Part 1: You will be assigned one of three problems based of your first name. Excute " (hash('your first name')%3)+1" in python, and that will tell you your assigned problem. Include this line of code in your submitted notebook pdf. You can find the problem description in q[assigned number].md. They are based-off problems from Leetcode.

In [ ]:
```
(hash('Ben Ho')%3)+1
```

Out[ ]: 3

See assignment 3 here: https://github.com/UofT-DSI/algorithms_and_data_structures/blob/main/assignments/q3.md

Part 2: In a Jupyter Notebook (.ipynb) file, create 6 headings are write down the following:

Paraphrase the problem in your own words

- Give a range of unsorted and non-unique integers, return a sorted list of missing integers. If there are no missing values, return -1

In the .md file containing your problem, there are examples that illustrate how the code should work. Create 2 new examples that demonstrate you understand the problem.

In [ ]:
```
### Example 1

Input: `lst = [5, 0, 4, 12, 9]`

Output: [1, 2, 3, 6, 7, 8, 10, 11]

### Example 2

Input: `lst = [1, 2, 3, 4]`

Output: -1
```

Code the solution to your assigned problem in Python (code chunk). Try to find the best time and space complexity solution!

In [ ]:
```
def missing_num(nums: lst) -> int:
  #create sorted list
  lst_sorted=sorted(lst) #O(n^2)
  #create full range of numbers with first and last object
  full_range=range(lst_sorted[0], lst_sorted[-1], 1)

  #create new list to store output
  lst_missing=[]

  #iterate though full_range and return items if it is not in input list to
```

```
  for i in full_range: #O(n)
    if i not in lst:
      lst_missing.append(i)

  #return new list is size is not 0, else returns -1
  if len(lst_missing) > 0:
    return(lst_missing)
  else:
    return(-1)
```

In [ ]:
```
#test 1
lst = [5, 0, 4, 12, 9]
missing_num(lst)
```

Out[ ]:
```
[1, 2, 3, 6, 7, 8, 10, 11]
```

In [ ]:
```
#test 2
lst = [1, 2, 3, 4]
missing_num(lst)
```

Out[ ]:
```
-1
```

Explain why your solution works

This solution works because it ignores duplicated values by taking the first and last value in a sorted list. Once the minimum and maximum values are obtained, then the rest of the solution is down returning items that has not appeard in the original list.

Explain the problem's and space complexity

There are two bottleneck for time complexity in the implemented solution: the first is from sorting the list with sorted() which has O(n^2); checking for missing values from the original list through iterative loop have time complexity of O(n). As such, the overall time complexity is O(n^2)

In terms of space complexity, there are 3 lists that are created and they are all O(n).

Explain the thinking to an alternative solution (no coding required, but a classmate reading this should be able to code it up based off your text)

In [ ]:
```
#alternative solution using set difference
#1. given input integer 'lst', create sorted range of numbers
#2. get min max from sorted range and create 'full_range'
#3. apply difference() of full_range on lst
```