# 题目

给定一组非负整数 nums，重新排列每个数的顺序（每个数不可拆分）使之组成一个最大的整数。

注意：输出结果可能非常大，所以你需要返回一个字符串而不是整数。

示例 1：

输入：nums = [10,2]

输出："210"

示例 2：

输入：nums = [3,30,34,5,9]

输出："9534330"

提示：

1 <= nums.length <= 100

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

# 分析

## 方法一：排序

思路：

上述代码的实现思路主要是将整数数组转换为字符串数组，并根据特定的比较规则对这些字符串进行排序，以构造出由这些数字组成的最大整数。具体步骤如下：

1、初始化字符串数组：首先，将输入的整数数组 nums 中的每个元素转换为字符串，并将这些字符串存储在一个新的 std::vector<std::string> 类型的变量 numStrs 中。

2、定义比较函数：定义一个比较函数 compareForLargestNumber，该函数接受两个字符串参数 a 和 b，并返回 a + b 是否大于 b + a。这个比较函数用于确定在构建最大整数时 a 和 b 的相对顺序。如果 a + b 大于 b + a，则 a 应该排在 b 前面，反之亦然。

3、排序字符串数组：使用 std::sort 函数对 numStrs 进行排序，传入比较函数 compareForLargestNumber 作为排序依据。这样，排序后的 numStrs 将按照能够构建最大整数的顺序排列。

4、连接字符串：将排序后的字符串数组 numStrs 中的所有字符串连接起来，形成一个单独的字符串 result。

5、处理特殊情况：如果连接后的字符串 result 以 '0' 开头且长度大于 1（即原始数组中存在一个或多个零，并且零不是唯一元素），则返回字符串 "0"。这是因为在这种情况下，无法构造出非零的最大整数。

6、返回结果：最后，返回构造好的最大整数字符串 result。

通过这个过程，代码能够根据输入的整数数组，找出一种排列方式，使得这些整数连接而成的字符串表示的数值最大。

代码：

// 定义普通函数用于比较字符串，以构建最大的整数

bool compareForLargestNumber(const std::string& a, const std::string& b) {

// 比较ab和ba哪个更大

return a + b > b + a;

}

class Solution {

public:

std::string largestNumber(std::vector<int>& nums) {

// 将整数转换为字符串，并存储到新的vector中

std::vector<std::string> numStrs;

for (int num : nums) {

numStrs.push\_back(std::to\_string(num));

}

// 使用普通函数对字符串进行排序，以构建最大的整数

std::sort(numStrs.begin(), numStrs.end(), compareForLargestNumber);

// 将排序后的字符串连接起来

std::string result;

for (const std::string& numStr : numStrs) {

result += numStr;

}

// 如果结果以0开头，则直接返回"0"

if (result[0] == '0' && result.size() > 1) {

return "0";

}

return result;

}

};

或：

// 函数对象（仿函数）用于比较字符串，以构建最大的整数

struct CompareForLargestNumber {

bool operator()(const std::string& a, const std::string& b) const {

return a + b > b + a;

}

};

class Solution {

public:

std::string largestNumber(std::vector<int>& nums) {

// 将整数转换为字符串，并存储到新的vector中

std::vector<std::string> numStrs;

for (int num : nums) {

numStrs.push\_back(std::to\_string(num));

}

// 使用函数对象对字符串进行排序，以构建最大的整数

std::sort(numStrs.begin(), numStrs.end(), CompareForLargestNumber());

// 连接字符串并处理特殊情况

std::string result;

for (const std::string& numStr : numStrs) {

result += numStr;

}

if (result[0] == '0' && result.size() > 1) {

return "0";

}

return result;

}

};

或：

class Solution {

public:

std::string largestNumber(std::vector<int>& nums) {

// 将整数转换为字符串，并存储到新的vector中

std::vector<std::string> numStrs;

for (int num : nums) {

numStrs.push\_back(std::to\_string(num));

}

// 使用lambda表达式对字符串进行排序，以构建最大的整数

std::sort(numStrs.begin(), numStrs.end(), [](const std::string& a, const std::string& b) {

return a + b > b + a;

});

// 连接字符串并处理特殊情况

std::string result;

for (const std::string& numStr : numStrs) {

result += numStr;

}

if (result[0] == '0' && result.size() > 1) {

return "0";

}

return result;

}

};

或：

class Solution {

public:

std::string largestNumber(std::vector<int>& nums) {

// 将整数转换为字符串，并存储到新的vector中

std::vector<std::string> numStrs;

for (int num : nums) {

numStrs.push\_back(std::to\_string(num));

}

// 使用lambda表达式定义比较函数，并赋值给变量

auto compareForLargestNumber = [](const std::string& a, const std::string& b) {

return a + b > b + a;

};

// 使用变量对字符串进行排序，以构建最大的整数

std::sort(numStrs.begin(), numStrs.end(), compareForLargestNumber);

// 连接字符串并处理特殊情况

std::string result;

for (const std::string& numStr : numStrs) {

result += numStr;

}

if (result[0] == '0' && result.size() > 1) {

return "0";

}

return result;

}

};