<https://leetcode.com/problems/find-the-key-of-the-numbers/description/>

You are given three **positive** integers num1, num2, and num3.

The key of num1, num2, and num3 is defined as a four-digit number such that:

Initially, if any number has **less than** four digits, it is padded with **leading zeros**.

The ith digit (1 <= i <= 4) of the key is generated by taking the **smallest** digit among the ith digits of num1, num2, and num3.

Return the key of the three numbers **without** leading zeros (*if any*).

**Example 1:**

**Input:** num1 = 1, num2 = 10, num3 = 1000

**Output:** 0

**Explanation:**

On padding, num1 becomes "0001", num2 becomes "0010", and num3 remains "1000".

The 1st digit of the key is min(0, 0, 1).

The 2nd digit of the key is min(0, 0, 0).

The 3rd digit of the key is min(0, 1, 0).

The 4th digit of the key is min(1, 0, 0).

Hence, the key is "0000", i.e. 0.

**Example 2:**

**Input:** num1 = 987, num2 = 879, num3 = 798

**Output:** 777

**Example 3:**

**Input:** num1 = 1, num2 = 2, num3 = 3

**Output:** 1

**Constraints:**

1 <= num1, num2, num3 <= 9999

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

**Solution 1: String (30 min)**

class Solution {

    public int generateKey(int num1, int num2, int num3) {

        // Convert numbers to strings with leading zeros to have same length

        String s1 = String.valueOf(num1);

        String s2 = String.valueOf(num2);

        String s3 = String.valueOf(num3);

        int maxLen = Math.max(s1.length(), Math.max(s2.length(), s3.length()));

        // Pad with leading zeros

        s1 = String.format("%" + maxLen + "s", s1).replace(' ', '0');

        s2 = String.format("%" + maxLen + "s", s2).replace(' ', '0');

        s3 = String.format("%" + maxLen + "s", s3).replace(' ', '0');

        StringBuilder sb = new StringBuilder();

        for(int i = 0; i < maxLen; i++) {

            char c1 = s1.charAt(i);

            char c2 = s2.charAt(i);

            char c3 = s3.charAt(i);

            char minChar = c1;

            if(c2 < minChar) {

                minChar = c2;

            }

            if(c3 < minChar) {

                minChar = c3;

            }

            sb.append(minChar);

        }

        // Remove leading zeros

        String key = sb.toString();

        int index = 0;

        while(index < sb.length() - 1 && sb.charAt(index) == '0') {

            index++;

        }

        key = key.substring(index);

        return Integer.parseInt(key);

    }

}

Time Complexity: O(maxLen) where maxLen is the number of digits of the largest number.

The operations per digit are constant.

Space Complexity: O(maxLen) for storing the padded strings.

**Refer to Deepseek**

I'll provide a Java solution for LeetCode 3270. "Find the Key of the Numbers", along with an explanation and complexity analysis.

**Problem Description**

You are given three integers num1, num2, and num3. The key of the numbers is defined as the smallest number that can be formed by aligning the digits of these numbers in a specific way. Specifically, you need to compare the digits of the three numbers one by one (from least significant to most significant) and for each position, take the smallest digit available. Then form the key from these digits.

More formally:

Convert each number to a string with leading zeros to make them of equal length (the length of the largest number).

For each digit position (from left to right), choose the smallest digit among the three numbers at that position.

The key is the number formed by these chosen digits (without leading zeros).

**Java Solution**

class Solution {

public int generateKey(int num1, int num2, int num3) {

// Convert numbers to strings with leading zeros to have same length

String s1 = String.valueOf(num1);

String s2 = String.valueOf(num2);

String s3 = String.valueOf(num3);

int maxLen = Math.max(s1.length(), Math.max(s2.length(), s3.length()));

// Pad with leading zeros

s1 = String.format("%" + maxLen + "s", s1).replace(' ', '0');

s2 = String.format("%" + maxLen + "s", s2).replace(' ', '0');

s3 = String.format("%" + maxLen + "s", s3).replace(' ', '0');

StringBuilder keyBuilder = new StringBuilder();

for (int i = 0; i < maxLen; i++) {

char c1 = s1.charAt(i);

char c2 = s2.charAt(i);

char c3 = s3.charAt(i);

// Find the smallest digit at this position

char minDigit = c1;

if (c2 < minDigit) minDigit = c2;

if (c3 < minDigit) minDigit = c3;

keyBuilder.append(minDigit);

}

// Convert to integer to remove any leading zeros (if any) but note: the key might have leading zeros?

// However, the problem says "smallest number", so we should return without leading zeros.

String keyStr = keyBuilder.toString();

// Remove leading zeros

int startIndex = 0;

while (startIndex < keyStr.length() - 1 && keyStr.charAt(startIndex) == '0') {

startIndex++;

}

keyStr = keyStr.substring(startIndex);

return Integer.parseInt(keyStr);

}

}

**Explanation**

**Equalize Lengths**: Convert each number to a string and pad with leading zeros so all have the same length (the length of the largest number).

**Build Key Digit by Digit**: For each digit position (from left to right), compare the digits of the three numbers and choose the smallest.

**Remove Leading Zeros**: The formed key might have leading zeros. We remove them to form the smallest number without leading zeros.

**Return as Integer**: Convert the resulting string to an integer.

**Example Walkthrough**

Let num1 = 1, num2 = 10, num3 = 100:

After padding:

s1 = "001"

s2 = "010"

s3 = "100"

For each position:

Position0: min('0','0','1') = '0'

Position1: min('0','1','0') = '0'

Position2: min('1','0','0') = '0'

Key string: "000" → after removing leading zeros: "0" → return 0.

Another example: num1 = 123, num2 = 456, num3 = 789:

After padding: all have length 3.

Position0: min('1','4','7') = '1'

Position1: min('2','5','8') = '2'

Position2: min('3','6','9') = '3'

Key: "123" → return 123.

**Complexity Analysis**

**Time Complexity**: O(maxLen) where maxLen is the number of digits of the largest number. The operations per digit are constant.

**Space Complexity**: O(maxLen) for storing the padded strings.

This solution efficiently constructs the key by comparing digits position-wise. Let me know if you need further clarification!

**Can you explain how this line works ?**

String.format("%" + maxLen + "s", s1).replace(' ', '0')

Certainly! Let's break down the line:

This line is used to pad the string s1 with leading zeros so that its length becomes maxLen. Here's a step-by-step explanation:

**1. String.format("%" + maxLen + "s", s1)**

The String.format method is used to format a string according to a format specifier.

The format specifier %" + maxLen + "s" is constructed dynamically. For example, if maxLen is 5, the specifier becomes "%5s".

"%5s" means: format the string to be **right-justified** in a field of width 5. If the original string is shorter than 5, it will be padded with spaces on the left.

Example:

If s1 = "123" and maxLen = 5, then String.format("%5s", "123") returns " 123" (with two leading spaces).

**2. .replace(' ', '0')**

This replaces every space character in the formatted string with a zero.

Continuing the example:

" 123".replace(' ', '0') becomes "00123".

**Why is this done?**

We need to compare digits of numbers position-wise (from left to right). To do this, we must ensure all numbers have the same number of digits (by adding leading zeros).

For instance, if we have numbers 1, 10, and 100, we want to represent them as 001, 010, and 100 to compare digits at each position.

**Example:**

Suppose:

num1 = 7 (which is "7")

maxLen = 3 (because the largest number has 3 digits)

Then:

String.format("%3s", "7") gives " 7" (two spaces and then '7').

After replacing spaces with zeros: "007".

This ensures that all numbers have the same length for digit-by-digit comparison.

**Alternative:**

You can also use:

s1 = "0".repeat(maxLen - s1.length()) + s1;

But the String.format method is more concise and avoids explicit loop for padding.

Let me know if you need further clarification!

**Refer to**

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