CSP-J中，排序算法考点

一、排序算法基础定义与核心性质（必考基础）

1. 核心概念

- 排序定义：将一组无序的数据，按照指定规则（升序/降序）重新排列为有序序列的过程，是CSP-J基础算法的核心模块；
- 关键术语：稳定性（排序后，值相等的元素保持原有相对顺序）、时间复杂度（排序过程中元素比较/移动的操作次数）、空间复杂度（排序所需额外存储空间）；
- CSP-J适配场景：单独考查排序模板调用、作为后续算法的前置步骤（如二分查找、去重、贪心问题）、选择题考查算法性质对比。

2. 核心性质（CSP-J解题关键）

- 稳定性区分：稳定排序（冒泡、插入）、不稳定排序（选择、快速、希尔），CSP-J偶考稳定性判断，无需深入理解原理，记准结论即可；
- 效率优先级：CSP-J中优先使用STL排序函数（sort），其底层为快速排序优化版本，兼顾效率与便捷性，手写排序仅考查基础模板；
- 适用场景差异：不同排序算法适配不同数据量、数据分布，需根据题干条件选择（如小数据量可用冒泡，大数据量用sort）。

3. 考情说明

核心考查两大方向：一是STL sort函数的调用（必考，占比最高）；二是基础排序算法（冒泡、选择、插入）的手写模板与性质；快速排序考查基础思路，不考复杂优化；归并、堆排序等为S组考点，J组不涉及。

二、CSP-J适配的排序算法（重点掌握）

CSP-J排序考点聚焦“实用型”，分为「STL函数调用」和「基础手写排序」两类，无需掌握复杂排序算法，覆盖这两类即可应对所有考点。

1. STL sort函数（首选，必考）

- 核心特点：C++ STL算法库中的排序函数，底层基于“快速排序+插入排序+堆排序”混合实现（introsort），平均时间复杂度O(nlogn)，不稳定排序；
- 适配场景：所有基础排序场景，从小数据量（n≤100）到中大数据量（n≤1e5）均适用，是CSP-J解题的首选方案；

- 注意事项：需包含<algorithm>头文件，默认升序排序，可自定义排序规则（降序、结构体排序）。

2. 基础手写排序（必考模板，3种）

- 冒泡排序：简单易写，基于“相邻元素两两比较交换”，时间复杂度O(n^2)，稳定排序，适合小数据量；
- 选择排序：基于“每次选最小/最大值放到对应位置”，时间复杂度O(n^2)，不稳定排序，交换次数少于冒泡；
- 插入排序：基于“将元素插入已排序序列”，时间复杂度O(n^2)，稳定排序，数据接近有序时效率极高。

3. 补充说明（避坑重点）

- 不考内容：归并排序、堆排序、希尔排序、快速排序手写（仅考基础思路）、计数排序等高级排序；
- 替代方案：若题目禁止使用sort函数（极少数情况），用手写冒泡/插入排序即可，无需追求高效算法。

三、排序算法核心模板（CSP-J必考）

1. STL sort函数（高频必考，重中之重）

核心考点：基础升序/降序排序、结构体排序、数组/vector排序，是CSP-J排序题的主流解法，代码简洁易记。

```
#include <iostream>
#include <vector>
#include <algorithm> // sort函数必备头文件
using namespace std;

// 自定义降序排序规则（函数版）
bool cmp(int a, int b) {
    return a > b; // 降序：a比b大则排在前面
}

// 结构体排序（真题高频场景）
struct Student {
    string name;
    int score;
    // 自定义排序规则（按分数升序，分数相同按姓名字典序升序）
    bool operator<(const Student& other) const {
        if (score != other.score) {
            return score < other.score;
```

```
        }
        return name < other.name;
    }
};

int main() {
    // 1. 数组排序（升序，默认）
    int arr[] = {3, 1, 4, 1, 5, 9, 2, 6};
    int n = sizeof(arr) / sizeof(arr[0]);
    sort(arr, arr + n); // 排序范围：[起始地址，结束地址)
    cout << "数组升序排序：";
    for (int x : arr) cout << x << " ";
    cout << endl;

    // 2. 数组降序排序（自定义cmp）
    sort(arr, arr + n, cmp);
    cout << "数组降序排序：";
    for (int x : arr) cout << x << " ";
    cout << endl;

    // 3. vector排序
    vector<int> nums = {3, 1, 4, 1, 5, 9, 2, 6};
    sort(nums.begin(), nums.end()); // 升序
    cout << "vector升序排序：";
    for (int x : nums) cout << x << " ";
    cout << endl;

    // 4. 结构体排序
    vector<Student> stu = {"Alice", 85, "Bob", 92, "Charlie", 85};
    sort(stu.begin(), stu.end()); // 调用结构体内部的<运算符
    cout << "结构体排序（分数升序）：" << endl;
    for (auto& s : stu) {
        cout << s.name << " " << s.score << endl;
    }

    return 0;
}
```

2. 基础手写排序模板（3种，必考）

核心考点：手写模板的正确性，侧重冒泡、插入排序，选择排序作为补充，需熟练默写，应对“禁止使用STL”的场景（极少数）。

```
#include <iostream>
#include <vector>
```

```cpp
using namespace std;

// 1. 冒泡排序（升序，稳定）
void bubbleSort(vector<int>& nums) {
    int n = nums.size();
    for (int i = 0; i < n - 1; i++) { // 外层循环：控制排序轮次
        bool swapped = false; // 优化：无交换则说明已有序，提前退出
        for (int j = 0; j < n - 1 - i; j++) { // 内层循环：相邻元素比较交换
            if (nums[j] > nums[j + 1]) {
                swap(nums[j], nums[j + 1]);
                swapped = true;
            }
        }
        if (!swapped) break;
    }
}

// 2. 选择排序（升序，不稳定）
void selectSort(vector<int>& nums) {
    int n = nums.size();
    for (int i = 0; i < n - 1; i++) { // 外层循环：确定当前位置的最小值
        int minIdx = i; // 记录最小值下标
        for (int j = i + 1; j < n; j++) { // 内层循环：寻找最小值
            if (nums[j] < nums[minIdx]) {
                minIdx = j;
            }
        }
        swap(nums[i], nums[minIdx]); // 将最小值放到当前位置
    }
}

// 3. 插入排序（升序，稳定）
void insertSort(vector<int>& nums) {
    int n = nums.size();
    for (int i = 1; i < n; i++) { // 外层循环：待插入元素（从第2个元素开始）
        int temp = nums[i]; // 保存待插入元素
        int j = i - 1; // 内层循环：已排序序列的末尾
        while (j >= 0 && nums[j] > temp) { // 寻找插入位置
            nums[j + 1] = nums[j]; // 元素后移
            j--;
        }
        nums[j + 1] = temp; // 插入待插入元素
    }
}

int main() {
    vector<int> nums = {3, 1, 4, 1, 5, 9, 2, 6};
```

```
    vector<int> nums1 = nums, nums2 = nums, nums3 = nums;

    bubbleSort(nums1);
    selectSort(nums2);
    insertSort(nums3);

    cout << "冒泡排序结果: ";
    for (int x : nums1) cout << x << " ";
    cout << endl;

    cout << "选择排序结果: ";
    for (int x : nums2) cout << x << " ";
    cout << endl;

    cout << "插入排序结果: ";
    for (int x : nums3) cout << x << " ";
    cout << endl;

    return 0;
}

```

3. 快速排序基础思路（仅考选择，不考手写）

CSP-J仅考查快速排序的核心思路，无需手写完整模板，理解即可应对选择题。

- 核心思想：分治法，选择一个“基准值”，将数组分为两部分（小于基准值、大于基准值），再递归排序两部分；
- 关键性质：平均时间复杂度O(nlogn)，最坏时间复杂度O(n^2)（基准值选择不当），不稳定排序；
- 备注：STL sort函数已优化快速排序的缺陷，无需手动优化，直接调用即可。

四、CSP-J排序算法避坑与备考要点

1. 易错点

- 头文件遗漏：使用sort函数必须包含<algorithm>头文件，否则编译错误；
- 排序范围错误：sort函数的结束地址是“待排序最后一个元素的下一个地址”（如数组arr[0..n-1]，排序范围为arr到arr+n）；
- 结构体排序错误：自定义排序规则时，需注意运算符重载或cmp函数的逻辑，避免排序混乱；
- 稳定性误区：无需纠结排序的稳定性，除非题目明确要求，否则优先用sort函数。

2. 备考优先级

- 必掌握：STL sort函数（基础排序、结构体排序）、冒泡排序、插入排序模板；

- 选掌握：选择排序模板、快速排序基础思路；
- 不考内容：归并、堆、希尔等高级排序，快速排序手写优化、计数排序等。

3. 刷题建议

重点刷两类题：一是sort函数的应用（结构体排序、自定义规则排序），二是基础手写排序模板题；每类刷3-5道J组真题，结合二分查找、去重等场景练习，熟练掌握排序的前置/后置应用，避免孤立刷题。