# 题目

整数的数组形式num是按照从左到右的顺序表示其数字的数组。

例如，对于num = 1321 ，数组形式是 [1,3,2,1] 。

给定num，整数的数组形式，和整数k ，返回整数num + k的数组形式 。

示例 1：

输入：num = [1,2,0,0], k = 34

输出：[1,2,3,4]

解释：1200 + 34 = 1234

示例 2：

输入：num = [2,7,4], k = 181

输出：[4,5,5]

解释：274 + 181 = 455

示例 3：

输入：num = [2,1,5], k = 806

输出：[1,0,2,1]

解释：215 + 806 = 1021

提示：

1 <= num.length <= 104

0 <= num[i] <= 9

num 不包含任何前导零，除了零本身

1 <= k <= 104

# 分析

要解决“整数的数组形式与整数相加”的问题，核心思路是模拟手工加法过程，从最低位（数组末尾）开始逐位相加，处理进位，最终得到结果的数组形式。

解题思路

1、模拟加法过程：

整数的数组形式（如[1,2,0,0]）和整数k（如34）相加，本质是两个数的逐位累加。由于数组按“最高位→最低位”存储，而加法需从“最低位”开始，因此需从数组末尾遍历，同时逐步提取k的每一位（通过k % 10）。

2、处理进位：

每一步计算当前位的总和（数组当前位 + k的当前位 + 上一步的进位），当前位的结果为总和的个位数（sum % 10），新的进位为总和的十位数（sum / 10）。

3、处理剩余进位：

当数组遍历完毕或k的所有位处理完毕后，若仍有进位，需将进位的每一位依次添加到结果中。

4、调整结果顺序：

由于计算过程是从最低位到最高位存储结果，最终需将结果逆序，得到“最高位→最低位”的正确数组形式。

代码实现（含详细注释）：

class Solution {

public:

vector<int> addToArrayForm(vector<int>& num, int k) {

vector<int> result; // 存储结果（先按“最低位→最高位”顺序）

int carry = 0; // 进位，初始为0

int i = num.size() - 1; // 从数组末尾（最低位）开始遍历

// 循环条件：数组未遍历完 或 k未处理完 或 仍有进位

while (i >= 0 || k > 0 || carry > 0) {

// 取数组当前位的值（若已遍历完，视为0）

int digit\_num = (i >= 0) ? num[i] : 0;

// 取k的当前位的值（k%10），并更新k（移除当前位）

int digit\_k = k % 10;

k = k / 10;

// 计算当前位的总和（数组位 + k的位 + 进位）

int sum = digit\_num + digit\_k + carry;

// 当前位的结果为总和的个位数

result.push\_back(sum % 10);

// 更新进位（总和的十位数）

carry = sum / 10;

// 数组指针左移（处理更高一位）

i--;

}

// 结果当前是“最低位→最高位”，逆序后转为“最高位→最低位”

reverse(result.begin(), result.end());

return result;

}

};

代码解释

1、初始化与遍历：

从数组末尾（i = num.size() - 1）和k的个位（k % 10）开始，同时处理数组、k和进位。

2、逐位计算：

每一步计算总和（数组当前位 + k当前位 + 进位），将个位数存入结果，十位数作为新的进位。

3、处理剩余情况：

当数组遍历完毕（i < 0）或k减为0后，若仍有进位（如999 + 1 = 1000），继续将进位的每一位添加到结果。

4、结果逆序：

由于结果是按“最低位→最高位”存储的（如1234会先存储为[4,3,2,1]），逆序后得到正确的数组形式[1,2,3,4]。

复杂度分析

- 时间复杂度：O(max(n, log k))，其中n是数组num的长度，log k是整数k的位数（以10为底）。循环次数取决于数组长度、k的位数和进位的位数，取最大值。

- 空间复杂度：O(max(n, log k) + 1)，结果数组的长度最多为max(n, log k) + 1（考虑进位导致的长度增加，如999 + 1的结果比原数多一位）。

该方法高效模拟了加法过程，无需将数组转换为大整数（避免溢出风险），直接通过逐位计算得到结果，适合题目中数组长度和k的范围约束。