Documentation: Forming the Largest Number from a List of Non-Negative Integers

Problem Statement

- Given a list of non-negative integers, nums, the objective is to arrange them in such a way that they form the largest possible number when concatenated. The result should be returned as a string, as the output may be too large to be represented as an integer.

Key Considerations

1. **String Representation:**

- Each integer in the list should be treated as a string for concatenation purposes. This allows for flexible arrangement of digits to form different potential numbers.

2. **Custom Sorting:**

- The primary challenge is to determine the correct order in which the numbers should be arranged to form the largest concatenated result.
- For any two numbers, a and b, the order in which they should appear depends on which of the two concatenations, a + b or b + a, is greater.
- For example, for numbers 3 and 30, the concatenations are 330 and 303. Since 330 is larger than 303, 3 should come before 30.

3. **Edge Cases:**

- ***All Zeros:*** If the list contains only zeros, the output should be "0". This is to avoid leading zeros and to ensure that the representation of the number is correct.
- ***Single Element:*** If the list has only one element, the output should be the string representation of that single element.
- ***Mixed Digits:*** Different lengths and combinations of numbers require careful comparison to ensure the largest result is formed.

Constraints

- 1 <= nums.length <= 100: The length of the input list can range from 1 to 100.
- 0 <= nums[i] <= 10^9: Each element in the list is a non-negative integer and can be as large as (10^9).

Approach to Solution

1. **Convert Integers to Strings:**
   - Convert each integer in the list to its string representation. This is crucial because string concatenation allows us to evaluate which order produces the larger number.

2. **Define a Custom Comparator:**
   - A comparator function is defined to compare two string representations, x and y.
   - The function compares the concatenated results of x + y and y + x.
   - If x + y is greater than y + x, x should come before y in the final sorted order.
   - Conversely, if y + x is greater, then y should come before x.

3. **Sort the List Using the Custom Comparator:**

- The list of strings is sorted using the custom comparator. This ensures that the numbers are arranged in such a way that their concatenation yields the largest possible number.

4. **Concatenate the Sorted List:**

- After sorting, concatenate all the string elements of the list to form the final result.

5. **Handle the Edge Case of All Zeros:**

- After concatenation, if the first character of the result is '0', the entire number is zero (e.g., when the input is [0, 0]), and thus, the function should return "0".

6. **Return the Result:**

- Finally, return the concatenated string as the result.

Example 1:

- **Input:** nums = [10, 2]
- **Process:**
  - ***Convert to strings:*** ["10", "2"]
  - ***Custom sort:*** ["2", "10"]
  - ***Concatenate:*** "210"
- **Output:** "210"

Example 2:

- **Input:** nums = [3, 30, 34, 5, 9]
- **Process:**
  - ***Convert to strings:*** ["3", "30", "34", "5", "9"]
  - ***Custom sort:*** ["9", "5", "34", "3", "30"]
  - ***Concatenate:*** "9534330"
- **Output:** "9534330"

Complexity Analysis

- **Time Complexity:** The sorting step dominates the time complexity. Given that there are n numbers and each comparison between two numbers can take up to (O(k)) time (where k is the maximum number of digits in any number, which is at most 9 for 10^9), the overall time complexity is (O(n log n cdot k)).

- **Space Complexity:** The space complexity is (O(n)) due to the space required to store the string representations of the numbers.

Conclusion

- The problem of forming the largest number from a list of non-negative integers involves using a custom sorting strategy based on string concatenation comparisons. By carefully ordering the numbers and handling edge cases, we can efficiently determine the largest possible number that can be formed. The solution effectively balances the constraints and provides a clear, concise output for any valid input list.