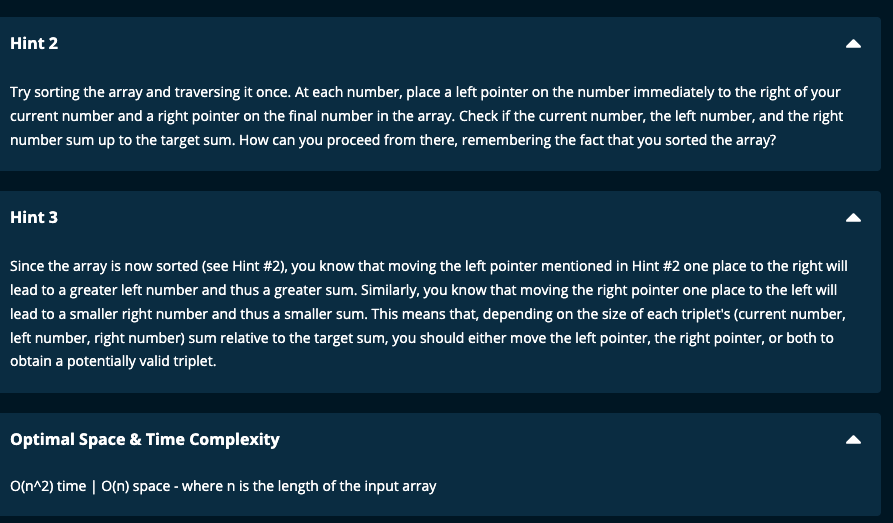
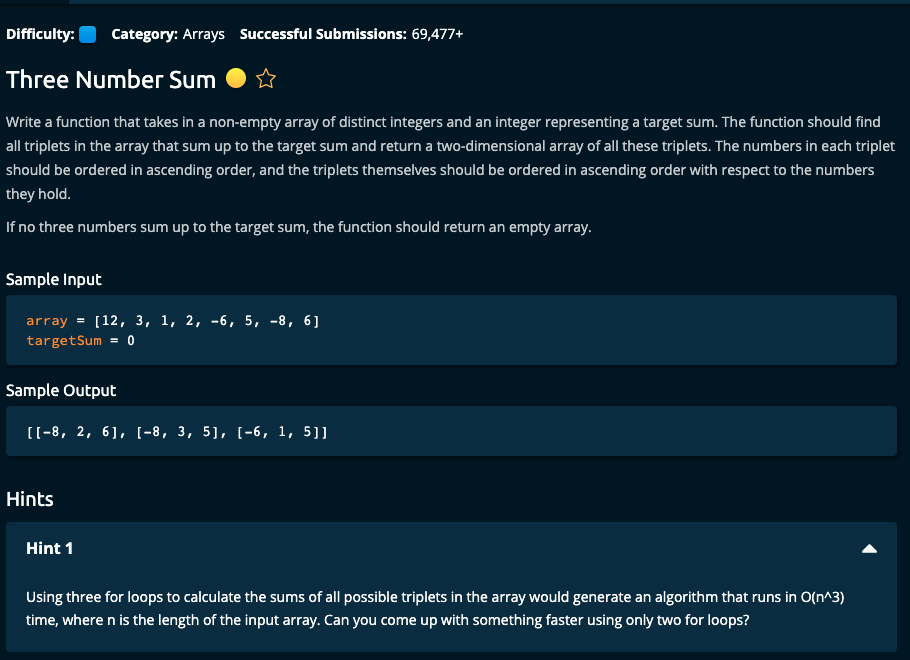
Three Number Sum. (Medium)

<https://www.algoexpert.io/questions/three-number-sum>



My Solution 1:

def threeNumberSum(array, targetSum):

array = sorted(array)

n = len(array)

result = []

n = len(array)

for i in range(n - 2):

target = targetSum - array[i]

left = i + 1

right = n - 1

while left < right:

if array[left] + array[right] == target:

result.append([array[i], array[left], array[right]])

left += 1

right -= 1

elif array[left] + array[right] >= target:

right -= 1

else:

left += 1

return result

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

My Solution 2:

def threeNumberSum(array, targetSum):

array = sorted(array)

result = []

for i in range(len(array) - 2):

target = targetSum - array[i]

res = twoSum(array, target, i)

result += res

return result

def twoSum(array, target, i):

left = i + 1

right = len(array) - 1

res = []

while left < right:

if array[left] + array[right] == target:

res.append([array[i], array[left], array[right]])

left += 1

right -= 1

elif array[left] + array[right] >= target:

right -= 1

else:

left += 1

return res

====================================

Notes:

1. Sort the array.
2. Have a for loop and run through the array except for the last two numbers.
3. Find out the target for the other two numbers (target = targetSum – array[i])
4. Have a left and right pointer and go through the array from left and right towards each other as in Two Sum.
5. In my second solution, I have made TwoSum into a separate function.
6. If the target is equal to (targetSum – array[i]) then advance both pointers towards each other after appending the triplet list to the result array. If the target is less, then move the left pointer to the right (i.e. increment left pointer), and if target is more, then move the right pointer to the left (i.e. decrement right pointer)
7. Time complexity: O(n^2) |. Space complexity = O(n)

Sorting in the array takes O(n.log(n)) which is less than O(n^2)

Algoexpert Solution:

def threeNumberSum(array, targetSum):

array.sort()

triplets = []

for i in range(len(array)- 2):

left = i + 1

right = len(array) - 1

while left < right:

currentSum = array[i] + array[left] + array[right]

if currentSum == targetSum:

triplets.append([array[i], array[left], array[right]])

left += 1

right -= 1

elif currentSum > targetSum:

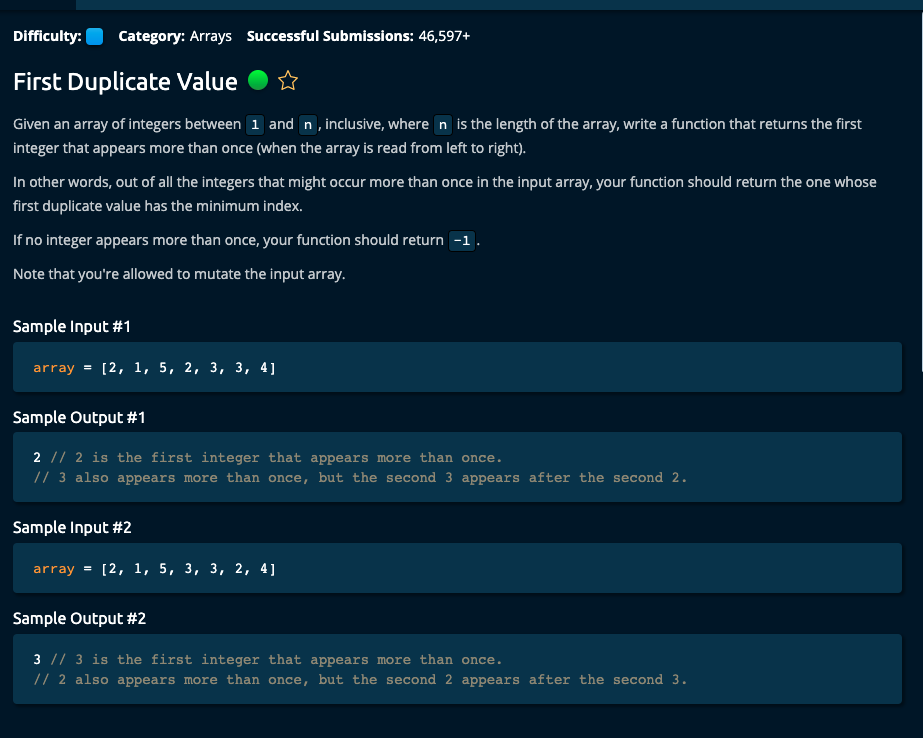
right -= 1

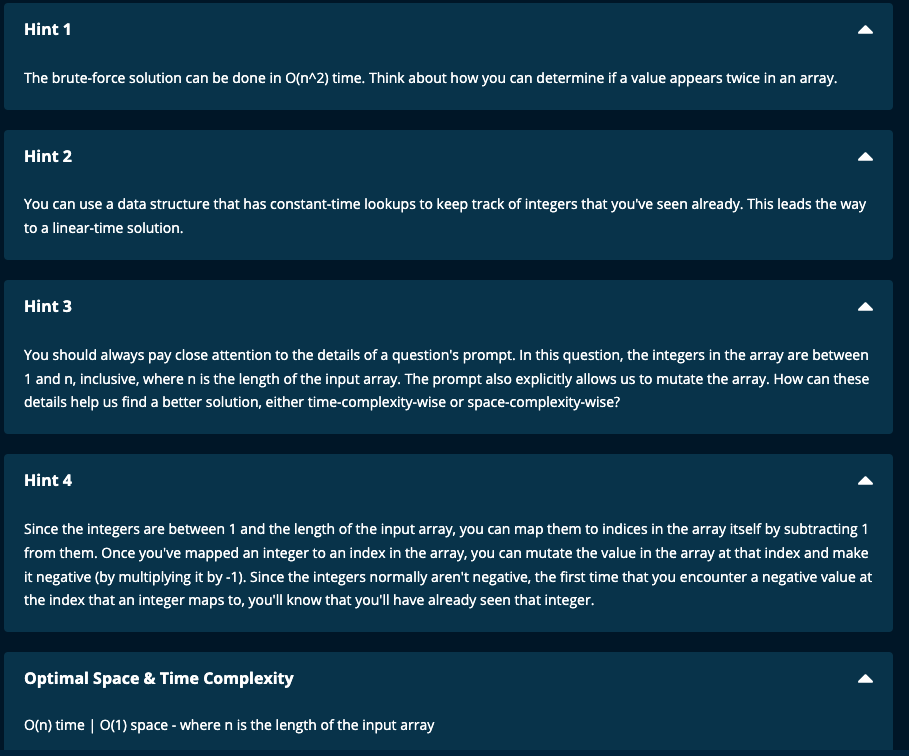
elif currentSum < targetSum:

left += 1

return triplets

First Duplicate Value (Medium)





My Solutions:

Solution 1:

def firstDuplicateValue(array):

myDict = {}

for num in array:

if num not in myDict.keys():

myDict[num] = 1

else:

return num

return -1

JJ Notes:

1. Initialize an empty dictionary.
2. Traverse the array using a for loop. If the element is not in the dictionary add it as a key with value 1. If the element is already in the dictionary return the element.
3. If nothing has been returned so far after traversing the array, then return -1.

Time Complexity: O(n) for traversing the array (also lookup is O(1) time)

Space Complexity: O(n) for seen which can contain all the elements of the array.

====================================

Solution 2

def firstDuplicateValue(array):

seen = set()

for element in array:

if element not in seen:

seen.add(element)

else:

return element

return -1

JJ Notes:

1. Initialize an empty set as seen.
2. Traverse through the array using a for loop. If the element is not in seen, add the element to seen. If the element already exists in seen, then return the element.
3. Finally, if no element has been returned so far, then return -1.

Time Complexity: O(n) for traversing the array (also lookup is O(1) time)

Space Complexity: O(n) using seen which can contain all the elements of the array

================================================================

Algoexpert Solutions:

Solution1 – Brute Force – Time Complexity : O(n^2), Space complexity = O(1)

Time complexity n + (n – 1) + (n -2) + … + 1 = n (n+1)/2 and so this is O(n^2)

def firstDuplicateValue(array):

minimumSecondIndex = len(array)

for i in range(len(array)):

value = array[i]

for j in range(I + 1, len(array)):

valueToCompare = array[j]

if value == valueToCompare:

minimumSecondIndex = min(minimumSecondIndex, j)

if minimumSecondIndex == len(array):

return -1

return array[minimumSecondIndex]

Solution 2: Time Complexity : O(n), Space complexity = O(n) since seen can contain the whole array.

def firstDuplicateValue(array):

seen = set()

for element in array:

if element in seen:

return element

seen.add(element)

return -1

Solution 3: Time Complexity : O(n), Space complexity = O(1) since we are mutating the array and also using the fact the array of length n contains elements from 1 .. n.

The index of the array is from 0 to n -1.

We are mapping each element to its index by multiplying it by -1.

def firstDuplicateValue(array):

for value in array:

absvalue = abs(value)

if array[absvalue - 1] < 0:

return absvalue

array[absvalue - 1] \*= -1

return -1

Walkthrough

array = [2, 1, 5, 3, 2, 3, 4]

This array is of length = 7 and elements in the array are in the range 1..7.

Now we will map each element to its index for which we need to subtract 1 since index is from 0..6 and we will multiply the element at that index by -1 to signify that we have already seen that number.

Traverse the array.

Step 1: We see 2. Take the absolute value of 2 and subtract 1 to get its index. This is 1.

Check if array[1] < 0. No, since array = [2, 1, 5, 3, 2, 3, 4]

Go to array[1] and multiply this by -1.

array = [2, -1, 5, 3, 2, 3, 4]

Step 2: We see 1. Take the absolute value of 1and subtract 1 to get its index. This is 0.

Check if array[0] < 0. No, since array = [2, -1, 5, 3, 2, 3, 4]

Go to array[0] and multiply this by -1.

array = [-2, -1, 5, 3, 2, 3, 4]

Step 3: We see 5. Take the absolute value of 5 and subtract 1 to get its index. This is 4.

Check if array[4] < 0. No, since array = [-2, -1, 5, 3, 2, 3, 4]

Go to array[4] and multiply this by -1.

array = [-2, -1, 5, 3, -2, 3, 4]

Step 4: We see 3. Take the absolute value of 3 and subtract 1 to get its index. This is 2.

Check if array[2] < 0. No, since array = [-2, -1, 5, 3, -2, 3, 4]

Go to array[2] and multiply this by -1.

array = [-2, -1, -5, 3, -2, 3, 4]

Step 5: We see -2. Take the absolute value of -2 and subtract 1 to get its index. This is 1.

Check if array[1] < 0. Yes, since array = [-2, -1, -5, 3, -2, 3, 4]

Now return absolute value of -2 = 2.