<https://leetcode.com/problems/smallest-value-of-the-rearranged-number/description/>

You are given an integer num. **Rearrange** the digits of num such that its value is **minimized** and it does not contain **any** leading zeros.

Return *the rearranged number with minimal value*.

Note that the sign of the number does not change after rearranging the digits.

**Example 1:**

**Input:** num = 310

**Output:** 103

**Explanation:** The possible arrangements for the digits of 310 are 013, 031, 103, 130, 301, 310.

The arrangement with the smallest value that does not contain any leading zeros is 103.

**Example 2:**

**Input:** num = -7605

**Output:** -7650

**Explanation:** Some possible arrangements for the digits of -7605 are -7650, -6705, -5076, -0567.

The arrangement with the smallest value that does not contain any leading zeros is -7650.

**Constraints:**

-10^15 <= num <= 10^15

**Attempt 1: 2024-09-07**

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

class Solution {

    public long smallestNumber(long num) {

        if(num == 0) {

            return 0;

        }

        boolean isNegative = num < 0;

        num = Math.abs(num);

        // Convert the number to a string to process digits

        char[] digits = String.valueOf(num).toCharArray();

        if(isNegative) {

            // For negative numbers: we want the largest absolute

            // value (which becomes the smallest negative)

            Arrays.sort(digits);

            // Reverse to get descending order (largest number)

            reverse(digits);

            return -Long.parseLong(new String(digits));

        } else {

            // For positive numbers: we want the smallest number without leading zeros

            Arrays.sort(digits);

            // Find the first non-zero digit to swap with the first digit

            int firstNonZeroIndex = 0;

            while(firstNonZeroIndex < digits.length && digits[firstNonZeroIndex] == '0') {

                firstNonZeroIndex++;

            }

            // If there are leading zeros, swap the first zero with the first non-zero

            if(firstNonZeroIndex > 0) {

                char tmp = digits[0];

                digits[0] = digits[firstNonZeroIndex];

                digits[firstNonZeroIndex] = tmp;

            }

            return Long.parseLong(new String(digits));

        }

    }

    private void reverse(char[] chars) {

        int i = 0;

        int j = chars.length - 1;

        while(i < j) {

            char tmp = chars[i];

            chars[i] = chars[j];

            chars[j] = tmp;

            i++;

            j--;

        }

    }

}

Time Complexity: O(d log d) where d is the number of digits (for sorting).

Space Complexity: O(d) for storing the digits.

**Refer to Deepseek**

I'll provide a Java solution for LeetCode 2165. "Smallest Value of the Rearranged Number", along with an explanation and complexity analysis.

**Problem Description**

You are given an integer num (which can be negative). Rearrange the digits of num to create the smallest possible number (without leading zeros) and return it.

**Java Solution**

import java.util.\*;

class Solution {

public long smallestNumber(long num) {

if (num == 0) return 0;

boolean isNegative = num < 0;

num = Math.abs(num);

// Convert the number to a string to process digits

char[] digits = String.valueOf(num).toCharArray();

if (isNegative) {

// For negative numbers: we want the largest absolute value (which becomes the smallest negative)

Arrays.sort(digits);

// Reverse to get descending order (largest number)

reverse(digits);

return -Long.parseLong(new String(digits));

} else {

// For positive numbers: we want the smallest number without leading zeros

Arrays.sort(digits);

// Find the first non-zero digit to swap with the first digit

int firstNonZeroIndex = 0;

while (firstNonZeroIndex < digits.length && digits[firstNonZeroIndex] == '0') {

firstNonZeroIndex++;

}

// If there are leading zeros, swap the first zero with the first non-zero

if (firstNonZeroIndex > 0) {

char temp = digits[0];

digits[0] = digits[firstNonZeroIndex];

digits[firstNonZeroIndex] = temp;

}

return Long.parseLong(new String(digits));

}

}

private void reverse(char[] arr) {

int left = 0, right = arr.length - 1;

while (left < right) {

char temp = arr[left];

arr[left] = arr[right];

arr[right] = temp;

left++;

right--;

}

}

}

**Explanation**

**Handle Zero**: If num is 0, return 0.

**Check Sign**: Determine if the number is negative. For negative numbers, we need to form the largest absolute value (which becomes the smallest negative when sign is applied).

**Process Digits**:

**For Negative Numbers**:

Sort the digits in ascending order, then reverse to get descending order (largest absolute value).

Return the negative of this value.

**For Positive Numbers**:

Sort the digits in ascending order.

If there are leading zeros, find the first non-zero digit and swap it with the first digit to avoid leading zeros.

Return the formed number.

**Helper Function**: reverse to reverse the sorted array for negative numbers.

**Example Walkthrough**

**Positive Number**: num = 310

Digits: ['3','1','0'] → sorted: ['0','1','3']

First non-zero is '1' at index 1. Swap with first digit: ['1','0','3']

Result: 103

**Negative Number**: num = -7605

Absolute value: 7605 → digits: ['7','6','0','5']

Sorted ascending: ['0','5','6','7'] → reversed: ['7','6','5','0']

Result: -7650 (which is the smallest negative, i.e., the largest absolute value with negative sign).

**Complexity Analysis**

**Time Complexity**: O(d log d) where d is the number of digits (for sorting).

**Space Complexity**: O(d) for storing the digits.

This solution efficiently handles both positive and negative cases by leveraging sorting and careful swapping for leading zeros. Let me know if you need further clarification!

**Refer to**

[L179.Largest Number (Ref.L2165,L3270)](note://WEB9f5b5d5c2dc29249536a62e172c6549e)