Question Three: Missing Number in Range

You are given a list containing n integers in the range [0, n]. Return a list of numbers that are missing from the range [0, n] of the array. If there is no missing number, return -1. Note, all the integers in the list may not be unique.

Section 1: Paraphrase the problem in your own words.

Partner's Summary

Write a function that takes a list of random and non-unique integers between 0 and n as an argument, and prints any missing numbers within that range. If ther eare no missing numbers, the function should simply print -1.

My Summary

In this problem you are given a list of non-unique integers as an input. A list is an ordered seqence, so we know our input will retain its displayed order. The list includes integers from 0 to n. We are required to implement a function that can take this list as input and return any missing numbers witin that range as output. However, if there are no missing integers in this list then we should return -1.

Section 2: Create 1 new example that demonstrates you understand the problem. Trace/walkthrough 1 example that your partner made and explain it.

My Example:

Input: lst = [5, 7, 3, 2]
Output: [0, 1, 4, 6]

Partner's Example:

Input: lst = [2, 4, 7, 0]
Output: [1, 3, 5, 6]

The input for my partner's example was a list with a range from 0-7. However, the list only contains 0, 2, 4 and 7. If we imagine the range of 0-7, that means we're missing 1, 3, 5, and 6. As such, we can see the output corresponds with this logic.

Section 3: Copy the solution your partner wrote.

```
# Import typing module with 'List' element
from typing import List
# Define the "missing number" function
def missing_num(nums: List) -> int:
# Create a master list with all integers between 0 and n
  nums = [0]
  n = max(lst)
  counter = 1
  for i in range (n):
    nums.append(counter)
    counter += 1
# Display the master list
  print ("Master list with all the integers:", nums)
# Create a nested loop that compares the submitted list and the master list
  for iteration in range(len(lst)):
    for n in reversed(range(len(nums))): # Iterating through "nums" in reverse order
      if lst[iteration] == nums[n]:
        nums.pop(n) # remove numbers found in submitted list
# Print "-1" if the two lists are identical otherwise display the missing numbers
  if not nums:
    print(-1)
  else:
    print ("Here are the missing numbers:", nums)
# Function ends and regular code with argument begins

lst = [0, 1, 2, 3, 10, 3, 7]
missing_num(lst)
```

```
Master list with all the integers: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
Here are the missing numbers: [4, 5, 6, 8, 9]
```

Section 4: Explain why their solution works in your own words.

Their solution works by taking advantage of an interative approach using for loops. More specifically they have decided to implement a nested for loop, which first iterates over the input list [0, n] and then iterates again over a master list [0, n]. The master list differs from the input list because it contains all integers in the range of the input list. The function iterates over each integer in the input list and then compares it to every value in the master list - if there's a match, then that

match is popped off the master list. Once each value in the input list has been iterated, the remaining values in the master list consisute the missing values.

Section 5: Explain the problem's time and space complexity in your own words.

The time complexity of this solution is O(n^2). The reason why is because there is a time complexity of O(n) for each for loop. Since these loops are nested, we must multiply the complexities together which gives us a time complexity of O(n^2).

The space complexity of this solution is O(n). This solution needs to generate and store a master list, nums, which depends on the range of the input list.

Section 6: Critique your partner's solution, including explanation, if there is anything should be adjusted.

The solution is very easy to understand, which is great. My only critique would be that we could potentially implement a solution that is more efficient in terms of time complexity. We could still use an interative strategy with loops, but instead of using nested for loops we could use a single while loop as seen below:

```
def missing_num(nums: List) -> int:
    i = 0
    missing = []
    while i<max(nums):
      if i not in nums:
        missing.append(i)
      i += 1
    if missing == []:
      print("-1")
    else:
      print(missing)

lst = [0, 1, 2, 3, 10, 3, 7]
missing_num(lst)
```

```
 [4, 5, 6, 8, 9]
```

This new implementation allows for a time complexity of O(n).

Reflection

These assignments collectively forced me to improve my familiarity with Python and as a regular user of R, I've come to appreciate this language. I find it much more intuitive and appreciate how much easier it is to interpret others' code. I have previous experience using loops in R, so to complete Assignment 1 I decided to study recursive functions thoroughly to implement them into my solution. I learned practically how to take advantage of the call stack to create tree paths in a stepwise manner. I still find it challenging to logically follow the call stack, but recursive functions are a tool I will use moving forward. Furthermore, I was required to study Big O notation and had to think critically to apply it to my solutions time and space complexity. Moving forward, I will take time complexity into consideration when designing solutions – particularly when working with big sequencing datasets. Finally, I think my favourite part of these assignments was critiquing my partners solution. It was fun to critically think about whether a more efficient solution might exist in terms of complexity and then trying to code that myself. One of the most interesting aspects of solving these problems is recognizing how many different implementations exist that could result in the same answer.