排序算法

冒泡排序

定义

冒泡排序是一种基础的交换排序算法 。其核心原理是通过对待排序序列进行相邻元素的比较与交换，使关键字较大（或较小）的元素逐渐从序列前端移向后端 。

算法形式化描述

对于一个含有 n 个元素的序列 $A = \{a_1, a_2, \ldots, a_n\}$，冒泡排序的执行过程如下：

1. 比较与交换：在每一轮迭代中，依次比较相邻的两个元素 a_j 和 a_{j+1}。若满足条件 $a_j > a_{j+1}$（升序排序）。
2. 局部有序化：每完成一轮迭代，序列中的最大值将被移动至该轮参与比较的最后位置。经过第 i 轮迭代后，序列末尾的 i 个元素达到全局有序状态 。
3. 终止条件：总计进行 $n-1$ 轮迭代，或在某一轮迭代中未发生任何元素交换（即序列已达到有序状态）时停止。

算法复杂度分析

- 时间复杂度：
  - 最坏情况：当输入序列为逆序时，复杂度为 $O(n^2)$。
  - 最好情况：当输入序列已完全有序且使用了 `flag` 优化时，复杂度为 $O(n)$。
  - 平均情况：$O(n^2)$。
- 空间复杂度：$O(1)$，属于原地排序，仅需常数级的辅助空间用于元素交换。
- 稳定性：稳定排序。由于相同大小的元素在比较时不会触发交换（即 $a_j = a_{j+1}$ 时不移动），因此它们在排序前后的相对顺序保持不变。

图解

1. 案例设定

- 待排序序列：[5, 2, 8, 1, 9]
- 目标：升序排列（从小到大）
- 核心逻辑：相邻元素两两比较，若左边大于右边则交换。

2. 第一轮迭代

这是最关键的一轮，目标是将序列中的最大值移动到最后。

1. 比较 5 和 2：$5 > 2$，交换 $\rightarrow$ `[2, 5, 8, 1, 9]`
2. 比较 5 和 8：$5 < 8$，不交换 $\rightarrow$ `[2, 5, 8, 1, 9]`
3. 比较 8 和 1：$8 > 1$，交换 $\rightarrow$ `[2, 5, 1, 8, 9]`
4. 比较 8 和 9：$8 < 9$，不交换 $\rightarrow$ `[2, 5, 1, 8, 9]`

本轮结果：最大值 9 已固定在末尾。

3. 第二轮迭代

由于 9 已经固定，本轮只需处理前 4 个元素。

1. 比较 2 和 5：$2 < 5$，不交换 → `[2, 5, 1, 8, 9]`
2. 比较 5 和 1：$5 > 1$，交换 → `[2, 1, 5, 8, 9]`
3. 比较 5 和 8：$5 < 8$，不交换 → `[2, 1, 5, 8, 9]`

本轮结果：次大值 8 已固定。

4. 第三轮迭代

处理剩余的 3 个元素。

1. 比较 2 和 1：$2 > 1$，交换 → `[1, 2, 5, 8, 9]`
2. 比较 2 和 5：$2 < 5$，不交换 → `[1, 2, 5, 8, 9]`

本轮结果：5 已固定。

5. 第四轮迭代

只剩最后两个元素。

1. 比较 1 和 2：$1 < 2$，不交换 → `[1, 2, 5, 8, 9]`

最终结果：序列完全有序。

冒泡排序标准代码模板

```cpp
#include <iostream>
#include <algorithm> // 用于使用 swap 函数
using namespace std;

int main()
{
    int n;
    int a[1005]; // 假设最多1000个数

    // 1. 输入数据
    cin >> n;
    for(int i = 0; i < n; i++) {
        cin >> a[i];
    }

    // 2. 冒泡排序核心代码
    for(int i = 0; i < n - 1; i++) {       // 控制排序轮数
        for(int j = 0; j < n - 1 - i; j++) { // 每一轮比较相邻的两个数
            if(a[j] > a[j+1]) {           // 如果前面的比后面大，则交换
                swap(a[j], a[j+1]);
            }
        }
    }

    // 3. 输出结果
    for(int i = 0; i < n; i++) {
        cout << a[i] << (i == n - 1 ? "" : " ");
    }
    cout << endl;

    return 0;
}
```

优化冒泡排序标准代码模板

```cpp
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

/**
 * 优化版冒泡排序算法
 * @param a 待排序的数组
 * @param n 数组长度
 */
void bubbleSort(int a[], int n) {
    // 外层循环控制排序轮数，共需要n-1轮
    for (int i = 0; i < n - 1; i++) {
        // swapped标志位，用于优化：如果一轮中没有交换，说明已完全有序
        bool swapped = false;

        // 内层循环进行相邻元素比较交换
        // 注意：每轮排序后，最后的i+1个元素已有序，所以比较范围逐渐减小
        for (int j = 0; j < n - 1 - i; j++) {
            // 如果前一个元素大于后一个元素，则交换（升序排序）
            if (a[j] > a[j + 1]) {
                swap(a[j], a[j + 1]);
                swapped = true;  // 标记发生了交换
            }
        }

        // 优化：如果本轮没有发生交换，说明数组已完全有序，提前结束排序
        if (!swapped) break;
    }
}

int main() {
    // 关闭C与C++输入输出流的同步，提高输入输出效率
    ios::sync_with_stdio(false);
    // 解除cin与cout的绑定，进一步提高效率
    cin.tie(0);

    int n;
    // 读取数组长度，如果读取失败则直接退出
    if (!(cin >> n)) return 0;

    // 定义数组，假设最大长度为10005（根据题目要求）
    int data[10005];
    // 读取数组元素
    for (int i = 0; i < n; i++) {
        cin >> data[i];
    }

    // 调用优化版冒泡排序
    bubbleSort(data, n);

    // 输出排序结果
    // 注意：最后一个元素后面不加空格，符合常见输出格式要求
    for (int i = 0; i < n; i++) {
        cout << data[i] << (i == n - 1 ? "" : " ");
    }
    cout << endl;

    return 0;
```

```
}
```

题目

题目名称：排个序

题目描述给定一个长度为 n 的数组 a。给定一个长度为 m 的互不相同的数组 p，意味着你可以交换 $a[p_i]$ 和 $a[p_i+1]$ 任意次。请确定是否可以用仅允许的交换方式使得 a 数组非严格递减。

输入描述

- 第一行输入一个 n 和 m。
- 第二行输入 n 个整数 $a_1, a_2, \ldots, a_n$。
- 第三行输入 m 个整数 $p_1, p_2, p_3, \ldots, p_m$。
- 数据范围限制：$1 \leq m < n \leq 10^3, 1 \leq a_i \leq n, 1 \leq p_i < n$。

输出描述 如果可以输出 YES，否则输出 NO。

输入输出样例 示例 1

- 输入：

```
3 2
3 2 1
1 2
```

- 输出：

```
YES
```

运行限制

- 最大运行时间：1s
- 最大运行内存：128M

```cpp
#include <iostream>
#include <algorithm>
#include <vector>
using namespace std;

bool can_sawp[1005];
int a[1005];


int main()
{
  ios::sync_with_stdio(false);
  cin.tie(0);

  int n,m;
  if (!(cin >> n >> m)) return 0;

  for(int i=1;i<=n;i++){
    cin >> a[i];
  }

  for(int i=0;i<m;i++){
```

```
    int p;
    cin >> p;
    can_sawp[p] = true;
  }

  for(int i=1;i<n;i++){
    for(int j=1;j<=n-1;j++){
      if(a[j]>a[j+1]){
        if(can_sawp[j]){
          swap(a[j],a[j+1]);
        }
      }
    }
  }

  bool ok = true;
  for(int i=1;i<n;i++){
    if(a[i]>a[i+1]){
      ok = false;
      break;
    }
  }

  if(ok) cout << "YES" << endl;
  else cout << "NO" << endl;

  return 0;
}
```

选择排序

定义

选择排序的基本思想是：每一轮从待排序的数据元素中选出最小（或最大）的一个元素，存放在序列的起始位置，直到全部待排序的数据元素排完 。

算法执行步骤

假设我们要将一个长度为 n 的数组按升序（从小到大）排列：

- 初始化：从数组的第一个位置（索引 0）开始 。
- 寻找最小元素：在当前未排序的部分（从索引 i 到 $n-1$）中，通过遍历找到最小的元素 。
- 交换位置：将找到的最小元素与当前未排序部分的第一个元素（位置 i）进行交换 。
- 重复迭代：将起始位置向后移动一位（$i = i+1$），重复上述寻找和交换的过程，直到 $i = n-1$ 为止 。

算法特性分析

根据蓝桥杯的竞赛要求和性能考量 ，你需要了解该算法的以下特性：

特性	说明
时间复杂度	无论数据初始状态如何，都需要进行两层循环，复杂度始终为 $O(n^2)$ 。
空间复杂度	只需要一个额外的空间用于交换元素，属于原地排序，复杂度为 $O(1)$ 。

特性	说明
稳定性	不稳定。在交换过程中，可能会改变相同数值元素的相对顺序。
适用场景	适用于数据量较小（如 $n < 1000$）的情况。在蓝桥杯编程大题中，若遇到大数据量压力测试，需优先考虑更高效率的算法。

图解

将一组数据 [29, 10, 14, 37, 13] 按从小到大进行排序。

选择排序的核心是：“每轮从剩下的数字里选出最小的一个，放到前面。”

- 第一轮：
  - 在 [29, 10, 14, 37, 13] 中寻找最小值。
  - 发现 10 是最小的，将它与第一个位置的 29 交换。
  - 结果： [10, 29, 14, 37, 13]（10 已固定）。
- 第二轮：
  - 在剩下的 [29, 14, 37, 13] 中寻找最小值。
  - 发现 13 是最小的，将它与第二个位置的 29 交换。
  - 结果： [10, 13, 14, 37, 29]（13 已固定）。
- 第三轮：
  - 在剩下的 [14, 37, 29] 中寻找最小值。
  - 发现 14 本身就在正确位置，无需移动。
  - 结果： [10, 13, 14, 37, 29]（14 已固定）。
- 第四轮：
  - 在剩下的 [37, 29] 中寻找最小值。
  - 发现 29 是最小的，将它与第四个位置的 37 交换。
  - 结果： [10, 13, 14, 29, 37]（排序完成）。

```cpp
#include <iostream>
#include <algorithm>

using namespace std;

int main() {
    int arr[] = {29, 10, 14, 37, 13};
    int n = 5;  // 数组长度

    // 选择排序主循环
    for (int i = 0; i < n - 1; i++) {        // i表示当前已排序部分的末尾位置
        int min_idx = i;                     // 假设当前位置i的元素是最小值

        // 在未排序部分中寻找真正的最小值
        for (int j = i + 1; j < n; j++) {    // j遍历未排序部分
            if (arr[j] < arr[min_idx]) {     // 发现更小的元素
                min_idx = j;                 // 更新最小值索引
            }
        }

        // 将找到的最小值与当前位置交换
        if (min_idx != i) {                  // 如果最小值不是当前位置
            swap(arr[i], arr[min_idx]);      // 交换两个位置的元素
        }
```

```
    }

    // 输出排序后的数组
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }

    return 0;
}
```

选择排序标准代码模板

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

using namespace std;

int main() {
    int n;
    // 读取数组大小，如果读取失败则直接结束程序
    if (!(cin >> n)) return 0;

    // 创建动态数组存储n个整数
    vector<int> a(n);
    // 读取n个整数到数组中
    for (int i = 0; i < n; i++) {
        cin >> a[i];
    }

    // 选择排序算法开始
    // 外层循环：每次确定一个位置的最小值
    for (int i = 0; i < n - 1; i++) {
        int min_idx = i;  // 假设当前位置为最小值所在位置

        // 内层循环：在未排序部分中寻找真正的最小值
        for (int j = i + 1; j < n; j++) {
            if (a[j] < a[min_idx]) {
                min_idx = j;  // 更新最小值的位置
            }
        }

        // 如果找到的最小值不在当前位置，则交换
        if (min_idx != i) {
            swap(a[i], a[min_idx]);  // 将最小值放到正确位置
        }
    }

    // 输出排序后的数组
    for (int i = 0; i < n; i++) {
        // 最后一个元素后不输出空格
        cout << a[i] << (i == n - 1 ? "" : " ");
    }
    cout << endl;  // 换行

    return 0;
}
```

插入排序

定义

插入排序（Insertion Sort）的工作原理非常类似于人们手动整理扑克牌的过程。

在算法执行过程中，数组被划分为两个部分：

- 已排序部分：最初只包含数组的第一个元素。
- 未排序部分：包含剩下的所有元素。

算法执行步骤

根据蓝桥杯对“算法灵活运用能力”的考查要求 ，你需要深刻理解以下动态逻辑：

- 从第一个元素开始，该元素可以认为已经被排序。
- 取出下一个元素（作为待插入的关键值 key），在已经排序的元素序列中从后向前扫描。
- 比较与移动：如果已排序序列中的元素大于 key，则将该元素移到下一位置。
- 插入：重复步骤 3，直到找到已排序的元素小于或者等于 key 的位置，将 key 插入到该位置后。
- 循环：重复步骤 2~4 直到所有元素处理完毕 。

算法特性评估

在蓝桥杯这种“侧重效率”的竞赛中 ，了解算法的性能是拿高分的关键：

特性	描述	竞赛意义
时间复杂度	平均与最坏情况均为 $O(n^2)$	仅适用于 n 较小（通常 $n < 5000$）的场景 。
空间复杂度	$O(1)$	原地排序，极其节省内存，符合 4G 内存限制要求 。
稳定性	稳定	相同数值的相对位置不会改变，这在某些结构体排序题中很重要 。
最佳情况	如果数组已基本有序，复杂度接近 $O(n)$	在处理“几近有序”的数据时，效率远高于选择排序 。

插入排序标准代码模板

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

using namespace std;

int main() {
    int n;
    // 读取数组长度，输入失败则退出程序
    if (!(cin >> n)) return 0;

    // 创建动态数组存储n个元素
    vector<int> a(n);
    // 读取数组元素
    for (int i = 0; i < n; i++) {
        cin >> a[i];
    }
```

```
    // 插入排序算法
    // 从第二个元素开始（索引1），因为单个元素是有序的
    for (int i = 1; i < n; i++) {
        int key = a[i];     // 当前待插入的元素
        int j = i - 1;      // 已排序部分的最后一个元素的索引

        // 将比key大的元素向右移动，为key腾出位置
        while (j >= 0 && a[j] > key) {
            a[j + 1] = a[j];  // 将元素向右移动
            j--;              // 向左移动比较指针
        }

        // 在正确位置插入key
        a[j + 1] = key;
    }

    // 输出排序后的数组
    for (int i = 0; i < n; i++) {
        // 控制输出格式：最后一个元素后不加空格
        cout << a[i] << (i == n - 1 ? "" : " ");
    }
    cout << endl;

    return 0;
}
```

std::sort

基础用法模板

这是比赛中最常用的方式，只需一行代码即可完成升序排列。

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

using namespace std;

int main() {
    int n;
    // 读取数组长度，输入失败则退出程序
    if (!(cin >> n)) return 0;

    // 创建动态数组存储n个元素
    vector<int> a(n);
    // 读取数组元素
    for (int i = 0; i < n; i++) cin >> a[i];

    // 使用C++标准库的sort函数进行排序
    // a.begin()返回数组开始位置的迭代器，a.end()返回结束位置的迭代器
    // sort函数将对这个范围内的元素进行升序排序
    sort(a.begin(), a.end());

    // 输出排序后的数组
    for (int i = 0; i < n; i++) {
        // 控制输出格式：元素之间用空格分隔，最后一个元素后不加空格
        cout << a[i] << (i == n - 1 ? "" : " ");
    }
    return 0;
```

```
}
```

结构体自定义排序

```cpp
#include <iostream>
#include <algorithm>
#include <vector>
#include <string>

using namespace std;

// 定义学生结构体
struct Student {
    string name;   // 学生姓名
    int score;     // 学生分数
    int id;        // 学生学号（用于排序时的辅助标识）
};

// 自定义比较函数，用于排序规则
// 返回 true 表示 x 应该排在 y 前面
bool cmp(Student x, Student y) {
    // 首先按分数降序排序（分数高的排在前面）
    if (x.score != y.score) {
        return x.score > y.score;  // 分数不同时，分数高的在前
    }
    // 分数相同时，按学号升序排序（学号小的排在前面）
    return x.id < y.id;
}

int main() {
    int n;  // 学生数量
    cin >> n;

    // 创建学生向量
    vector<Student> sts(n);

    // 读取每个学生的信息
    for (int i = 0; i < n; i++) {
        cin >> sts[i].name >> sts[i].score >> sts[i].id;
    }

    // 使用自定义比较函数进行排序
    // sort函数会根据cmp函数定义的规则对sts进行排序
    sort(sts.begin(), sts.end(), cmp);

    // 输出排序后的学生信息
    for (const auto& s : sts) {
        // 使用范围for循环，const auto&避免拷贝
        cout << s.name << " " << s.score << " " << s.id << endl;
    }
    return 0;
}
```