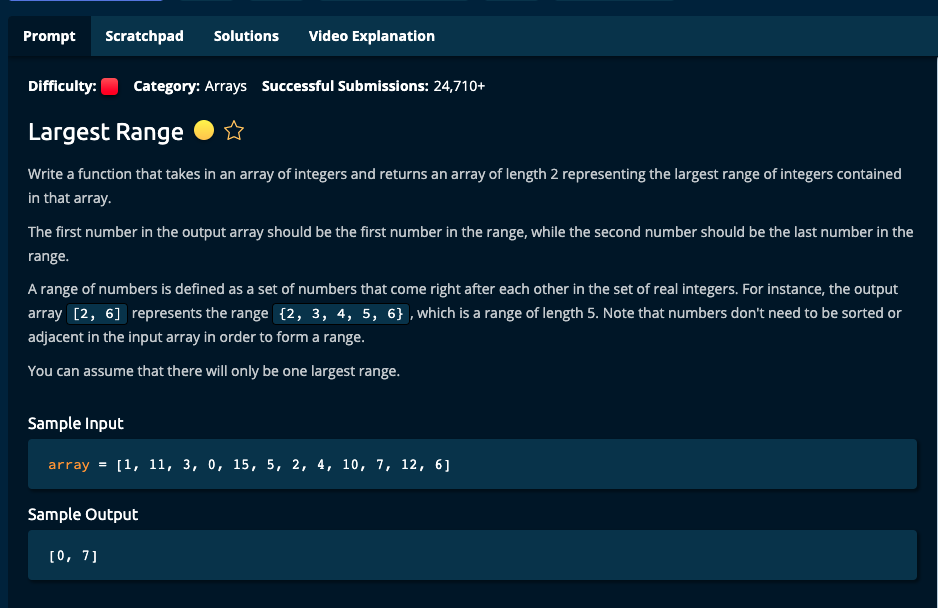
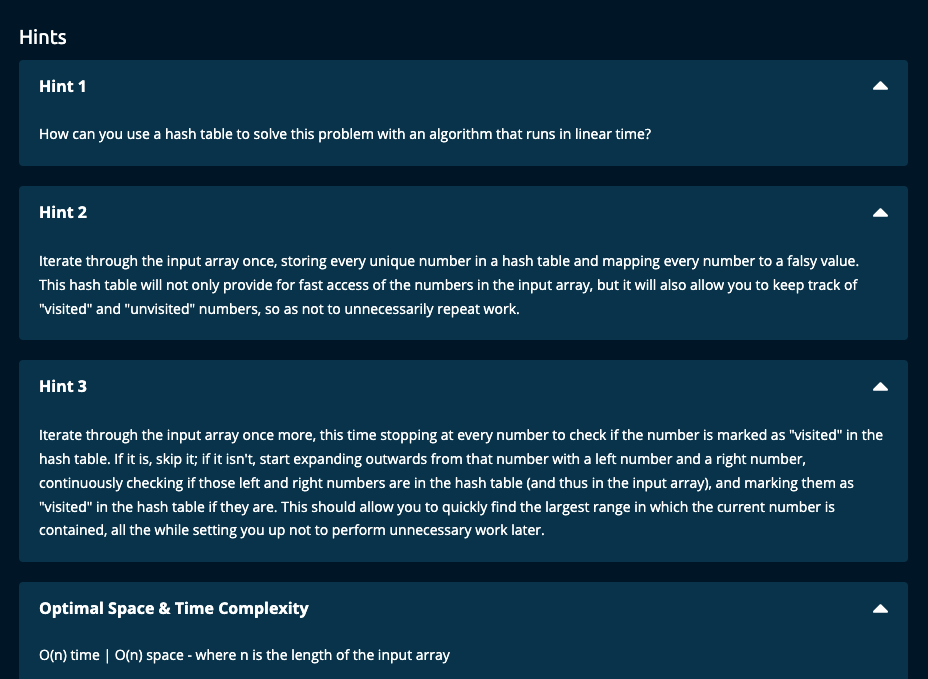
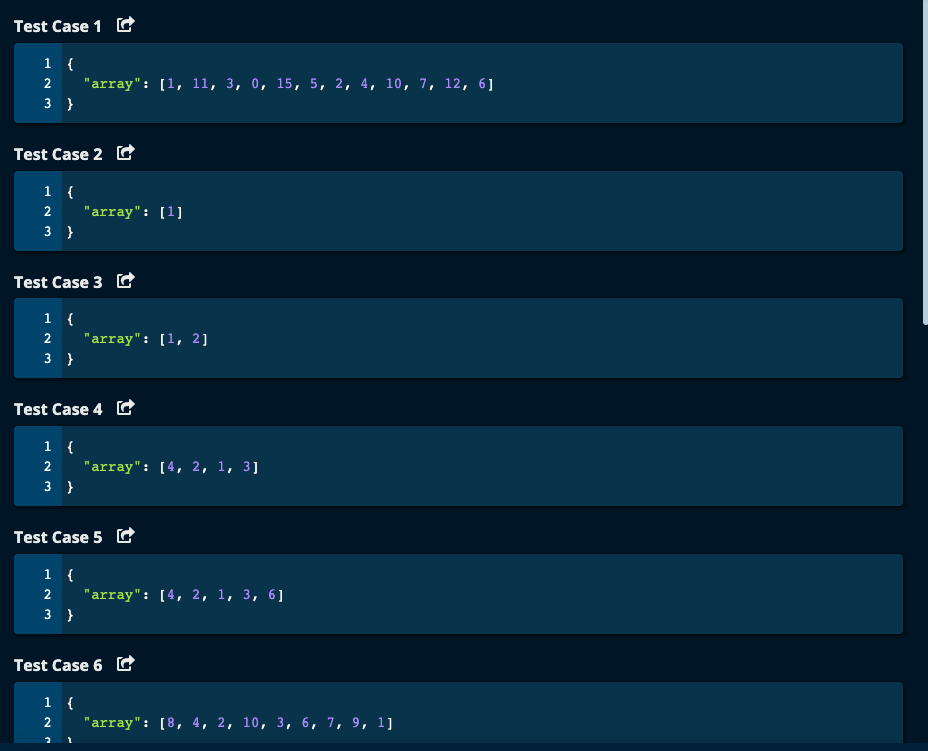
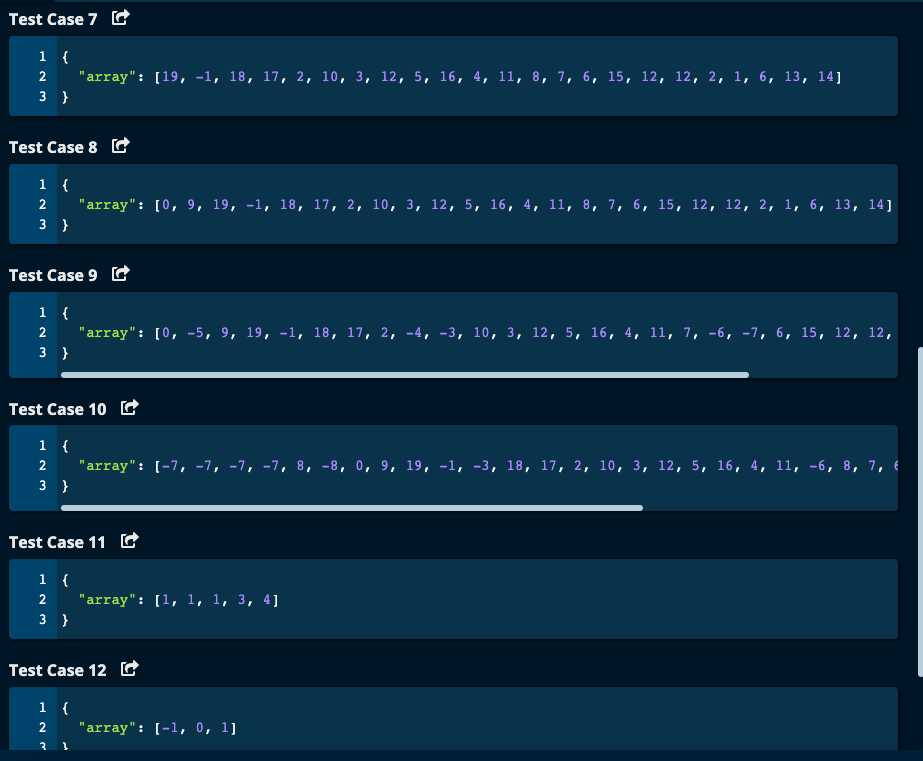
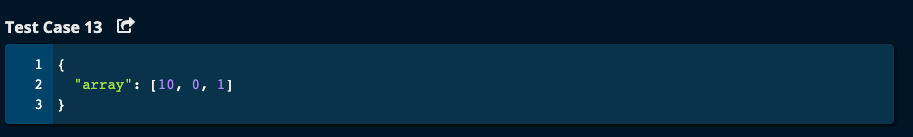
Largest Range (Hard)











My Solutions:

Solution 1: Using Sorting – O(log(n)) Time | O(n) space

# Since we are sorting, time complexity is O(nlog(n)).

# My Solution with sorting -- O(n) time | O(n) space

def largestRange(array):

sortedArray = sorted(array)

print("sortedArray = ", sortedArray)

results = []

temparray = []

temparray.append(sortedArray[0])

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

if sortedArray[i] == sortedArray[i - 1] + 1:

temparray.append(sortedArray[i])

else:

results.append(temparray)

temparray = []

temparray.append(sortedArray[i])

print("i = ", i, "temparray = ", temparray, "results = ", results)

results.append(temparray)

print("final results = ", results)

maxLength = len(results[1])

startEle = results[0][0]

endEle = results[0][-1]

for subarray in results:

subarrayLength = len(subarray)

if subarrayLength > maxLength:

maxLength = subarrayLength

startEle = subarray[0]

endEle = subarray[-1]

return [startEle, endEle]

JJ Notes:

1. Sort the array and call it sortedArray.

2. Initialize result to an empty list to hold the list of ranges.

3. Initialize a temparray to hold the list of the element in that specific range only.

4. Iterate through the array and if the current element is 1 more than the previous element, expand the range by adding the element to temparray. Otherwise, append the temparray to result, and reinitialize the temparray to an empty list and add the current element to it.

5. After exiting the for loop, what ever is left in the temparray, append it to result array.

6. Now iterate through result array to find the longest subarray.

7. Return the longest subarray beginning and end elements as a list.

Solution 2 – Using Hash table (dictionary). – based on algoexpert solution

Time Complexity = O(n) as we do not sort and traverse the array just once to store the elements in the dictionary

My Solution based on Algoexpert conceptual overview using Hash table -- O(n) time | O(n) space

def largestRange(array):

mydict = {}

# Create the hash table by traversing the array

for num in array:

mydict[num] = True # set the value initially to True

maxLen = 0

# Traverse the array and get the bestRange

bestRange = []

for num in array:

if mydict[num]:

mydict[num] = False # this value has now been visited and is set to False

left = num - 1

right = num + 1

curLen = 1

while left in mydict.keys():

mydict[left] = False

curLen += 1

left -= 1

while right in mydict.keys():

mydict[right] = False

curLen += 1

right += 1

if curLen > maxLen:

maxLen = curLen

bestRange = [left + 1, right - 1]

return bestRange

JJ Notes:

1. Traverse through the array once and store the elements as dictionary keys with value True representing not visited.
2. Initialize maxLen to 0 and bestRange to an empty list.
3. Traverse the array again to get the best range. For the element in the array, go left to the next integer as far as possible, and go right to the next integer as far as possible. For going left and going right, the elements are retrieved from the dictionary in O(1) time and update the length of the range as every integer is added to the range. Now compare this length of this range to the maxLen that we have so far. If current length is more update the maxLen and bestRange.
4. Return the bestRange.

Algoexpert Solution has the same logic as my solution 2 but has continue. 2 out of 13 cases failed as a result. O(n) time and O(n) space using a dictionary as no sorting is done.

