LeetCode 179. Largest Number

1. Problem Title & Link

- **179. Largest Number**
- https://leetcode.com/problems/largest-number/

2. Problem Statement (Short Summary)

We are given a list of non-negative integers nums. Arrange them such that they form the **largest possible number** when concatenated.
Return the result as a string (since the number may be very large).

3. Examples (Input → Output)

Input: nums = [10,2]

Output: "210"

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

Output: "9534330"

4. Constraints

- 1 <= nums.length <= 100
- 0 <= nums[i] <= 10^9

5. Thought Process (Step by Step)

- This is **not just sorting numerically**.
- Example: 9 + 34 → "934", but 34 + 9 → "349".
- Rule: Compare concatenated results of a+b vs b+a (as strings).
- Use custom comparator:
  - If a+b > b+a, then a should come before b.

6. Pseudocode (Language-Independent)

```
function largestNumber(nums):
    convert all numbers to strings
    sort strings using custom comparator:
        if a+b > b+a → a before b
    join sorted strings
    if result starts with "0":
        return "0"
    else:
        return result
```

7. Code Implementation

✅ **Python**

```
from functools import cmp_to_key

class Solution:
    def largestNumber(self, nums: List[int]) -> str:
        def compare(a, b):
            if a + b > b + a:
                return -1
            elif a + b < b + a:
                return 1
            else:
                return 0

        nums = list(map(str, nums))
        nums.sort(key=cmp_to_key(compare))
        result = "".join(nums)
        return "0" if result[0] == "0" else result
```

✅ **Java**

```
class Solution {
    public String largestNumber(int[] nums) {
        String[] arr = new String[nums.length];
        for (int i = 0; i < nums.length; i++) {
            arr[i] = String.valueOf(nums[i]);
        }

        Arrays.sort(arr, (a, b) -> (b + a).compareTo(a + b));

        if (arr[0].equals("0")) return "0";
        StringBuilder sb = new StringBuilder();
        for (String s : arr) sb.append(s);

        return sb.toString();
    }
}
```

8. Time & Space Complexity Analysis

- Sorting complexity: O(n log n * k) where k is average string length.
- Space complexity: O(n) for storing string representations.

9. Common Mistakes / Edge Cases

- Forgetting to check for leading "0" → result like "0000" instead of "0".
- Sorting only numerically instead of using a+b vs b+a.
- Not converting numbers to strings before comparison.

10. Variations / Follow-Ups

- Form **smallest number** instead of largest (reverse comparator).
- Handle very large arrays efficiently.

11. Dry Run (Step by Step Execution)

👉 Input: nums = [3, 30, 34, 5, 9]

1. Convert to strings: ["3","30","34","5","9"]
2. Sort with custom comparator:
   - Compare "3" vs "30" → "330" vs "303" → "3" > "30" → "3" before "30".
   - Compare "34" vs "3" → "343" vs "334" → "34" before "3".
   - Compare "9" vs "34" → "934" vs "349" → "9" before "34".
   - Compare "5" vs "9" → "59" vs "95" → "9" before "5".
3. Sorted order: ["9","5","34","3","30"]
4. Join: "9534330"

✅ Output: "9534330"