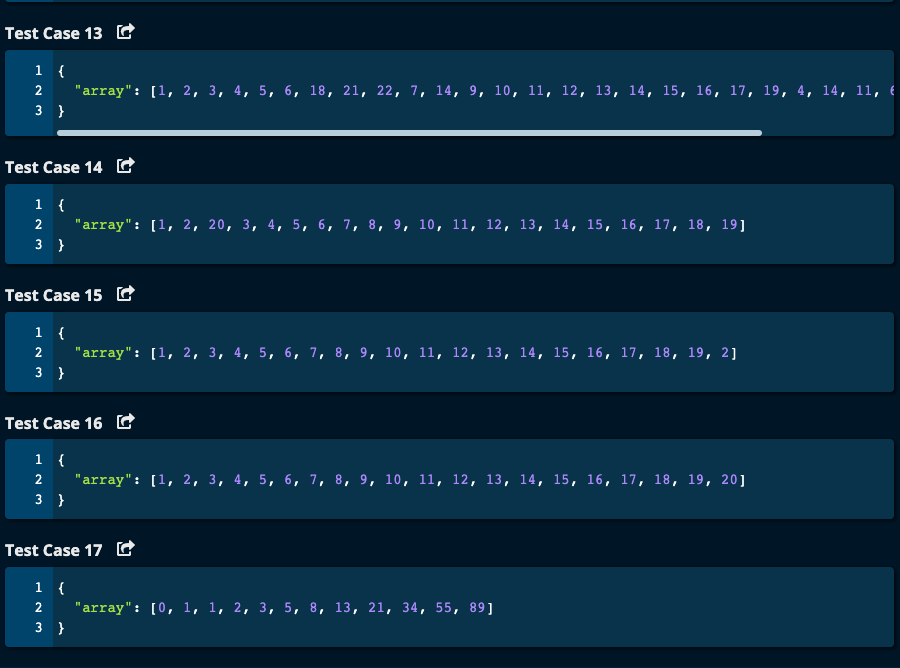
Subarray Sort (Hard)











My Solution:

Solution:

# My Solution -- O(n) Time Complexity | O(1) Space Complexity

def subarraySort(array):

if len(array) == 2:

if array[0] <= array[1]:

return [-1, -1]

elif array[0] > array[1]:

return [0, 1]

smallest = float("inf") # smallest element in unsorted subarray

largest = float("-inf") # largest element in unsorted subarray

for i in range(1, len(array)):

if (i == 0 and array[i] > array[i + 1]) or \

(i == len(array) - 1 and array[i] < array[i - 1]) or \

(i > 0 and i < len(array) - 1 and (array[i - 1] > array[i] or array[i] > array[i + 1])):

smallest = min(smallest, array[i])

largest = max(largest, array[i])

if smallest == float("inf"):

return [-1, -1]

# Search the array to get the index of smallest and largest in unsortedList

left = 0

smallestIdx = 0

largestIdx = len(array) - 1

while left < len(array) and smallest >= array[left]:

left += 1

smallestIdx = left

if left == len(array):

smallestIdx = len(array) - 1

right = len(array) - 1

while right > -1 and largest <= array[right]:

right -= 1

largestIdx = right

return [smallestIdx, largestIdx]

JJ Notes:

1. If length of the array is 1, and if the elements are in sorted order return [-1, -1].

However, if they are not sorted order return [0, 1].

1. Initialize smallest element in unsorted subarray to float(“inf”) and largest element in unsorted subarray to float(“-inf”).
2. Now, iterate through the array. If it is the first element in the array, compare with the next element. If it is the last element in the array compare with the last but one element. If it is not the first nor the last element, then compare it with its previous and next neighbor. If the element in question is not in order, then update smallest and largest.
3. After traversing the array, if smallest is still float(“inf”) then array is already in order and so return [-1, -1].
4. If there is a subarray that has unsorted elements, then search the whole array to get the index of the smallest and largest unsorted elements.
5. For this, initialize the left pointer to 0, and the smallest index i.e. smallestIdx to 0.

While the left pointer is less than array length, and smallest is greater than the array element at index of left pointer, we will keep moving the left pointer to the right (increment left pointer by 1) since it is already sorted so far.

When we break out of the while loop, update smallestIdx to left.

1. Now, initialize right pointer to the length of array -1 (i.e. last element of the array) and keep moving left i.e. decrement the right pointer by 1 until we hit the largest element of the unsorted subarray. When we break out of the while loop, update largestIdex to right.
2. Return [smallestIdx, largestIdx].

# Algoexpert Solution - O(n) Time | O(1) space

def subarraySort(array):

minOutOfOrder = float("inf")

maxOutOfOrder = float("-inf")

for i in range(len(array)):

num = array[i]

if isOutOfOrder(i, num, array):

minOutOfOrder = min(minOutOfOrder, num)

maxOutOfOrder = max(maxOutOfOrder, num)

if minOutOfOrder == float("inf"):

return [-1, -1]

subarrayLeftIdx = 0

while minOutOfOrder >= array[subarrayLeftIdx]:

subarrayLeftIdx += 1

subarrayRightIdx = len(array) - 1

while maxOutOfOrder <= array[subarrayRightIdx]:

subarrayLeftIdx -= 1

return [subarrayLeftIdx, subarrayRightIdx]

def isOutOfOrder(i, num, array):

if i == 0:

return num > array[i + 1]

if i == len(array) - 1:

return num < array[i - 1]

return num > array[i + 1] or num < array[i - 1]

JJ Notes:  
  
1. Basically the same steps, but this has a function isOutOfOrder to check whether the element at index i in the array is out of order (i.e. not sorted).