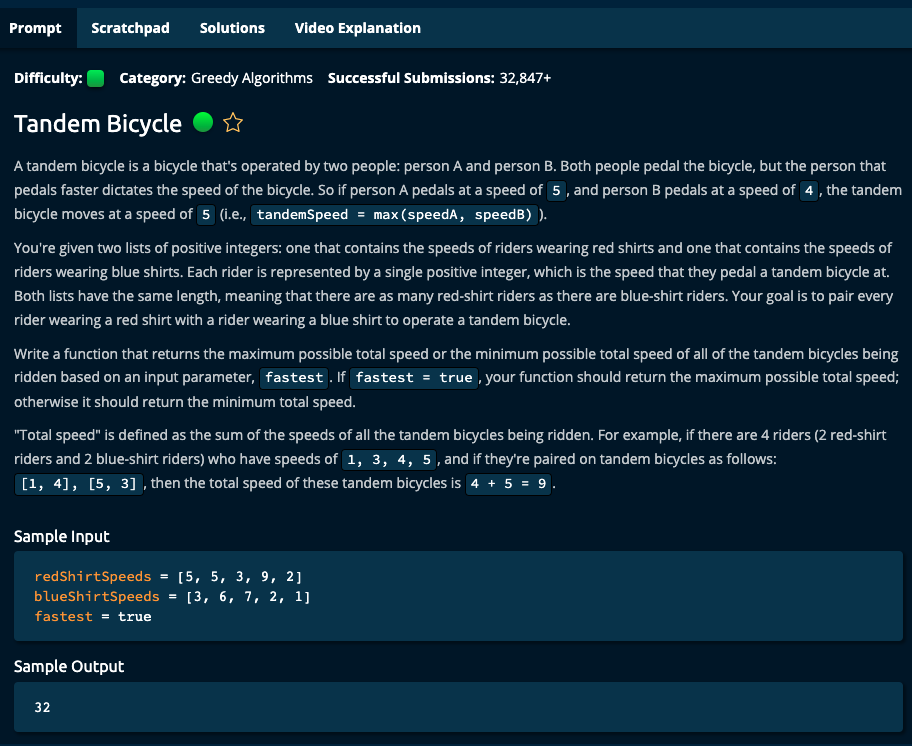
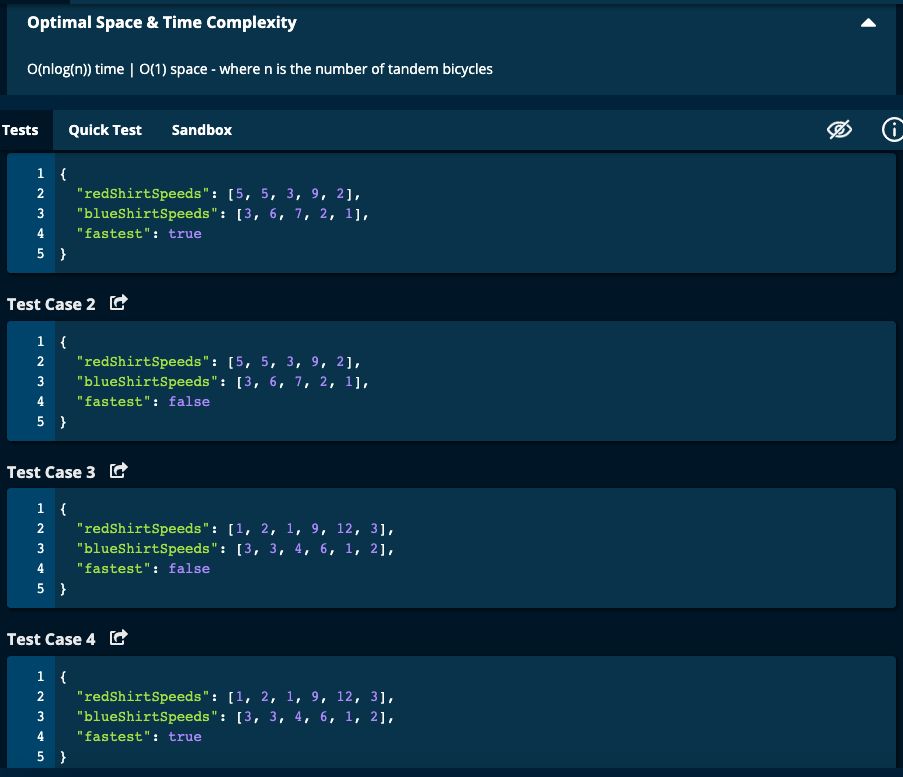
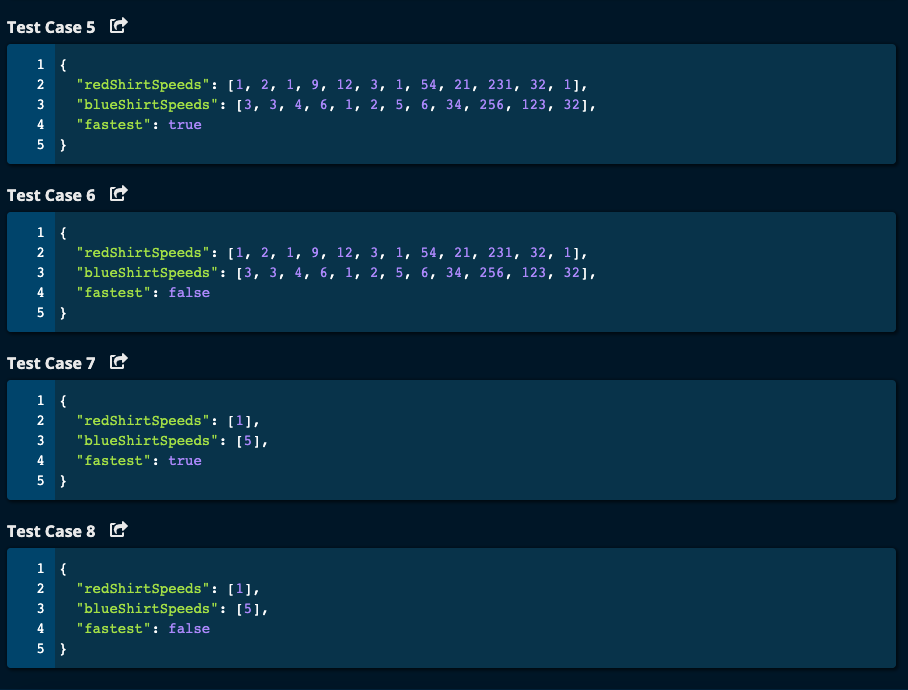
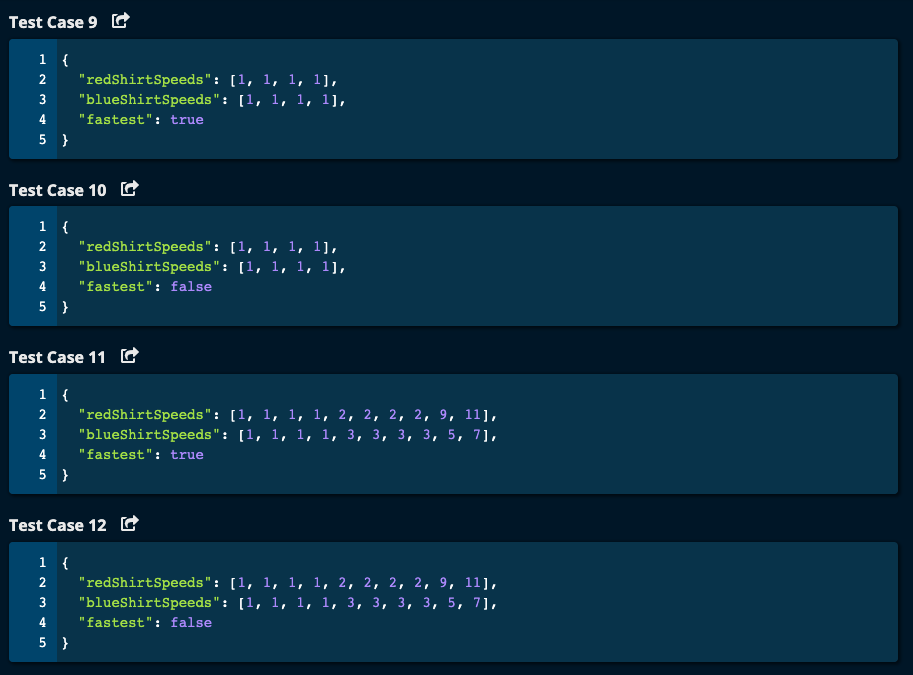
Tandem Bicycle (Easy)

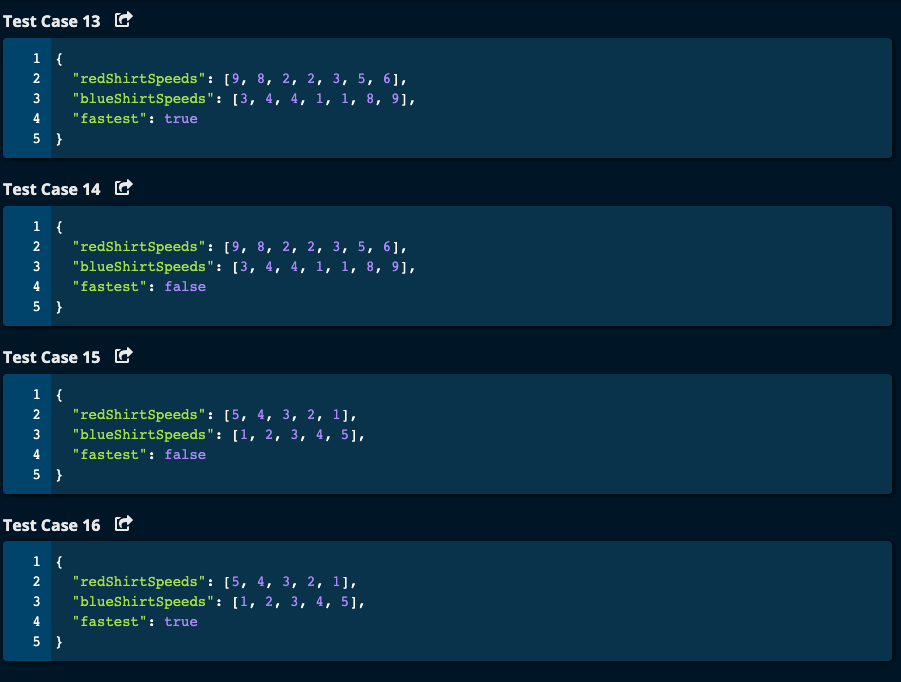


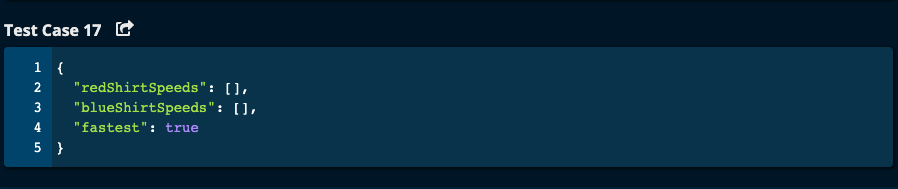












My Solution: O(nlog(n)) time | O(1) space

def tandemBicycle(redShirtSpeeds, blueShirtSpeeds, fastest):

redShirtSpeeds = sorted(redShirtSpeeds)

blueShirtSpeeds = sorted(blueShirtSpeeds)

totalSpeed = 0

n = len(redShirtSpeeds)

if fastest: # fastest is True

for i in range(n):

totalSpeed += max(redShirtSpeeds[i], blueShirtSpeeds[n - i - 1])

else: # fastest is False

for i in range(n):

totalSpeed += max(redShirtSpeeds[i], blueShirtSpeeds[i])

return totalSpeed

JJNotes:

1. Sort both lists.
2. Initialize totalSpeed to 0.
3. If fastest is True then traverse both speed array by pairing the smallest redShirtSpeed with the largest blueShirtSpeed and the largest redShirtSpeed with the smallest blueshirtSpeed.

That is, redShirtSpeed is ascending and blueshirtSpeed is descending. This way we will achieve the maximum speed.

1. If fastest is False, then traverse both speeds by pairing the smallest redShirtSpeed with the smallest blueShirtSpeed and the largest redShirtSpeed with the largest blueShirtSpeed. That is, both speeds must be ascending. This way, we will get the minimum speed.
2. Return totalSpeed.
3. Time Complexity is O(nlog(n)) for sorting and O(n) for traversing the array which gives O(nlog(n)). Space complexity is O(1) since we are sorting in place and we have only one variable totalSpeed.

Algoexpert Solution:

# Algoexpert Solution

def tandemBicycle(redShirtSpeeds, blueShirtSpeeds, fastest):

redShirtSpeeds.sort()

blueShirtSpeeds.sort()

if fastest:

reverseArrayInPlace(redShirtSpeeds)

totalSpeed = 0

for idx in range(len(redShirtSpeeds)):

rider1 = redShirtSpeeds[idx]

rider2 = blueShirtSpeeds[idx]

totalSpeed += max(rider1, rider2)

return totalSpeed

def reverseArrayInPlace(array):

start = 0

end = len(array) - 1

while start < end:

array[start], array[end] = array[end], array[start]

start += 1

end -= 1

Algoexpert Solution Notes:

1. Sort both arrays
2. If fastest is True, then reverse one of the arrays say redShirtSpeeds. This is done in placeby swapping the first and last elements in the function reverseArrayInPlace () and then start pointer and end pointer will move towards the center until they meet or cross.
3. Initialize totalSpeed to 0.
4. Traverse through both arrays. If fastest is True, then the smallest in one array will be paired with the largest in the other array and vice versa. If fastest is False, then we want to pair the smallest in one array with the smallest in the other array, and then largest in the first array with the largest in the second array.
5. Time Complexity is O(nlog(n)) for sorting and O(n) for traversing the array which gives O(nlog(n)). Space complexity is O(1) since we are sorting in place and we have only one variable totalSpeed.