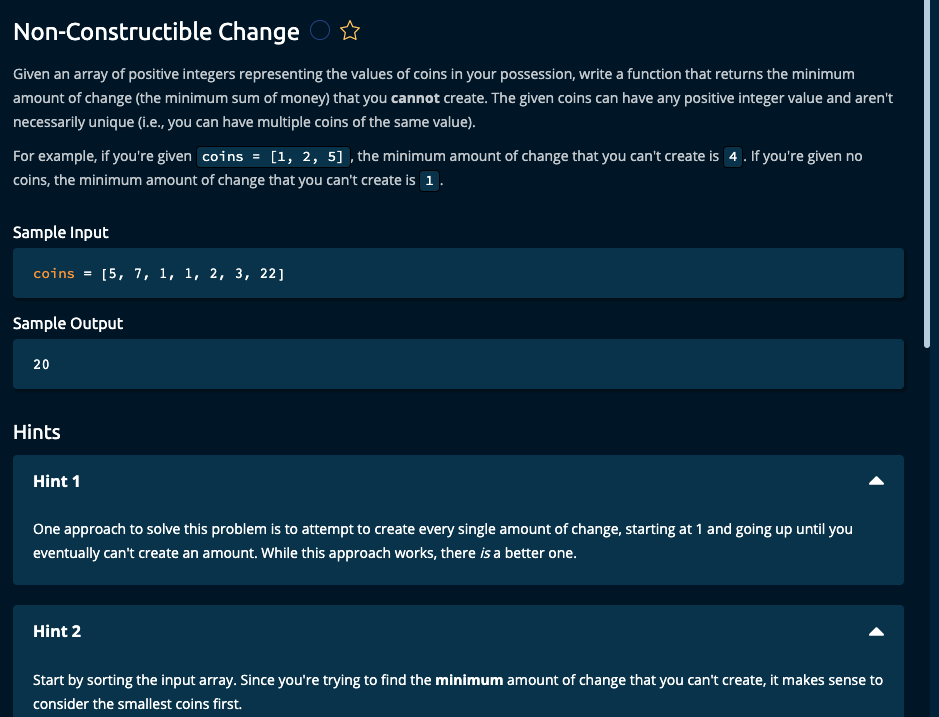
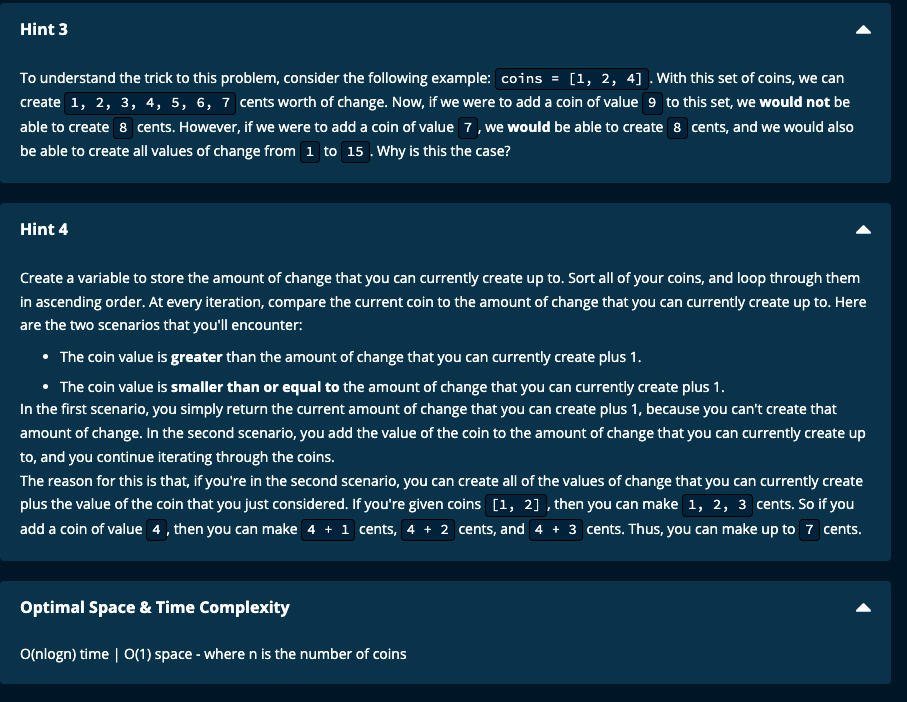
Non Constructible Change. (Easy)

<https://www.algoexpert.io/questions/non-constructible-change>





My Solution: Time complexity: O(nlog(n)) | Space complexity: O91)

def nonConstructibleChange(coins):

coins.sort()

if (not coins) or (coins[0] > 1):

return 1

currentChange = 0

for coin in coins:

if coin > currentChange + 1:

return currentChange + 1

else:

currentChange += coin

return currentChange + 1

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

Sort the coins array. If the sort can be done in place, time complexity = O(nlog(n) and space complexity = O(1)

Initialize change that can be made to 0, i.e. currentChange = 0

We have to iterate through the array just once.

If the first coin is not 1, then we cannot make a change of 1. So return 1.

At every step, we check if the coin is less than or equal to currentChange, and if so, add it to the currentChange, since if coin is more change + 1, we cannot make a change + 1.

If coin is greater than change + 1, return change + 1 (since we cannot make change + 1).

If we reach the end of the array, then the maximum amount of change that can be made is currentChange. So return currentChange + 1.

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

AlgoExpert Solution:

def nonConstructibleChange(coins):

coins.sort()

currentChangeCreated = 0

for coin in coins:

if coin > currentChangeCreated + 1:

return currentChangeCreated + 1

else:

currentChangeCreated += coin

return currentChangeCreated + 1

Time Complexity: O(nlogn) since sorting takes O(nlogn) and then iterating once through the array is O(n). However, since O(nlogn) > O(n), the time complexity is O(nlogn) + O(n) = O(nlogn)

Space complexity: If the sorting is done in place, no space is required for sorting. We then just need to store currentChangeCreated which is O(1) space.

However, if sorting is not done in place and a new array is created, then we need O(n) space for sorting. O(n) + O(1) is still O(n) since O(1) < O(n).