# 题目

给你一个整数数组 nums，请你将该数组升序排列。

示例 1：

输入：nums = [5,2,3,1]

输出：[1,2,3,5]

示例 2：

输入：nums = [5,1,1,2,0,0]

输出：[0,0,1,1,2,5]

提示：

1 <= nums.length <= 50000

-50000 <= nums[i] <= 50000

# 分析

## 方法一：桶排序

class Solution {

public:

// 桶排序 通过（最快）52ms

vector<int> bucketSort(vector<int>& nums) {

if (nums.empty()) return {};

int low = \*min\_element(nums.begin(), nums.end());

int high = \*max\_element(nums.begin(), nums.end());

int n = high - low + 1;

vector<int> buckets(n);

for (auto x : nums) ++buckets[x - low];

vector<int> res;

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

for (int j = 0; j < buckets[i]; ++j) {

res.push\_back(i + low);

}

}

return res;

}

}

或：

class Solution {

public:

void bucketSort(vector<int>& nums) {

int minVal = \*min\_element(nums.begin(), nums.end());

int maxVal = \*max\_element(nums.begin(), nums.end());

// 计算桶的数量

int bucketCount = maxVal - minVal + 1;

// 初始化桶

vector<int> buckets(bucketCount, 0);

// 统计每个元素的出现次数

for (int num : nums) {

buckets[num - minVal]++;

}

// 将桶中的元素按顺序放回原数组

int index = 0;

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

while (buckets[i] > 0) {

nums[index++] = i + minVal;

buckets[i]--;

}

}

}

vector<int> sortArray(vector<int>& nums) {

bucketSort(nums);

return nums;

}

};

## 方法二：归并排序

// 归并排序 通过 232ms

vector<int> mergeSort(vector<int>& nums, int l, int r) {

if (l > r) return {};

if (l == r) return {nums[l]};

vector<int> res;

int m = l + (r - l) / 2;

auto ln = mergeSort(nums, l, m);

auto rn = mergeSort(nums, m + 1, r);

int i = 0;

int j = 0;

while (i < ln.size() && j < rn.size()) {

if (ln[i] <= rn[j]) {

res.push\_back(ln[i++]);

} else {

res.push\_back(rn[j++]);

}

}

while (i < ln.size()) res.push\_back(ln[i++]);

while (j < rn.size()) res.push\_back(rn[j++]);

return res;

}

## 方法三：快速排序

// 快速排序 通过 72ms

void quickSort(vector<int>& nums, int l, int r) {

if (l >= r) return;

int m = l + (r - l) / 2;

swap(nums[r], nums[m]);

int t = l;

for (int i = l; i < r; ++i) {

if (nums[i] < nums[r]) {

swap(nums[t++], nums[i]);

}

}

swap(nums[t], nums[r]);

quickSort(nums, l, t - 1);

quickSort(nums, t + 1, r);

}

## 方法四：三路快排

// 三路快排 68ms

void quickSortV2(vector<int>& nums, int l, int r) {

if (l >= r) return;

int t = nums[l + (r - l) / 2];

int tl = l;

int tr = r;

for (int i = tl; i <= tr;) {

if (nums[i] == t) {

++i;

} else if (nums[i] < t) {

swap(nums[i++], nums[tl++]);

} else {

swap(nums[i], nums[tr--]);

}

}

quickSortV2(nums, l, tl - 1);

quickSortV2(nums, tr + 1, r);

}

## 方法五：希尔排序

// 希尔排序 通过 80ms

void shellSort(vector<int>& nums) {

for (int gap = nums.size() / 2; gap > 0; gap /= 2) {

for (int i = gap; i < nums.size(); ++i) {

for (int j = i; j - gap >= 0 && nums[j - gap] > nums[j]; j -= gap) {

swap(nums[j - gap], nums[j]);

}

}

}

}

// 堆排序 通过 80ms

void adjust(vector<int>& nums, int p, int s) {

while (2 \* p + 1 < s) {

int c1 = 2 \* p + 1;

int c2 = 2 \* p + 2;

int c = (c2 < s && nums[c2] > nums[c1]) ? c2 : c1;

if (nums[c] > nums[p]) swap(nums[c], nums[p]);

else break;

p = c;

}

}

void heapSort(vector<int>& nums) {

for (int i = nums.size() / 2; i >= 0; --i) {

adjust(nums, i, nums.size());

}

for (int i = nums.size() - 1; i > 0; --i) {

swap(nums[0], nums[i]);

adjust(nums, 0, i);

}

}

## 方法六：冒泡排序

// 冒泡排序 超时

void bubbleSort(vector<int>& nums) {

for (int i = 0; i < nums.size(); ++i) {

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

if (nums[i] < nums[j]) {

swap(nums[i], nums[j]);

}

}

}

}

## 方法七：插入排序

// 插入排序 超时

void insertSort(vector<int>& nums) {

for (int i = 0; i < nums.size(); ++i) {

for (int j = i; j - 1 >= 0 && nums[j - 1] > nums[j]; --j) {

swap(nums[j - 1], nums[j]);

}

}

}

## 方法八：选择排序

// 选择排序 超时

void selectSort(vector<int>& nums) {

for (int i = 0; i < nums.size(); ++i) {

int min\_ind = i;

for (int j = i; j < nums.size(); ++j) {

if (nums[j] < nums[min\_ind]) {

min\_ind = j;

}

}

swap(nums[i], nums[min\_ind]);

}

}

vector<int> sortArray(vector<int>& nums) {

return bucketSort(nums);

}

};