<https://leetcode.com/problems/largest-number/description/>

Given a list of non-negative integers nums, arrange them such that they form the largest number and return it.

Since the result may be very large, so you need to return a string instead of an integer.

**Example 1:**

**Input:** nums = [10,2]

**Output:** "210"

**Example 2:**

**Input:** nums = [3,30,34,5,9]

**Output:** "9534330"

**Constraints:**

1 <= nums.length <= 100

0 <= nums[i] <= 10^9

**Attempt 1: 2025-09-07**

**Solution 1: Sorting + String (10 min)**

class Solution {

    public String largestNumber(int[] nums) {

        String[] numStrs = new String[nums.length];

        for(int i = 0; i < nums.length; i++) {

            numsStrs[i] = String.valueOf(nums[i]);

        }

        // Custom comparator: compare a+b vs b+a

        Arrays.sort(numStrs, (a, b) -> {

            String order1 = a + b;

            String order2 = b + a;

            // Descending order

            return order2.compareTo(order1);

        });

        // If the largest number is "0", return "0"

        // e.g nums = {0, 0}

        if(numStrs[0].equals("0")) {

            return "0";

        }

        // Build the result string

        StringBuilder sb = new StringBuilder();

        for (String numStr : numStrs) {

            sb.append(numStr);

        }

        return sb.toString();

    }

}

Time Complexity: O(n)

Space Complexity: O(n)

**Refer to Deepseek**

I'll provide a Java solution for LeetCode 179. "Largest Number", along with an explanation and complexity analysis.

**Problem Description**

Given a list of non-negative integers nums, arrange them such that they form the largest number and return it as a string. Note that the result may be very large, so you need to return a string instead of an integer.

**Java Solution**

import java.util.\*;

class Solution {

public String largestNumber(int[] nums) {

// Convert integers to strings

String[] numStrs = new String[nums.length];

for (int i = 0; i < nums.length; i++) {

numStrs[i] = String.valueOf(nums[i]);

}

// Custom comparator: compare a+b vs b+a

Arrays.sort(numStrs, (a, b) -> {

String order1 = a + b;

String order2 = b + a;

return order2.compareTo(order1); // Descending order

});

// If the largest number is "0", return "0"

if (numStrs[0].equals("0")) {

return "0";

}

// Build the result string

StringBuilder sb = new StringBuilder();

for (String numStr : numStrs) {

sb.append(numStr);

}

return sb.toString();

}

}

**Explanation**

**Convert to Strings**:

Convert each integer in nums to a string to facilitate custom comparison.

**Custom Sorting**:

The key insight is that to form the largest number, we should compare two numbers by concatenating them in different orders.

For two strings a and b, we compare a+b and b+a. If b+a is larger than a+b, then b should come before a in the sorted array.

Example: compare "3" and "30".

"3" + "30" = "330"

"30" + "3" = "303"

Since "330" > "303", "3" should come before "30".

**Handle Leading Zeros**:

After sorting, if the largest number (first element) is "0", it means all numbers are zero, so we return "0" to avoid a string like "00...0".

**Build Result**:

Concatenate the sorted strings to form the largest number.

**Example Walkthrough**

Let nums = [3, 30, 34, 5, 9]:

Convert to strings: ["3", "30", "34", "5", "9"]

Sort using custom comparator:

Compare "3" and "30": "330" vs "303" → "330" is larger → "3" comes before "30".

Similarly, compare "9" and "34": "934" vs "349" → "934" is larger → "9" comes before "34".

Sorted order: ["9", "5", "34", "3", "30"]

Concatenate: "9534330" which is the largest number.

**Complexity Analysis**

**Time Complexity**: O(n log n) due to sorting, where n is the number of elements. Each comparison takes O(k) time (where k is the average length of numbers), so overall O(k \* n log n).

**Space Complexity**: O(n) for storing the string array.

This solution efficiently handles the problem by leveraging custom string comparison. Let me know if you need further clarification!

**Refer to**

[L2165.Smallest Value of the Rearranged Number (Ref.L179)](note://WEB9ca2a74d19fe213bbddba90f96774b8d)

[L3270.Find the Key of the Numbers (Ref.L179)](note://WEB533d97f4385f313bb61ca56677ca766d)