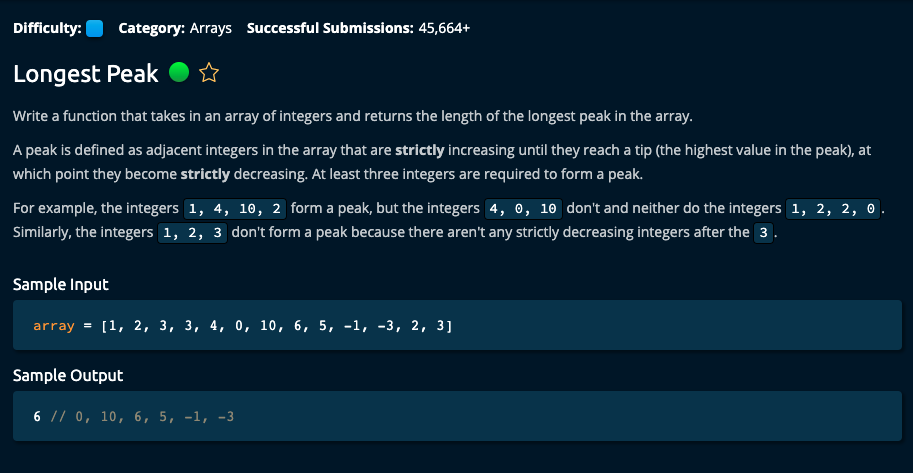
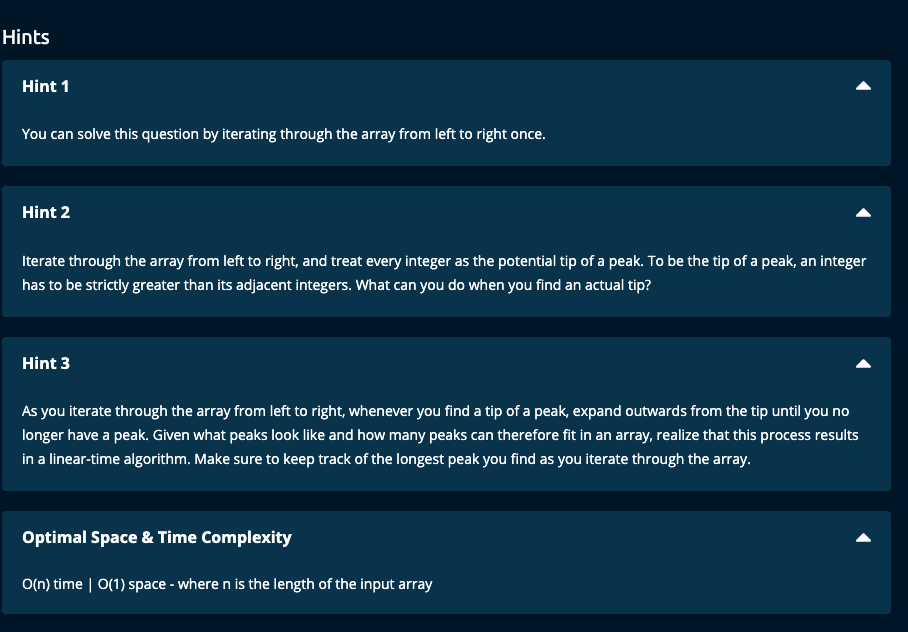
Longest Peak (Medium)

------------------------------------------------



My Solution:

Solution 1:

def longestPeak(array):

n = len(array)

if n < 3:

return 0

longestPeak = 0

for i in range(1, n-1): # Start at the second value and end at the last but one value

if array[i] > array[i - 1] and array[i] > array[i + 1]: # compare value to adjacent ones

peak = i

left = i - 1

right = i + 1

currentPeakLength = 3

# See how far we can go to the left of the peak

while (left > 0) and array[left - 1] < array[left]:

left -= 1

currentPeakLength += 1

# See how far we can go to the right of the peak

while (right < n - 1) and array[right + 1] < array[right]:

right += 1

currentPeakLength += 1

longestPeak = max(longestPeak, currentPeakLength)

return longestPeak

--------------------------------

JJ Notes:

1. Let n be the length of the array. If n is less than 3, then return 0 since we need at least 3 elements to have a peak.
2. Initialize longestpeak to 0.
3. Compare each element to its adjacent elements and if it is greater than the adjacent elements, it forms a peak. So identify the peak.
4. For each peak, find how far we can go to the left and how far we can go to the right to determine the length of this peak, i.e. currentPeakLength.
5. Find the max of longestPeak and currentPeakLength.
6. Return longestPeak.

Time Complexity: O(n)

Space Complexity: O(1)

Solution 2:

def longestPeak(array):

n = len(array)

if n < 3:

return 0

longestPeak = 0

i = 1

while i <= n - 2:

if array[i] > array[i - 1] and array[i] > array[i + 1]:

peak = i

left = i - 1

right = i + 1

currentPeakLength = 3

while (left > 0) and array[left - 1] < array[left]:

left -= 1

currentPeakLength += 1

while (right < n - 1) and array[right + 1] < array[right]:

right += 1

currentPeakLength += 1

longestPeak = max(longestPeak, currentPeakLength)

i = right

else:

i += 1

return longestPeak

--------------------------------

JJ Notes:

1. Let n be the length of the array. If n is less than 3, then return 0 since we need at least 3 elements to have a peak.
2. Initialize longestpeak to 0.
3. Initialize index i of the array to 1. We are starting from the element at index 1 and going up to the last but one element of the array to identify a peak (since a peak requires at least 3 points).
4. Compare the element at index I to its adjacent elements and if it is greater than the adjacent elements, it forms a peak. So identify the peak.
5. For each peak, find how far we can go to the left and how far we can go to the right to determine the length of this peak, i.e. currentPeakLength.
6. Find the max of longestPeak and currentPeakLength.
7. Return longestPeak.
8. Here we are using a while loop and once we have a peak, we start looking for the next peak only at the next element after the rightmost value of the peak.

Time Complexity: O(n)

Space Complexity: O(1)

Solution #3:

def longestPeak(array):

n = len(array)

if n < 3:

return 0

longestPeak = 0

i = 1

while i <= n - 2: # Start from second value and go up to the last but one

if array[i] > array[i - 1] and array[i] > array[i + 1]:

peak = i

left = i - 1

right = i + 1

currentPeakLength = 0

while (left > 0) and array[left - 1] < array[left]:

left -= 1

while (right < n - 1) and array[right + 1] < array[right]:

right += 1

currentPeakLength = right - left + 1

longestPeak = max(longestPeak, currentPeakLength)

i = right + 1

else:

i += 1

return longestPeak

JJ Notes: Almost the same as Solution #2, except for currentPeakLength = right – left + 1

Say array = [2, 1, 5,3, 2, 3, 4], consider peak at 5. Peak is from [1, 5, 3, 2] and currentPeakLength = 4. Index at peak beginning = 1 and index at peak end = 4.

Length = 4 – 1 + 1 = 4

# Algoexpert solution -- O(n) Time | O(1) Space

def longestPeak(array):

longestPeakLength = 0

i = 1

while i < len(array) - 1: # go only up to last but one value for peak

isPeak = array[i - 1] < array[i] and array[i] > array[i + 1]

if not isPeak:

i += 1

continue

leftIdx = i - 2

while leftIdx >= 0 and array[leftIdx] < array[leftIdx + 1]:

leftIdx -=1

rightIdx = i + 2

while rightIdx < len(array) and array[rightIdx] < array[rightIdx - 1]:

rightIdx += 1

currentPeakLength = rightIdx - leftIdx - 1

longestPeakLength = max(longestPeakLength, currentPeakLength)

i = rightIdx

return longestPeakLength

Almost same logic as mine.