(hash('RajaGopal')%3)+1

**3**

**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.

**Paraphrase the Problem:**

The task at hand involves analyzing a list of integers that span a specific numerical range from 0 to n, inclusive. Here, 'n' represents the maximum potential value that, based on the list's length or its content, should ideally be present within this collection. The primary objective is to identify any and all integers that are conspicuously absent from this specified range. It's important to note that the list may not be uniquely composed, meaning duplicates can exist. Moreover, the directive specifies that in scenarios where the list is complete—i.e., no numbers are found to be missing—the function should signify this by returning a value of -1.

**Create New Examples:**

**Illustrative Example 4**

Consider the input list: **lst = [3, 3, 0, 1]**.

The expected output for this scenario would be: **[2]**.

**Rationale**:

Here, the list ostensibly spans the numbers from 0 to 3. However, upon closer inspection, it becomes evident that the number 2 is absent, hence its appearance in the output as the missing element.

**Illustrative Example 5**

Now, take the input list: **lst = [0, 1, 2, 3, 4]**.

The anticipated output in this case would be: **-1**.

**Rationale**:

This example showcases a list that comprehensively includes all numbers within its expected range of 0 to 4. Consequently, there are no missing numbers to report, which is accurately reflected in the output as **-1**.

**Code the Solution:**

def missing\_num(nums):

# Determine the maximum value 'n' which sets the expected numerical range

n = max(nums)

# Comprehensively identify all numbers missing from the input list within the range of 0 to n

missing = [num for num in range(n + 1) if num not in nums]

# Conditionally return the list of missing numbers, or -1 if the list is found to be complete

return missing if missing else -1

**Explain Why Your Solution Works:**

This solution is effective because it directly addresses the problem's core requirement: to identify any numbers missing from a specified range within the input list. By constructing a range from 0 to 'n'—with 'n' being the highest value that should be present—and iterating through this range, the solution methodically checks for the absence of each number within the input list. The use of a list comprehension for this purpose not only simplifies the code but also ensures that each number's presence (or lack thereof) is checked. Should any numbers be found missing, they are collected into a list. Conversely, if no numbers are missing, the function defaults to returning -1, adhering to the problem's specifications.

**Explain the Problem's Time and Space Complexity:**

The proposed solution exhibits a worst-case time complexity of O(n^2), primarily due to the nested operations within the list comprehension. For every number in the range **[0, n]**, there's an associated check to determine its presence in the **nums** list, each of which incurs O(n) time complexity due to the linear search.

The space complexity is determined by the variable **missing**, which accumulates all absent numbers within a list. Hence, the space complexity is O(m), where 'm' represents the count of missing numbers. In scenarios where the input list is complete, and no numbers are missing, the space complexity effectively reduces to O(1), given that the only return value is **-1**, which doesn't necessitate additional space proportional to the input size.

**Thinking of an Alternative Solution:**

An efficient alternative approach leverages a HashSet (or simply, a set in Python) to facilitate more expedient lookups. The procedural outline is as follows:

1. **Initialization**: Populate a HashSet with all integers present in the input list. This data structure choice enables O(1) average time complexity for subsequent membership checks.
2. **Iterative Check**: Sequentially iterate through the entire range of **[0, n]**, where 'n' signifies the maximum value expected to be present.
3. **Missing Number Identification**: For each iteration, ascertain whether the current number is absent from the HashSet.
4. **Result Compilation**: Aggregate all numbers discovered to be missing into a cohesive list. Should this list be empty (indicating no missing numbers), return **-1** as per the problem's directive.

def find\_missing\_numbers(nums):

# Convert the list to a set for O(1) lookups

num\_set = set(nums)

# Determine the maximum value 'n' to set the range

n = max(nums)

# Identify missing numbers within the range [0, n]

missing\_numbers = [num for num in range(n + 1) if num not in num\_set]

# Return the missing numbers or -1 if there are none

return missing\_numbers if missing\_numbers else -1

# Example inputs

inputs = [

[3, 3, 0, 1], # Example 4

[0, 1, 2, 3, 4], # Example 5

[0, 2], # Additional example

[5, 0, 1] # Additional example

]

# Execute and print the output for each input

for input\_list in inputs:

output = find\_missing\_numbers(input\_list)

print(f"Input: {input\_list}\nOutput: {output}\n")

This methodology significantly enhances the time complexity to O(n), primarily because each number's presence within the set is ascertainable in constant time, and the process iterates through the range **[0, n]** and the input list only once to construct the set. The space complexity escalates to O(n) due to the necessity of storing each unique value from the input list in a HashSet, which facilitates the rapid execution of range checks.