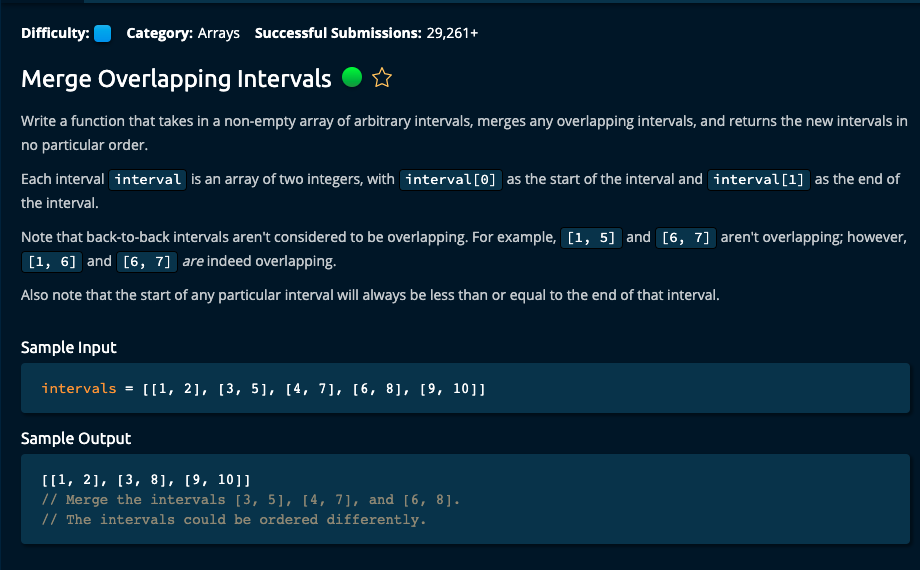
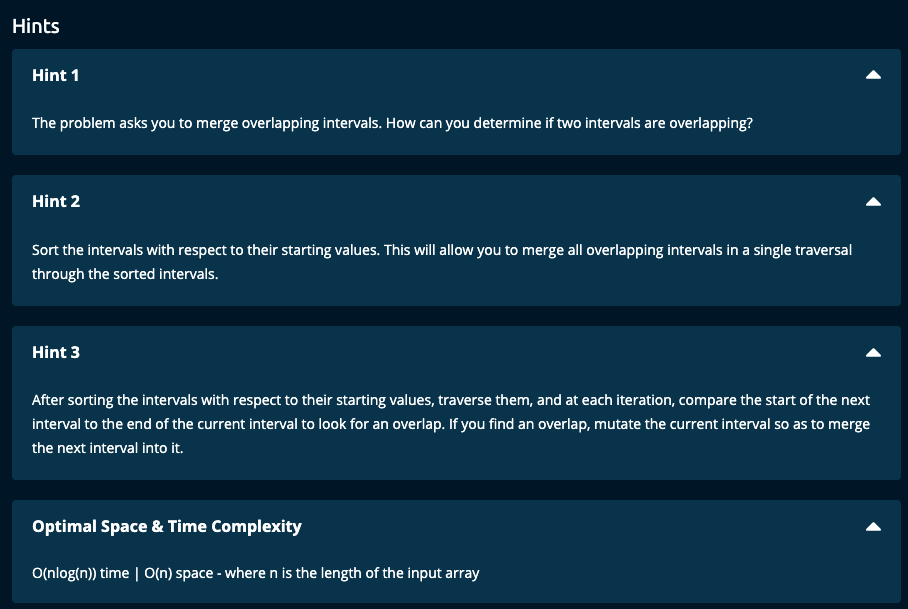
Merged Overlapping Intervals (Medium)





My Solution:

# My Solution using a stack -- O(log(n)) time | O(n) space

def mergeOverlappingIntervals(intervals):

sortedIntervals = sorted(intervals, key = lambda x: x[0])

stack = [sortedIntervals[0]]

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

if sortedIntervals[i][0] > stack[-1][1]:

stack.append(sortedIntervals[i])

else:

stack[-1][1] = max(stack[-1][1], sortedIntervals[i][1])

return stack

JJ Notes:

1. Sort the intervals by the starting time of each interval, i.e. sortedIntervals. This is done in O(nlog(n)) time.
2. Use a stack implementation. Initialize the stack to the first interval in sortedIntervals.
3. Traverse through the sortedIntervals array. If the start of the current interval is greater than the end of the last interval in the stack, there is no overlap. So, append the current interval to the stack. If not, modify the end of the last interval in the stack to be the maximum of the end of the current interval and the existing end of the last interval in the stack.

# Algoexpert solution -- O(nlog(n)) time | O(n) space

def mergeOverlappingIntervals(intervals):

sortedIntervals = sorted(intervals, key = lambda x: x[0])

mergedIntervals = []

currentInterval = sortedIntervals[0]

mergedIntervals.append(currentInterval)

for nextInterval in sortedIntervals:

\_, currentIntervalEnd = currentInterval

nextIntervalStart, nextIntervalEnd = nextInterval

if currentIntervalEnd >= nextIntervalStart:

currentInterval[1] = max(currentIntervalEnd, nextIntervalEnd)

else:

currentInterval = nextInterval

mergedIntervals.append(currentInterval)

return mergedIntervals