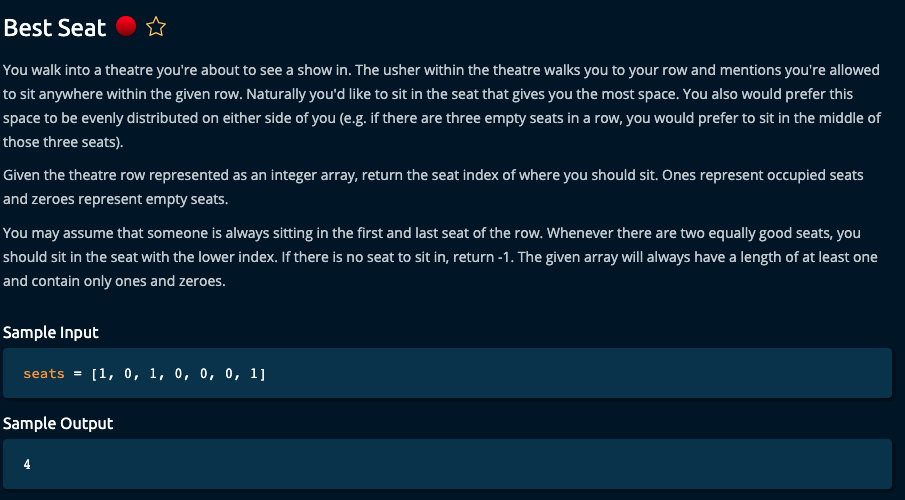
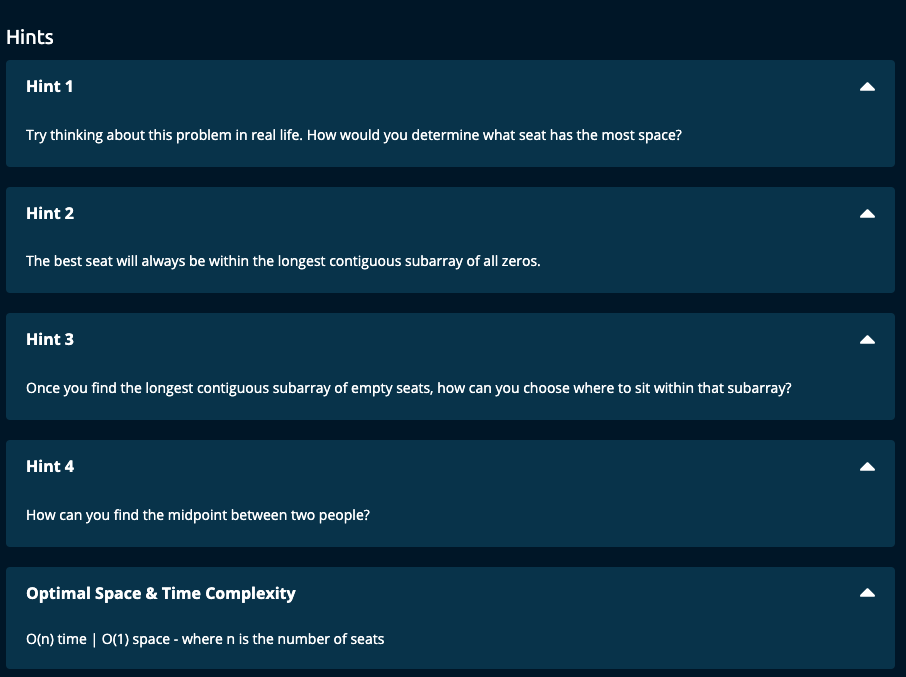
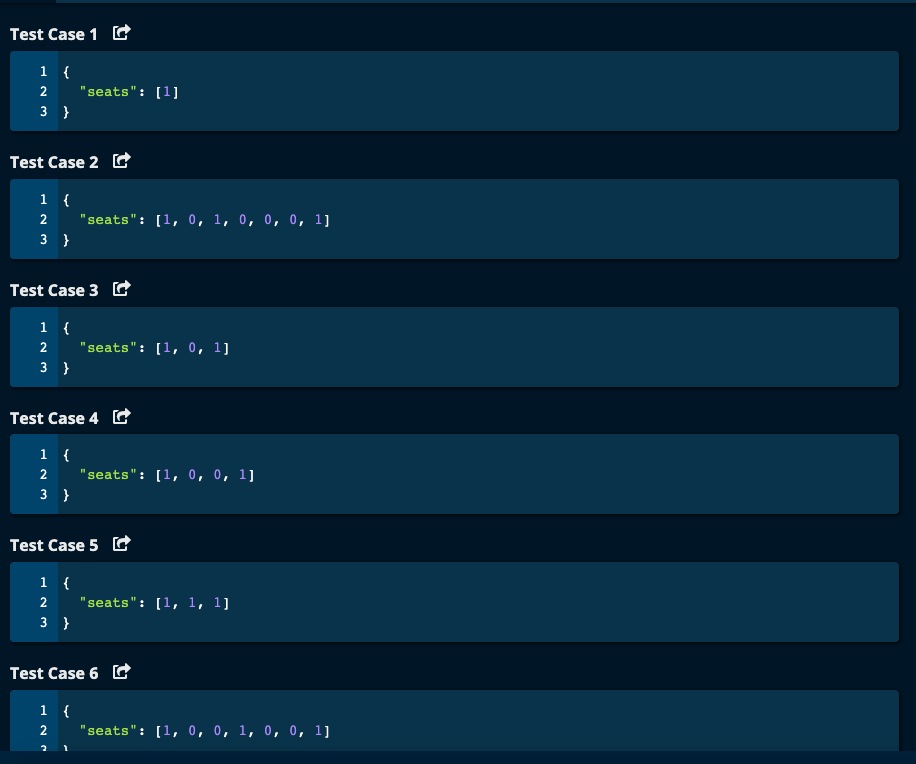
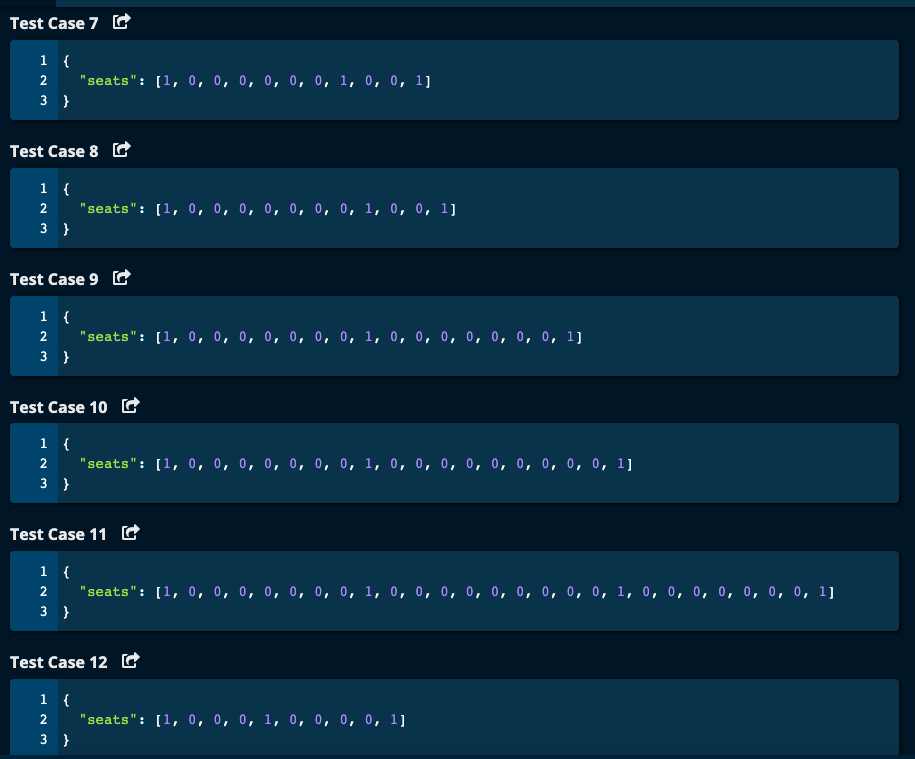
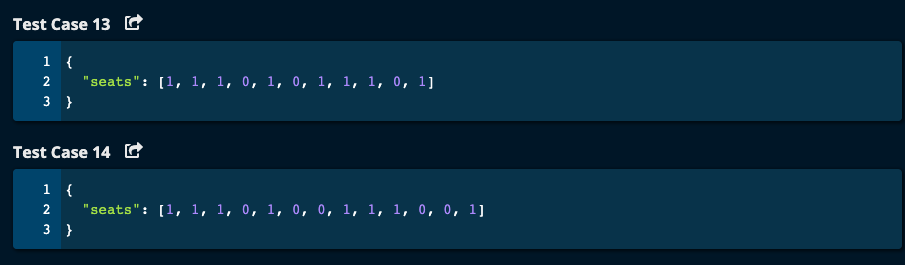
Best Seat (Medium)











My Solution :

def bestSeat(seats):

occupied = []

i = 0

while i < len(seats):

if seats[i] == 1:

occupied.append(i)

i += 1

print("occupied = ", occupied)

# to find max contiguous empty seats

maxContEmptySeats = 0

bestSeat = -1

emptySeats = 0

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

emptySeats = occupied[i] - occupied[i - 1] - 1 # seats in between the two occupied seats.

print("i = ", i, "emptySeats = ", emptySeats, "occupied[i] = ", occupied[i], "occupied[i - 1] = ", occupied[i -1])

if emptySeats > maxContEmptySeats:

maxContEmptySeats = emptySeats

bestSeat = (occupied[i - 1] + occupied[i])//2

return bestSeat

JJ Notes:

1. Find the list of occupied seats by traversing the seats array and checking for value of 1.

Now we have occupied array containing the index of the seats that are occupied.

1. Intialize bestSeat to -1, maxContEmptySeats to 0 and emptySeats to 0.
2. Traverse the occupied array. In the occupied array, check the difference between current position and the immediately preceding position to get the empty seats between two occupied seats, i.e. emptySeats. We need to subtract 1 because both numbers are excluded as they are occupied. If the empty seats are greats than maxContEmptySeats update it with emptySeats and bestSeat is middle of the continuous subarray of empty seats. If there are even number of empty seats, there can be two middle number. In this case, we will choose the one with the lower value.
3. Return bestSeat.

Time Complexity = O(n) as we traverse seats array only once and occupied array once (which is smaller than seats array).

Space Complexity =O(n) since occupied array can be as big as seats array.

NOTE: Algoexpert solution has O(1) space and is better.

Algoexpert Solution:

# Algoexpert Solution -- O(n) time | O(1) space

def bestSeat(seats):

bestSeat = -1

maxSpace = 0

left = 0

while left < len(seats):

right = left + 1

while right < len(seats) and seats[right] == 0:

right += 1

availableSpace = right - left - 1

if availableSpace > maxSpace:

maxSpace = availableSpace

bestSeat = (left + right)//2

left = right

return bestSeat

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

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