排序与逆序对

从基础排序、分治排序到严格弱序与逆序对计数

HGNU-ACM 实验室算法组

配套：基础排序与桶排序 + 快速/归并排序 + 逆序对 + 多关键字 + 4 道真题精讲





本章目录（前半）

基础排序思路、稳定性与桶排序

快速排序的分治划分与 std::sort

归并排序与二路归并

逆序对与归并分治计数

承接第 1 章《复杂度与前缀和、差分、双指针》：先搞懂 $O(n^2)$ 排序为何超时，再掌握桶排序、快速排序、归并排序与 $O(n \log n)$ 统计逆序对。

本章目录（后半）

结构体多关键字排序与严格弱序

四道排序与逆序对真题精讲

常见错误、检查清单与练习安排

后半章学会结构体多关键字排序与严格弱序规范，通过 4 道难度分层的真题实战，并用检查清单与练习路线完成闭环。

先想一个生活场景：为什么要先学会“排好队”？

场景：给全校 n 名同学按身高或成绩排成一排

第 1 章学过：在有序序列里可以用双指针和前缀和快速统计；如果序列本身是乱的，很多高效办法都用不上。

- 假如手里只有 5 张卡片 `[5，2，4，2，1]`，我们一眼就能按从小到大排成 `[1，2，2，4，5]`。
- 如果数据变成 $n = 10^5$ 个整数，肉眼和手写暴力循环就都不灵了：排序既要算得快，有时还要保住原本的先后顺序。
- 写排序程序前先回答三个问题：按什么规则比大小、相等的元素怎么放、数据范围 n 有多大。

本页要点

排序不是“把数换来换去”这么简单：小数据能用朴素方法，大数据必须选 $O(n \log n)$ 或桶排序。

三种基础 $O(n^2)$ 排序思路对照

排序方法	每轮在做什么（大白话）	最坏步数	是否稳定
冒泡排序	相邻两数逆序就交换，把最大值“冒”到末尾	$O(n^2)$	稳定
选择排序	每轮从剩下元素里挑最小值，直接换到最前面	$O(n^2)$	不稳定
插入排序	像抓扑克牌，把新牌插进前面已经排好序的牌里	$O(n^2)$	稳定

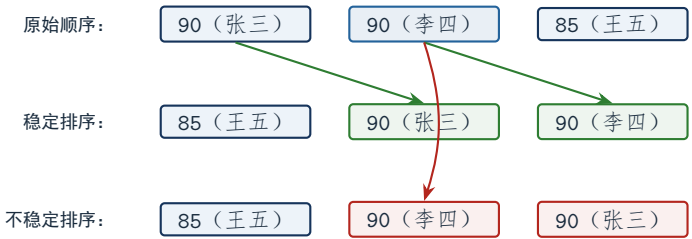
- 冒泡与插入：只在相邻元素大小关系错乱时移动，遇到相等元素不交换。
- 选择排序：跨过中间一堆数直接长距离交换，可能把原本靠前的相等元素甩到后面去。

本页要点

三种基础排序都要做两层循环：外层 n 轮、内层最多 n 次，总比较次数都是 $O(n^2)$ 级别。

什么是“稳定排序”？TikZ 状态图

新术语解释——稳定性（**Stability**）：如果两个元素排序依据的数值相等，排序后它们原本的前后相对顺序保持不变，就叫稳定排序。



本页要点

同分时想保住“谁先交卷谁在前”，就需要稳定排序；选择排序跨步交换会破坏稳定性。

手算跟踪表：插入排序排 [5，2，4，2]

每轮取出下标 i 的数 $key = a[i]$，往前找比 key 大的数往右挪，直到遇到 $\le key$ 或到达下标 0：

轮次 i	待插入 key	比较条件成立？	挪动后数组	停止原因说明
1	2	真（$5 > 2$）	[2，5，4，2]	到达左端边界 $j < 0$，条件为假停止
2	4	真（$5 > 4$）	[2，4，5，2]	遇到 $2 \le 4$，条件为假停止
3	2（第 2 个）	真（$5 > 2, 4 > 2$）	[2，2，4，5]	遇到第一个 $2 \le 2$，停在它右边（稳定）
4	$i = 4$	假（$i < 4$ 为假）	[2，2，4，5]	外层循环终止，全部排好

本页要点

第 3 轮遇到相等的 2 就立刻停手不跨过去，所以插入排序天生是稳定排序。

痛点估算：$O(n^2)$ 为什么在 $n = 10^5$ 时必超时？

假设我们要排 n 个打乱的数字，第 1 章学过 1 秒内计算机最多执行约 10^8 次简单操作：

数据规模 n	$O(n^2)$ 约需步数	$O(n\log_2 n)$ 约需步数	1 秒红线评测结果
$n = 100$	10^4 次	6.6×10^2 次	两种都瞬间 AC
$n = 5000$	2.5×10^7 次	6.1×10^4 次	$O(n^2)$ 勉强能过
$n = 10^5$	10^{10} 次（约 100 秒）	1.7×10^6 次	$O(n^2)$ 必 **TLE**！
$n = 10^7$	10^{14} 次	2.3×10^8 次（偏紧）	值域小时用桶排序 $O(n+V)$

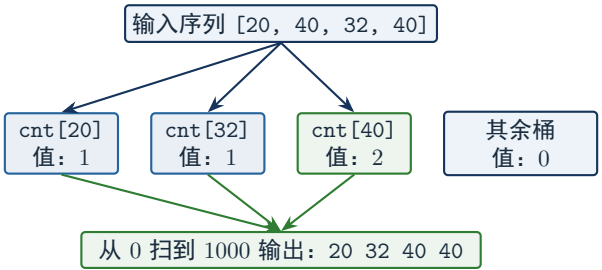
- 当 $n \geq 10^4$ 时就不要再写冒泡、选择或插入排序，直接换用快排或归并排序。
- 如果数值范围 V 很小（例如 $0 \leq x \leq 10^6$），还能用不用比较的计数桶排序做到 $O(n+V)$！

本页要点

看准 n 和值域 V：通用大数组用 $O(n\log n)$，整数值域很小用 $O(n+V)$ 桶排序。

小值域加速器：计数桶排序思路与 TikZ 图

生活引入：全校 10^5 张选票给 $1 \sim 1000$ 号候选人投票，不用把选票翻来覆去比大小，只要摆 1000 个投票箱，读一张往对应箱里加 1！



本页要点

桶排序不比较任何两个数的大小，直接把“数值”当成“数组下标”，总步数 $O(n+V)$。

桶排序模板代码（完整核心段）

```
int n, x;
if (!(cin >> n)) return 0;
vector<int> cnt(1001, 0);                    // 桶编号 0~1000，初值为 0
for (int i = 0; i < n; ++i) {
    cin >> x;
    ++cnt[x];                                // 对应编号桶加 1
}
for (int v = 0; v <= 1000; ++v) {
    while (cnt[v]--) cout << v << ' ';       // 出现几次输出几次；去重改 if(cnt[v
    ]>0)
}
cout << '\n';
```

为什么这样写

桶数组大小必须至少开到 $V_{\max}+1$（能装下下标 1000）；想顺便去重只要把 `while (cnt[v]--)` 改成 `if (cnt[v] > 0)` 即可。

桶排序上机验证（3 组数据 + 反事实追问）

使用完整程序 `code/bucket_demo.cpp`（读入 n 与 n 个 $0 \sim 1000$ 整数）进行上机测试：

- 普通数据：输入 `6 / 20 40 32 67 40 20` → 预期输出 `20 20 32 40 40 67`。
- 最小边界：输入 `1 / 0` → 预期输出 `0`（验证下标 0 没漏）。
- 最大与全等边界：输入 `3 / 1000 1000 1000` → 预期输出 `1000 1000 1000`。

反事实追问

如果输入范围改成包含负数 $-500 \sim 500$，直接写 `++cnt[x]` 会发生什么？
答：访问负下标导致越界 **RE**！只需在存入时统一加偏移量 `++cnt[x + 500]`，输出时打印 `v - 500` 即可。

本页要点

若值域高达 10^9（数组开不下），立刻放弃桶排序，改用 `std::sort`！

基础排序与桶排序易错对照

错误类型	常见错误写法	正确做法与原因
TLE（超时）	对 $n = 10^5$ 写冒泡或选择排序	改用 $O(n \log n)$ 的快排、归并或 `std::sort`
RE（越界）	值域 $0 \sim 1000$ 只开 `cnt(1000)`	必须开 `cnt(1001)`，覆盖下标 1000
RE / MLE	数值高达 10^9 硬开 `cnt[1000000001]`	内存爆掉，改用 `sort` 或离散化
WA（漏重/错序）	要求保留重复数却写了 `if(cnt[v]>0)`	保留重复写 `while(cnt[v]--)`，去重才写 `if`

本页要点

先看 n 再看值域 V：$n \le 10^5$ 用 $O(n \log n)$，值域 $V \le 10^6$ 且非负才直接开桶！

生活引入：体育课怎么快速按身高站成一排？

场景：全班 100 个同学乱糟糟站在操场上，想快速按身高从矮到高排队

如果像冒泡那样每次只比较相邻两人，跑完一轮才排好最后一个人，太慢了！

- 体育老师换了一个聪明的办法：先请出一位同学当标兵，喊一声口令：
  1. 比标兵矮的站到他左边去；
  2. 比标兵高的站到他右边去。
- 这一声口令下去，标兵左边的整拨人永远不会再跑到右边去，两边只要各自内部排队就行了！

本页要点

先找一个参照标准把大问题拆成“互不干扰的左右两半”，这就是快速排序的起点。

为什么比冒泡快？新术语“分治”与“基准值”

- 新术语——分治（**Divide and Conquer**）：“分而治之”，把一个长度 n 的大区间拆成两个更小的子区间，各自排好序后整个区间就自然有序。
- 新术语——基准值（**Pivot**）：用来做比较标准的那一个元素（就像刚才的体育课标兵）。

步数估算

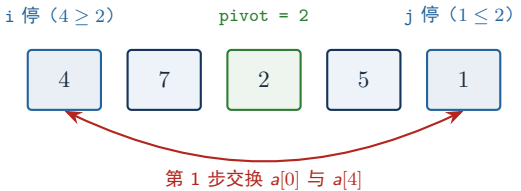
如果每次大致对半劈开，长度 $n = 10^5$ 的数组只需劈约 $\log_2 n \approx 17$ 层。每层总共扫描不超过 n 个元素，合计约 $17 \times 10^5 \approx 1.7 \times 10^6$ 步——比 $O(n^2)$ 的 10^{10} 步快了近一万倍！

本页要点

每次把问题规模劈成两半，只需要 $\log_2 n$ 层就能拆到长度为 1。

TikZ 图解：快排双指针划分（Partition）

以 [4，7，2，5，1] 为例，取中间位置基准 pivot = 2，左指针 i 找 ≥ 2，右指针 j 找 ≤ 2，停下后交换：



- 交换后变成 [1，7，2，5，4]，++i 到 1、--j 到 3；继续向中间靠拢。
- 只要指针遇到不该待在自己这一边的数就停下来，两人配对交换，一次就把错位的两头同时纠正！

本页要点

左指针 i 专找不该在左边的（$\geq$ *pivot*），右指针 j 专找不该在右边的（$\leq$ *pivot*）。

手算跟踪表：快排一轮划分 [4，7，2，5，1]

设 $l=0, r=4$，选中间值 $pivot = a[(0+4)/2] = a[2] = 2$，初始 $i=0, j=4$，循环条件 $i \le j$：

轮次	i 停在	j 停在	$i \le j$?	动作	动作后数组与指针
1	$0(4)$	$4(1)$	真 $(0 \le 4)$	交换 $a[0], a[4]$	`[1,7,2,5,4]`, $i=1, j=3$
2	$1(7)$	$2(2)$	真 $(1 \le 2)$	交换 $a[1], a[2]$	`[1,2,7,5,4]`, $i=2, j=1$
3	-	-	假 $(2 > 1)$	退出循环	拆成左 $[0,1]$ 与右 $[2,4]$

- 第 3 轮判断 $i \le j$ 为假退出后，左半 $[0,1]$ 是 [1，2]（全 ≤ 2），右半 $[2,4]$ 是 [7，5，4]（全 ≥ 2）。
- 接下来只要递归排序 $[l,j]$ 和 $[i,r]$ 即可！

本页要点

注意循环退出的那一刻 $i > j$，此时 $[l,j]$ 全 $\le pivot$，$[i,r]$ 全 $\ge pivot$！

正确性与隐忧：划分不变量与最坏退化

不变量阶段	要验证的内容
初始化	开始前左段 $[l, i-1]$ 与右段 $[j+1, r]$ 均为空，自然满足条件
保持	交换后 $a[i] \le pivot$ 且 $a[j] \ge pivot$，相向移动后性质保持不变
终止	退出时 $i > j$，左段 $[l, j]$ 全 $\le pivot$，右段 $[i, r]$ 全 $\ge pivot$

快速排序的两个隐忧

1. 最坏退化：若每次选到最值当基准，树高退化为 n 层，复杂度跌回 $O(n^2)$ 超时！
2. 不稳定：左右长距离交换会打乱相等元素的原始先后顺序。

本页要点

手写快排取中点作基准可防退化；实战首选 `std::sort`！

快速排序核心模板代码（Hoare 划分）

```
void quick_sort(int l, int r, vector<int> &a) {
    if (l >= r) return;                          // 区间 <= 1 个元素无需再排
    int pivot = a[(l + r) / 2];                  // 取中间元素作基准，防有序退化
    int i = l, j = r;
    while (i <= j) {
        while (a[i] < pivot) ++i;                // 左边找 >= pivot 的元素
        while (a[j] > pivot) --j;                // 右边找 <= pivot 的元素
        if (i <= j) swap(a[i++], a[j--]);        // 交换并各自推进一步
    }
    quick_sort(l, j, a);                         // 递归排左半 [l, j]
    quick_sort(i, r, a);                         // 递归排右半 [i, r]
}
```

本页要点

不能把 `a[i] < pivot` 写成 `<=`，否则全等数组会让指针越界或漏分治！

标准库救星：std::sort 的用法与复杂度保障

手写快排边界容易写错，竞赛中单关键字或多关键字排序统一用 <algorithm> 里的 std::sort：

```
vector<int> a = {5, 1, 4, 2, 3};
sort(a.begin(), a.end());                          // 默认升序：1 2 3 4 5
sort(a.begin(), a.end(), greater<int>());          // 传 greater<> 降序：5 4 3 2 1

int b[6] = {0, 30, 10, 50, 20, 40};                // 1-based 数组（下标 1~5）
sort(b + 1, b + 5 + 1);                            // 左闭右开 [b+1, b+6)，排 1~5
```

- 为什么不怕最坏数据：std::sort 底层是内省排序（快排 + 递归太深时切堆排序 + 小区间插排），最坏也稳保 $O(n \log n)$！
- 如果需要稳定排序：标准库还提供 stable_sort（底层基于归并），相等元素绝对不换位。

本页要点

记住区间是“左闭右开”：sort(b + 1, b + n + 1) 才能把第 n 格也排进去！

快速排序与 std::sort 上机验证（3 组 + 追问）

编译运行 code/sort_usage.cpp 或将上述 quick_sort 传入三组典型测试数据：

- 普通乱序：[5，1，4，2，3] → 升序 1 2 3 4 5，降序 5 4 3 2 1。
- 最小边界：n = 1，[42] → 42（直接 l >= r 返回）。
- 全等特殊序列：[3，3，3，3] → 3 3 3 3（验证没有死循环）。

反事实追问

如果手写快排每次固定写 int pivot = a[l];（总拿最左边做基准），当测试数据给出严格升序的 $1, 2, \ldots, 10^5$ 时会发生什么？
答：每次只能切出长度 1 和长度 $n-1$ 两段，递归深度达 10^5 层，操作数退化到 5×10^9 甚至栈溢出，触发 **TLE / RE**！

快速排序与 std::sort 易错对照

错误类型	常见错误写法	正确做法与原因
WA（少排尾项）	1-based 数组写 `sort(b+1, b+n)`	尾指针是开区间，必须写 `sort(b+1, b+n+1)`
RE（无限递归）	递归子区间写成 `quick_sort(l, i-1)` 配合中点基准	严格匹配 Hoare 划分结果：左 `[l, j]`、右 `[i, r]`
TLE / RE	固定取 `a[l]` 当基准遇到有序序列	改取 `a[(l+r)/2]` 或随机基准，或直接用 `std::sort`
WA（打乱次序）	需要同值保持输入先后却直接用 `std::sort`	换用 `stable_sort` 或把原始编号 `id` 写进比较器

本页要点

平时刷题一律优先用 `std::sort`；考手写分治原理时牢记 `[l, j]` 与 `[i, r]` 边界！

生活类比：合并两摞“已经各自排好序”的牌

场景：桌面有左、右两摞各 4 张按从小到大排好的扑克牌

左摞：[2，5，7，9]；右摞：[1，3，6，8]。怎样快速把它们合并成一大摞有序牌？

- 完全不需要把 8 张牌全摊开重新找最小！
- 因为两摞各自最上面的牌就是各自这一摞里最小的，所以每一步只需要做一件事：看左摞顶和右摞顶谁更小，把较小的放进结果堆！
- 新术语——二路归并（**Two-way Merge**）：把两段各自有序的序列合并成一段有序序列；两个指针各自从头往尾走一遍，总长度 n 只需 $O(n)$ 步！

本页要点

合并两段有序序列只要 $O(n)$ 线性时间——这正是归并排序能稳拿 $O(n \log n)$ 的底气！

为什么还学归并排序？快排 vs 归并对比

既然 C++ 已经有了很快的 `std::sort`，为什么还要专门学归并排序？

比较维度	手写快速排序	`std::sort`	归并排序
最坏时间复杂度	$O(n^2)$（可被卡）	$O(n \log n)$	严格 $O(n \log n)$
额外空间复杂度	$O(\log n)$ 栈	$O(\log n)$ 栈	$O(n)$ 临时数组 `buf`
是否稳定排序	不稳定	不稳定	稳定
能否求逆序对	不能直接求	不能	核心工具！

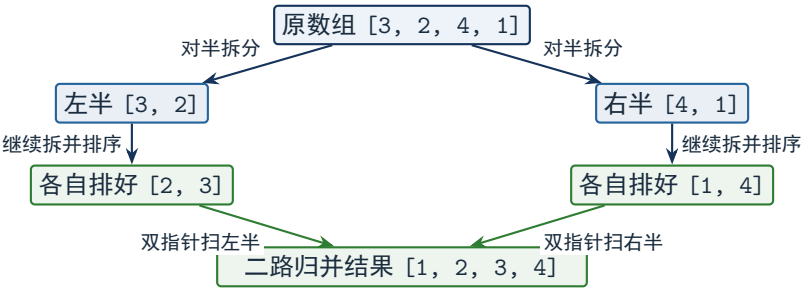
- 归并排序不管输入长什么样，每次都死死对半劈开：树高永远是 $\log_2 n$，绝不会退化。
- 下一节的求逆序对，正是利用二路归并时的“有序性”一口气统计跨越左右两半的逆序对！

本页要点

归并排序用 $O(n)$ 临时空间换来两件宝物：绝对稳定、绝对 $O(n \log n)$，还能顺便数逆序对！

TikZ 图解：分治拆半与自底向上二路归并

对数组 [3，2，4，1] 执行归并排序，先“拆到只剩 1 个”，再“两两向上合并”：



本页要点

采用左闭右开区间 [l，r)：当 r - l <= 1 时只有一个或零个元素，天然有序，直接返回！

手算跟踪表：合并左半 [2，3] 与右半 [1，4]

设 $l=0, mid=2, r=4$，指针 i 扫 $[0,2)$，j 扫 $[2,4)$，k 写缓冲区 `buf`，循环条件 $i<2 \wedge j<4$：

步	$(i<2 \wedge j<4)$?	比较 $a[i] \le a[j]$	写入 `buf[k]`	新 (i,j,k)	说明
1	真 $(0<2, 2<4)$	$2 \le 1$ 假	1（右）	$(0,3,1)$	取右半 $a[2]=1$
2	真 $(0<2, 3<4)$	$2 \le 4$ 真	2（左）	$(1,3,2)$	取左半 $a[0]=2$
3	真 $(1<2, 3<4)$	$3 \le 4$ 真	3（左）	$(2,3,3)$	取左半 $a[1]=3$
4	假 $(i=2)$	不再比较	退出主循环	$(2,3,3)$	收尾搬入右半残余 4

- 退出主循环时，i 和 j 必定只有一个走到了尽头，另一个还剩没搬完的尾部元素。
- 再用两个独立的 `while` 把剩下的元素原封不动拷进 `buf` 末尾，最后拷回 `a[l..r-1]`。

本页要点

最后一次 $i=2$ 使条件为假退出主循环后，必须用两个收尾 `while` 把残余元素接上！

正确性证明：归并排序的循环不变量

不变量阶段	要验证的内容
初始化	主循环开始前 $k = l$，`buf[l..l-1]` 为空；且左右子段 $a[l..mid-1]$、$a[mid..r-1]$ 已分别有序
保持	每次挑出 $a[i]$ 与 $a[j]$ 中较小者放进 `buf[k]`，所以 `buf[l..k]` 始终升序，且包含前 $k-l+1$ 小元素
终止	补齐残余并写回后，`a[l..r-1]` 严格升序；当 $a[i] == a[j]$ 时先取左半 $a[i]$，保证稳定！

为什么要写 `a[i] <= a[j]`？

左半的元素在原数组里的位置一定比右半靠前。遇到相等元素时写 `<=` 就能优先取走左半元素，相等元素的相对前后顺序永远不会被颠倒！

本页要点

靠 `<=` 保住稳定性，靠归并后拷贝回 `a` 让上一层继续拿到有序段。

归并排序模板代码（完整函数）

```
void merge_sort(int l, int r, vector<int> &a, vector<int> &buf) {
    if (r - l <= 1) return;                          // 长度 <= 1 天然有序
    int mid = (l + r) / 2;
    merge_sort(l, mid, a, buf);                      // 先排左半 [l, mid)
    merge_sort(mid, r, a, buf);                      // 再排右半 [mid, r)
    int i = l, j = mid, k = l;
    while (i < mid && j < r)
        buf[k++] = (a[i] <= a[j] ? a[i++] : a[j++]); // 写 <= 保证稳定
    while (i < mid) buf[k++] = a[i++];               // 搬运左半残余
    while (j < r)   buf[k++] = a[j++];               // 搬运右半残余
    for (int t = l; t < r; ++t) a[t] = buf[t];       // 拷回原数组 a
}
```

本页要点

临时数组 buf 在外面只开一次、传引用进来复用，避免在递归内部反复申请内存！

归并排序上机验证（3 组数据 + 反事实追问）

编译运行完整程序 `code/merge_demo.cpp`（调用方式 `merge_sort(0, n, a, buf)`）：

- 普通乱序：输入 `4 / 3 2 4 1` → 预期输出 `1 2 3 4`。
- 最小边界：输入 `1 / 9` → 预期输出 `9`（命中 `r - l <= 1` 返回）。
- 全等与全逆序：输入 `3 / 5 5 5` → `5 5 5`；输入 `4 / 4 3 2 1` → `1 2 3 4`。

反事实追问

如果写完后忘记最后一行 `for (int t = l; t < r; ++t) a[t] = buf[t];` 会出现什么结果？

答：所有排序结果都只写进了临时数组 `buf`，原数组 `a` 根本没变！上层递归读到的仍是乱序的 `a`，最终输出完全没排好序（**WA**）。

归并排序易错对照

错误类型	常见错误写法	正确做法与原因
RE（栈溢出）	终止条件写 `if (r <= l) return;`	`[l, r)` 长度为 1 时 `r - l == 1`，必须写 `r - l <= 1`
TLE / MLE	在 `merge_sort` 函数里面现开 `vector<int> buf(n)`	每层递归都开长 n 数组变成 $O(n^2)$，应在 `main` 开一次传引用
WA（忘写回）	排完 `buf` 忘记写回 `a[l..r-1]`	每次归并最后必须把 `buf[l..r-1]` 拷贝回 `a`
WA（不稳定）	判断写成 `a[i] < a[j]`	相等时优先取左半，必须写 `a[i] <= a[j]`

本页要点

记住“外部开 `buf`、左闭右开判 `r - l <= 1`、结尾拷贝回 `a`”三条铁律！

先手算小例子：什么是“逆序对”？

生活引入与定义：给定序列 $a=[3,2,3,2]$（下标 $1\sim4$）

如果两个位置满足在前面的反而更大，即 $i<j$ 且 $a_i>a_j$，就把这对位置 (i,j) 叫做一个逆序对。

我们来手动把 $[3,2,3,2]$ 的全部 6 对下标 (i,j) 一一查清楚：

- $(1,2)$：值为 $(3,2)$，$3>2 \Longrightarrow$ 算 **1** 对；
- $(1,3)$：值为 $(3,3)$，$3=3 \Longrightarrow$ 相等不算！
- $(1,4)$：值为 $(3,2)$，$3>2 \Longrightarrow$ 算 **1** 对；
- $(2,3)$：值为 $(2,3)$，$2<3 \Longrightarrow$ 不算；　$(2,4)$：$(2,2) \Longrightarrow$ 不算；
- $(3,4)$：值为 $(3,2)$，$3>2 \Longrightarrow$ 算 **1** 对。合计共 3 个逆序对！

本页要点

严格记住两个条件：$i<j$ 且 $a_i>a_j$；两个数相等时绝对不算逆序对！

为什么不能两层循环暴力？两个致命痛点

痛点一：时间复杂度 $O(n^2)$ 必定 TLE

枚举全部 $i < j$ 需要 $\frac{n(n-1)}{2}$ 次比较。当题目给出 $n = 10^5$ 时，总次数达到 5×10^9 次——按 1 秒 10^8 次红线，需要 50 秒才能跑完，必判 **TLE**！

痛点二：答案总数会直接撑爆 `int` 上限

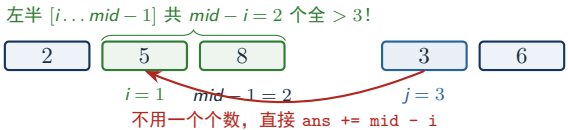
如果序列是完全倒序 $[n, n-1, \ldots, 1]$，每两个位置都构成逆序对，总数是 $\frac{10^5 \times 99999}{2} \approx 5 \times 10^9$。而 `int` 最大只能装约 2.14×10^9，不开 `long long` 必然溢出变负数 **WA**！

本页要点

统计逆序对必须同时解决两件事：算法从 $O(n^2)$ 降到 $O(n \log n)$，计数变量改用 `long long`！

TikZ 图解：为什么合并时一次就能加 mid - i 个？

归并排序合并左半有序段 $L = [2, 5, 8]$（下标 $0, 1, 2$）和右半有序段 $R = [3, 6]$（下标 $3, 4$），此时 $i = 1$, $j = 3$：



- 因为左半 $[l, mid)$ 早就排好序了，一旦发现 $a[i] > a[j]$（$5 > 3$），就说明从 i 到 $mid - 1$ 的所有元素都 $\geq a[i] > a[j]$！
- 所以当右半元素 $a[j]$ 出列放进 `buf` 时，它能与左半尚未出列的全部 $mid - i$ 个元素各凑成一对逆序！

本页要点

只在“右半 $a[j]$ 更小并出列”的那一刻，把左半剩下的 $mid - i$ 个一次性加进答案！

手算跟踪表：归并统计全逆序 [4，3，2，1]

按自底向上顺序跟踪每一次合并过程（右半元素出列时累加 $mid - i$）：

合并区间	左半有序段	右半有序段	新增 mid-i	合并与累加说明
$[0,2)$	$[4]$	$[3]$	$+1$	$4>3$，取右 3 时加 1 对，合为 $[3,4]$
$[2,4)$	$[2]$	$[1]$	$+1$	$2>1$，取右 1 时加 1 对，合为 $[1,2]$
$[0,4)$	$[3,4]$	$[1,2]$	$+2+2=+4$	取 1 加 2 对 $(3,1)(4,1)$；取 2 再加 2 对
合计	递归结束	全长有序	总计 $=6$	恰好等于 $\frac{4\times3}{2}=6$ 对！

本页要点

虽然合并时每段内部变有序了，但“右半全体原本都在左半全体右边”，跨区相对位置一点没丢！

正确性证明：为什么不重复也不遗漏？

任意一对合法位置 $p < q$ 在拆成左半 $[l, mid)$ 和右半 $[mid, r)$ 后，只有三种互不相交的归属：

不变量阶段	要验证的内容
全在左半（$p < q < mid$）	由左半递归 `merge_count(l, mid)` 负责统计完毕，不跨区
全在右半（$mid \le p < q$）	由右半递归 `merge_count(mid, r)` 负责统计完毕，不跨区
跨越左右（$p < mid \le q$）	左半全体位置 $<$ 右半全体位置；在合并 $[l, r)$ 时，每个右半元素 $a[q]$ 恰好在它出列那一瞬间与左半所有 $> a[q]$ 的元素配对一次，绝不重漏！

终止结论

当递归返回到顶层 $[0, n)$ 时，三种情况全部加总完毕，结果严格等于整个原序列的逆序对总数。

归并求逆序对模板代码（完整递归函数）

```
using ll = long long;
ll merge_count(int l, int r, vector<ll> &a, vector<ll> &buf) {
    if (r - l <= 1) return 0;                    // 长度 <= 1 无逆序对
    int mid = (l + r) / 2, i = l, j = mid, k = l;
    ll ans = merge_count(l, mid, a, buf) + merge_count(mid, r, a, buf);
    while (i < mid && j < r) {
        if (a[i] <= a[j]) buf[k++] = a[i++];     // 相等不算逆序对
        else { ans += mid - i; buf[k++] = a[j++]; }
    }
    while (i < mid) buf[k++] = a[i++];
    while (j < r)   buf[k++] = a[j++];
    for (int t = l; t < r; ++t) a[t] = buf[t];
    return ans;
}
```

注：只在 else 分支加 mid - i，返回值与 ans 一律用 long long！

归并求逆序对上机验证（3 组数据 + 反事实追问）

使用完整程序 `code/inversion_count.cpp`（读入 n 与序列）在本地验证三组关键边界：

- 严格升序：输入 `4 / 1 2 3 4` → 预期输出 `0`。
- 全相等序列：输入 `3 / 5 5 5` → 预期输出 `0`。
- 完全逆序：输入 `4 / 4 3 2 1` → 预期输出 `6`。

反事实追问

如果把核心判断 `if (a[i] <= a[j])` 手滑改成了 `if (a[i] < a[j])`，输入全相等序列 `2 / 5 5` 会输出什么？
答：会走进 `else` 分支，把 $5 = 5$ 错误算作逆序对，输出 `1` 而非 `0`（**WA**）！

逆序对易错对照

错误类型	常见错误写法	正确做法与原因
WA（整数溢出）	`int ans = ...;`	$n = 10^5$ 逆序对最高 5×10^9，必须全链路用 `long long`
WA（相等误算）	`if (a[i] < a[j])`	相等不是逆序对，必须写 `if (a[i] <= a[j])`
WA（漏算子段）	只记合并时累加，忘了加左右子递归返回值	`ans = merge_count(左) + merge_count(右) + 跨区`
TLE（超时）	对 $n = 10^5$ 用双层循环或冒泡数交换次数	改用 $O(n \log n)$ 归并排序计数

本页要点

遇到“最小相邻交换次数”或“多少对前面大于后面”，脑海里立刻锁定归并求逆序对！

生活引入：怎么给一张多科成绩单排队？

场景：按多条规则给 n 名同学排领奖顺序

规则依次为： 总分高的排前面； 总分相同看语文分高的在前； 都相同看学号小的在前。

- 为什么不能只给“总分数组”排序？如果你只把总分排了序，总分换了位置，却不知道它原来对应哪位同学、语文多少分！
- 解决方法：用一个结构体 `Student` 把每个人的“学号、语文、总分”捆绑成一个档案袋，排序时整袋一起搬动，再写一个比较函数告诉 `sort` “谁排在谁前面”。

本页要点

多字段关联数据必须先放进 `struct` 捆绑在一起，再配一个自定义比较器排序。

新术语解释：结构体封装与比较器 cmp

- 结构体（`struct`）：把同一个物体的多个属性（比如 `id`、`chi`、`total`）打包成一种新数据类型。
- 比较器（`cmp(x, y)`）：一个返回 `bool` 的函数，用来回答裁判的一个问题——“x 应不应该严格排在 y 前面？”
  - 返回 `true`：表示 x 必须排在 y 前面；
  - 返回 `false`：表示 x 不需要排在 y 前面（y 在前，或者两人等价）。

常见误解纠正

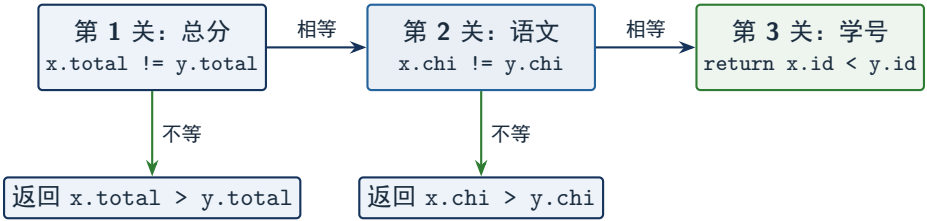
不要把 `cmp(x, y)` 理解成“x 赢不赢或平不平”；它只回答一个是非题——“x 是否严格优先于 y”。

本页要点

`cmp(x, y)` 返回 `true` 的唯一含义就是：x 必须站在 y 的前面。

TikZ 图解：三关键字比较的逐关决策流程

判断学生 x 能否严格排在学生 y 前面（cmp(x, y)）：



本页要点

写多关键字比较器遵循“不等立刻返回，相等再比下一关”；最后一关直接 return。

手算跟踪表：两两比较与“自己比自己”

把 4 名同学按“(总分降, 语文降, 学号升)”两两比较：A(250,90,1)、B(250,95,2)、C(250,90,3)、D(230,99,4)：

比较对 (x, y)	进入哪一关	cmp(x,y) 结果	原因说明
(A, D)	第 1 关（总分）	true	第一关 $250 > 230$ 直接分胜负
(B, A)	第 2 关（语文）	true	总分同 250，语文 $95 > 90$，B 在前
(A, C)	第 3 关（学号）	true	总分、语文全同，学号 $1 < 3$，A 在前
(A, A)	第 3 关（自己比自己）	必须 false	自己不能严格排在自己前面！

本页要点

最后一行 (A, A) 是检查比较器写对没有的试金石：任何时候 cmp(x, x) 都必须为 false！

正确性核心：为什么必须满足“严格弱序”？

新术语解释——严格弱序：比较函数必须遵守的三条数学纪律：

纪律名称	大白话含义与检验方法
反自反性	自己跟自己比永远返回 `false`（`cmp(x, x) == false`），绝不写 `<=` 或 `>=`
非对称性	若 `cmp(x, y)` 为真，则 `cmp(y, x)` 必须为假，不能互相抢前面
传递性	x 在 y 前、y 在 z 前，则 x 必须排在 z 前，否则排序会兜圈子

违反严格弱序的后果

若把 `<` 写成 `<=`，`std::sort` 的哨兵指针遇到等值元素时会一直向后越界扫描，最终触发段错误 **RE**！

结构体多关键字排序标准模板

```
struct Student { int id, chi, total; };
bool cmp(const Student &x, const Student &y) {
    if (x.total != y.total) return x.total > y.total; // 1. 总分降序
    if (x.chi != y.chi)     return x.chi > y.chi;     // 2. 语文降序
    return x.id < y.id;                               // 3. 学号升序
}
// 主函数调用：sort(a.begin(), a.end(), cmp);
```

- 传参写 `const Student &x`：既省去拷贝结构体的耗时，又防止函数误改原数据。

本页要点

前几关用 `!=` 分流，最后一关写严格不等号 `<`，绝不写 `<=`！

结构体排序上机验证（3 组数据 + 反事实追问）

用上述 Student 与 cmp（或 code/scholarship.cpp）验证三组常见考场边界：

- 总分不同看总分：A(1，50，200) vs B(2，99，190) → A 在前（第 1 关已定）。
- 总分相同看语文：A(1，80，250) vs B(2，90，250) → B 在前（第 2 关分出）。
- 全相同看学号：5 人三科全同 → 依次输出学号 1 2 3 4 5。

反事实追问

如果写比较器时漏写了 `if (x.total != y.total)`，直接写成 `if (x.total > y.total) return true; return x.id < y.id;` 会发生什么？
答：当 `x.total < y.total` 时程序掉进第 3 关比学号，如果 `x.id < y.id` 也会返回 `true`，不仅排名全乱（**WA**），还打破非对称性引发 **RE**！

结构体多关键字排序易错对照

错误类型	常见错误写法	正确做法与原因
RE（指针越界）	最后一关写成 `x.id <= y.id`	自己比自己返回 `true` 破坏严格弱序，必写 `<`
WA（漏掉反向）	只写 `if (x.a > y.a) return true;`	必须写 `if (x.a != y.a) return x.a > y.a;` 处理小于情况
WA（丢失编号）	排序前没存 `id`，排完后找回原下标	读入时就把 `id = i + 1` 写进结构体
TLE（拷贝慢）	写 `bool cmp(Student x, Student y)` 且结构体很大	改成 `const Student &x` 按常量引用传参

本页要点

写完 `cmp` 后默念一遍“不等立刻 return，最后一关不带等号，参数带 `const &`”！

[入门] 例 1 | 明明的随机数（一本通 1184）：先读懂要求

大白话题意

明明生成了 N 个 $1 \sim 1000$ 之间的随机整数（$1 \le N \le 100$）。请去掉重复出现的数（只留一个），再从小到大排序。输出第 1 行为不重复的个数 M，第 2 行为这 M 个数。

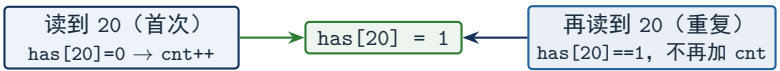
1. 输入：第 1 行正整数 N，第 2 行 N 个 $1 \sim 1000$ 的正整数。
2. 处理：去掉重复数字，并按从小到大顺序排好。
3. 输出：第 1 行输出去重后的数量 M；第 2 行按升序输出这 M 个数。

原题样例

输入：10 和 20 40 32 67 40 20 89 300 400 15
输出：8 和 15 20 32 40 67 89 300 400。

[入门] 例 1 | 怎么想到的与 TikZ 建模图

- 观察数据范围：每个数都在 $1 \sim 1000$ 之间，值域非常小，正是计数桶排序的主场！
- 桶去重策略：开一个标记数组 `has[1..1000]` 初值为 0。每读入一个数 x，只有当 `has[x] == 0` 时才把新数计数 `cnt++`，然后把 `has[x]` 设为 1。最后从小到大扫描 $1 \sim 1000$ 输出所有为 1 的下标。



本页要点

用桶标记数组既顺便完成了“去重”，又顺便完成了“升序排列”，一举两得！

[入门] 例 1 | 样例手推与核心步骤表

输入 20 40 32 67 40 20 89 300 400 15，跟踪不同数字个数 cnt 的变化：

按顺序计算步骤	纸上得到结果	核对问题
1. 读前 4 个 20,40,32,67	对应桶置 1，cnt = 4	首次遇到才让 cnt++
2. 再读 40，20	has[40]，has[20] 已是 1	重复出现，cnt 仍为 4
3. 读完 89,300,400,15	再置 4 个桶，cnt = 8	先输出第一行 8
4. 从 $v=1$ 扫到 1000	遇到 1 就打印 v	输出 15 20 32 40 67 89 300 400

本页要点

相对基础桶模板变化：加了一个变量 cnt 统计首次进桶的个数，先打印 cnt 再打印数值！

[入门] 例 1 | 完整可编译代码（code/random_dedup.cpp）

```cpp
#include <bits/stdc++.h>
using namespace std;
int main() {
    int n, x, cnt = 0;
    if (!(cin >> n)) return 0;
    vector<int> has(1001, 0);                    // 桶覆盖 1~1000
    for (int i = 0; i < n; ++i) {
        cin >> x;
        if (has[x] == 0) ++cnt;                  // 首次出现计数
        has[x] = 1;
    }
    cout << cnt << '\n';
    for (int v = 1; v <= 1000; ++v) if (has[v]) cout << v << ' ';
}
```

注：桶大小必须开 1001，确保能访问到 has[1000]！

[入门] 例 1 | 易错对照与三组自测边界

常见错误对照

- **WA**：先输出了排好的数组，忘记先在第一行输出总数 M；
- **RE**：桶只开了 1000 格，遇到 `x = 1000` 时下标越界；
- **WA**：扫描桶写成 `v < 1000`，漏掉了最大值 1000。

三组必测自检数据

1. 最小 N：`1 / 1` → 1 和 1；
2. 全相同：`5 / 7 7 7 7 7` → 1 和 7；
3. 值域上界：`2 / 1000 1` → 2 和 1 1000。

本页要点

拓展思考：如果值域变成 $1 \sim 10^9$，就改用 `sort` 加 `unique` 去重！

[基础] 例 2 | 奖学金（一本通 1179）：先读懂要求

大白话题意

有 n 位同学（$5 \le n \le 299$），学号按输入顺序编为 $1 \sim n$。每人有语、数、英三科成绩（$0 \sim 100$）。规则： 总分高在前； 总分相同语文高在前； 再相同学号小在前。输出前 5 名学号与总分。

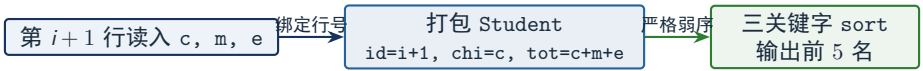
1. 输入：第 1 行 n，接下来 n 行每行 3 个整数（语、数、英）。
2. 处理：算出每人的总分，捆绑学号与语文成绩按三关键字排序。
3. 输出：5 行，每行两个整数“学号总分”。

原题样例

输入 6 人：(90,67,80), (87,66,91), (78,89,91), (88,99,77), (67,89,64), (78,89,98)
⇒ 输出前 5 名：6 265、4 264、3 258、2 244、1 237。

[基础] 例 2 | 怎么想到的与 TikZ 建模图

- 题目虽然输入了数学和英语分，但比较规则里根本用不到单科数学、英语，只用到总分！
- 因此在读入第 i 行时，直接算出总分 `c + m + e`，只把 `{id, chinese, total}` 装进结构体即可。



本页要点

先筛选真正参与比较的字段存进结构体，不用把无关信息全搬进内存。

[基础] 例 2 | 样例手推与核心步骤表

把样例 6 位同学转成（学号，语文，总分）后排序结果如下：

原学号	语文分	总分	排序名次	胜出关卡核对
6	78	265	第 1 名	第 1 关总分 265 全场最高
4	88	264	第 2 名	第 1 关总分 264 次高
3	78	258	第 3 名	第 1 关总分 258
2	87	244	第 4 名	第 1 关总分 244
1	90	237	第 5 名	第 1 关 237 > 号 5 的 220

本页要点

相对基础比较器模板变化：读入时一边累加三科总分一边记下 `i + 1` 作为学号，排完只取前 5 个输出。

[基础] 例 2 | 完整可编译代码 (1/3)（code/scholarship.cpp）

```cpp
#include <bits/stdc++.h>
using namespace std;

// 一本通 1179 奖学金：三关键字严格弱序排序，输出前 5 名学号与总分
struct Student {
    int id, chinese, total;
};

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

本页要点

先定义仅包含 id，chinese，total 三个整型成员的档案结构体。

[基础] 例 2 | 完整可编译代码 (2/3)（code/scholarship.cpp）

```
int n;
if (!(cin >> n)) return 0;
vector<Student> a(n);
for (int i = 0; i < n; ++i) {
    int c, m, e;
    cin >> c >> m >> e;
    a[i] = {i + 1, c, c + m + e};                    // 学号按输入顺序从 1 编号
}
sort(a.begin(), a.end(), [](const Student &x, const Student &y) {
    if (x.total != y.total) return x.total > y.total;          // 1. 总分高在前
    if (x.chinese != y.chinese) return x.chinese > y.chinese; // 2. 语文高在前
    return x.id < y.id;                                        // 3. 学号小在前
});
```

本页要点

Lambda 比较器里的第 3 关必须严格写成 `x.id < y.id`，满足严格弱序。

[基础] 例 2 | 完整可编译代码 (3/3)（code/scholarship.cpp）

```
    for (int i = 0; i < min(5, n); ++i) {
        cout << a[i].id << ' ' << a[i].total << '\n';
    }
    return 0;
}
```

- 输出使用 `min(5, n)`，即使遇到 $n < 5$ 的异常小测试也不会越界。
- 确认输出的是 `a[i].id` 和 `a[i].total`（不要误打成语文分）。

本页要点

完整程序按“读入打包 $\rightarrow$ 严格弱序排序 $\rightarrow$ 输出前 5 名”三步走，清晰不易错。

[基础] 例 2 | 易错对照与三组自测边界

常见错误对照

- **WA**：第 3 关学号写反成 `x.id > y.id`；或忘了存 `id`；
- **RE**：最后一关写成 `x.id <= y.id` 破坏严格弱序；
- **WA**：输出把总分错打印成语文分。

三组必测自检数据

1. 最小规模：`n = 5`，恰好全部输出；
2. 全同分：5 人三科相同 $\rightarrow$ 依次输出 `1..5`；
3. 同总分看语文：(60,70,70) 排在 (50,75,75) 前面。

本页要点

遇到多规则排名题，用几组专门测试第 2、第 3 关的平局数据比只测样例更有效！

[基础] 例 3 | 车厢重组（一本通 1310）：先读懂要求

大白话题意

有一列 N 节车厢（$1 \le N \le 10000$），初始编号是 $1 \sim N$ 的一个排列。每次只能把相邻两节车厢对调位置。问最少要对调多少次，才能按 $1 \sim N$ 从小到大排好？

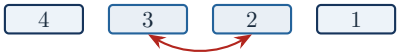
1. 输入：第 1 行整数 N，第 2 行 N 个互不相同的整数。
2. 处理：求把这列数通过相邻交换变成严格升序的最少操作次数。
3. 输出：一行一个整数，表示最少旋转（交换）次数。

原题样例

输入：4 和 4 3 2 1 ⇒ 输出：6。

[基础] 例 3 | 怎么想到的与 TikZ 建模图

- 每次只交换相邻两节车厢，就像冒泡排序里的相邻交换！如果左大右小就交换，会发生什么？



交换 $(3,2)$ 只消掉 1 对逆序，其余相对位置不变！

- 只要存在还没排好的车厢，就一定存在一对相邻逆序（左 $>$ 右）；每交换一次相邻逆序，总逆序对数量精确减 1。
- 排好序时逆序对数为 0，所以最少相邻交换次数 $\equiv$ 初始逆序对总数！
- $N \le 10000$ 时，冒泡最坏约 5×10^7 次，用归并统计仅 1.3×10^5 次，零超时风险！

本页要点

相邻两两交换的最少步数，永远严格等于序列的逆序对总数！

[基础] 例 3 | 样例手推与核心步骤表

用归并排序统计样例 [4，3，2，1]：

按顺序计算步骤	纸上得到结果	核对问题
1. 合并 [4] 与 [3]	取右 3，累加 $mid - i = 1$	有序段变为 [3，4]
2. 合并 [2] 与 [1]	取右 1，累加 $mid - i = 1$	有序段变为 [1，2]
3. 合并 [3,4] 与 [1,2]	取 1 加 2 对，取 2 加 2 对	跨区 $+4$，总计 $1 + 1 + 4 = 6$
4. 返回顶层 ans	最终输出 6	答案可能超 5×10^7，用 long long

本页要点

相对基础归并求逆序对模板：本题直接读入 N 个数，无需任何额外改动，套模板即 AC！

[基础] 例 3 | 完整可编译代码 (1/3)（code/carriage_reorder.cpp）

```cpp
#include <bits/stdc++.h>
using namespace std;

// 一本通 1310 车厢重组：相邻交换次数等于序列逆序对总数，归并 O(n log n) 统计
int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int n;
    if (!(cin >> n)) return 0;
    vector<int> a(n), buf(n);
    for (int &x : a) cin >> x;
```

本页要点

主函数先把序列和复用缓冲区 buf 一次性开好，准备进入递归计数。

[基础] 例 3 | 完整可编译代码 (2/3)（code/carriage_reorder.cpp）

```
auto merge_count = [&](auto &&self, int l, int r) -> long long {
    if (r - l <= 1) return 0;                    // 区间长度为 0 或 1 时无逆序对
    int mid = (l + r) / 2;
    long long ans = self(self, l, mid) + self(self, mid, r);
    int i = l, j = mid, k = l;
    while (i < mid && j < r) {
        if (a[i] <= a[j]) {
            buf[k++] = a[i++];                   // 左边较小或相等，不构成逆序对
        } else {
            ans += mid - i;                      // a[i..mid-1] 均大于 a[j]
            buf[k++] = a[j++];
        }
    }
```

本页要点

遇到 a[i] > a[j] 时立刻累加 mid - i，把右半出列元素贡献一次算完。

[基础] 例 3 | 完整可编译代码 (3/3)（code/carriage_reorder.cpp）

```
        while (i < mid) buf[k++] = a[i++];
        while (j < r) buf[k++] = a[j++];
        for (int t = l; t < r; ++t) a[t] = buf[t];  // 拷贝回原数组保证有序
        return ans;
    };

    cout << merge_count(merge_count, 0, n) << '\n';
    return 0;
}
```

- C++14 起可用泛型 Lambda `auto &&self` 实现原地递归函数，无需把数组写成全局变量。

本页要点

收尾两个 `while` 补完残余元素并拷贝回 `a`，最后直接输出 `merge_count(merge_count, 0, n)`。

[基础] 例 3 | 易错对照与三组自测边界

常见错误对照

- **TLE** 风险：$N = 10^4$ 逆序时手写双层循环交换约 5×10^7 次，某些卡常环境危险；
- **WA**：归并时忘记把 `buf` 写回 `a`，导致上层计算失真；
- **WA**：手滑只输出了排序后的数组而非总交换次数。

三组必测自检数据

1. 最小 N：`1 / 1` $\rightarrow$ 输出 `0`；
2. 已有序：`4 / 1 2 3 4` $\rightarrow$ 输出 `0`；
3. 全逆序：`4 / 4 3 2 1` $\rightarrow$ 输出 `6`。

本页要点

看懂“相邻交换 $\leftrightarrow$ 逆序对”这层等价转化，后面遇到类似冒泡步数题一秒钟就能建模！

[提高] 例 4 | 求逆序对（一本通 1311）：先读懂要求

大白话题意

给定长度为 n 的序列 $a_1, a_2, \ldots, a_n$（$1 \le n \le 10^5$，$1 \le a_i \le 10^5$，序列中允许出现相等的数字）。统计满足 $i < j$ 且 $a_i > a_j$ 的逆序对总数。

1. 输入：第 1 行 n，接下来 n 行每行一个整数 a_i。
2. 处理：在 $O(n \log n)$ 内准确统计所有严格大于的逆序对。
3. 输出：一行一个整数（最高约 5×10^9，超出 `int` 范围）。

原题样例

输入：4，接下来四行 3 / 2 / 3 / 2 $\Rightarrow$ 输出：3。

[提高] 例 4 | 怎么想到的与 TikZ 建模图

相比《车厢重组》，本题有两处硬核升级：

- 升级一（防溢出）：$n \le 10^5$ 时逆序对高达 5×10^9，答案变量及数组建议开 `long long`。
- 升级二（防相等误计）：存在相同数字（如两个 2）时，合并到相等元素必须走 `<=` 分支先取左边！



本页要点

当左右两段都出现 2 时，必须先让左半的 2 进缓冲区，右边才不会被左边后面比 2 大的 3 算漏或误算！

[提高] 例 4 | 样例手推与核心步骤表

手推样例 [3，2，3，2] 的合并全过程：

合并子段	执行分支与累加	易错核对
1. [3] 与 [2]（左半）	$3 > 2$，取右 2，ans += 1	变为 [2，3]
2. [3] 与 [2]（右半）	$3 > 2$，取右 2，ans += 1	变为 [2，3]
3. [2,3] 与 [2,3]	先判 $2 \le 2$ 取左 2；再 $3 > 2$ 取右 2 +1；最后 $3 \le 3$ 取左 3	相等 2 与 3 都走 <= 分支，最终得 3！

本页要点

如果第 3 步把 <= 错写成 <，遇到 $2 == 2$ 时会误取右边并加上左边的 2 和 3（多加 2 对）！

[提高] 例 4 | 完整可编译代码 (1/3)（code/inversion_count.cpp）

```cpp
#include <bits/stdc++.h>
using namespace std;

// 一本通 1311 求逆序对：n <= 1e5，逆序对总数可达 5e9，结果必须使用 long long
int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int n;
    if (!(cin >> n)) return 0;
    vector<long long> a(n), buf(n);
    for (auto &x : a) cin >> x;
```

本页要点

输入规模达 10^5 时务必加上 `ios::sync_with_stdio(false); cin.tie(nullptr);` 加速读入。

[提高] 例 4 | 完整可编译代码 (2/3)（code/inversion_count.cpp）

```
auto merge_count = [&](auto &&self, int l, int r) -> long long {
    if (r - l <= 1) return 0;
    int mid = (l + r) / 2;
    long long ans = self(self, l, mid) + self(self, mid, r);
    int i = l, j = mid, k = l;
    while (i < mid && j < r) {
        if (a[i] <= a[j]) buf[k++] = a[i++];    // 严格注意写 <=，相等不算逆序对
        else {
            ans += mid - i;
            buf[k++] = a[j++];
        }
    }
```

本页要点

第 7 行的 <= 既是稳定性的开关，更是过滤“相等元素不算逆序对”的关键闸门。

[提高] 例 4 | 完整可编译代码 (3/3)（code/inversion_count.cpp）

```
        while (i < mid) buf[k++] = a[i++];
        while (j < r) buf[k++] = a[j++];
        for (int t = l; t < r; ++t) a[t] = buf[t];
        return ans;
    };

    cout << merge_count(merge_count, 0, n) << '\n';
    return 0;
}
```

- 注意每行输入一个数时，`for (auto &x : a) cin >> x;` 同样能正确读完换行分隔的 n 个数。

本页要点

和《车厢重组》一脉相承：同一套归并计数模板，在 $n = 10^5$ 含重复值下依然稳过！

[提高] 例 4 | 易错对照与三组自测边界

常见错误对照

- **WA**：答案用 `int` 存储，遇到 $n = 10^5$ 大逆序溢出；
- **WA**：合并判断写 `a[i] < a[j]`，导致相等元素被当成逆序对；
- **TLE**：双重循环暴力跑 10^{10} 步超时。

三组必测自检数据

1. 全相等：`4 / 5 5 5 5` → 输出 `0`；
2. 严格升序：`4 / 1 2 3 4` → 输出 `0`；
3. 大规模上限验证：造 $n = 10^5$ 全逆序，检查答案是否等于 `4999950000`。

本页要点

造一组全相等和一组 $n = 10^5$ 逆序压力测试，能一次性排查 `<=` 和 `long long` 两大雷区！

排序与逆序对常见错误对照与定位指南

题目 / 场景	常见错误写法	正确做法与原因
多关键字比较器	最后一关写 `x.id <= y.id`	严格写 `</>`，自比必为 `false`
$O(n^2)$ 跑大数组	对 $n = 10^5$ 写冒泡或选择排序	必 **TLE**，换 `std::sort` 或归并
桶排去重越界	值域 $1 \sim 1000$ 只开 `cnt(1000)`	至少开 `V + 1`（1001）防 **RE**
带重复值逆序对	归并判断写成 `a[i] < a[j]`	相等不算逆序对，必须写 `a[i] <= a[j]`
大规模逆序对	用 `int ans` 接收返回结果	最高 5×10^9 必溢出 **WA**，开 `long long`

本页要点

定位口诀：**RE** 查 <= 与桶上界；**TLE** 查 $O(n^2)$；**WA** 查 `long long` 与 <= 分支！

提交前检查清单（六必查）

1. 数据规模 $n \geq 10^4$ 时，排序是否用了 $O(n \log n)$ 的 `std::sort` 或归并，而不是 $O(n^2)$ 冒泡/选择？
2. 自定义比较器 `cmp(x, x)` 自己跟自己比时，是否严格返回 `false`？
3. 如果输出要用到原始输入编号（如学号、报名号），是否在排序前把 `id` 存进了结构体？
4. 桶排序数组大小是否覆盖了最大可能值（值域含 V 时至少开 `V + 1`）？
5. 归并统计逆序对时，返回值与累加变量 `ans` 是否全用 `long long`？
6. 归并合并分支是否为 `if (a[i] <= a[j])`（相等时优先取左半且不加逆序对）？

最后一步

造一组全相等序列 `[5,5,5,5]` 和一组全逆序序列 `[5,4,3,2,1]` 上机跑通再提交！

本章全部练习与完成标准

来源	题名	难度	核心考点与对应源码
1184	明明的随机数	[入门]	值域 $1 \sim 1000$ 计数桶去重升序（random_dedup.cpp）
1185	单词排序	[入门]	字符串 sort + unique 按字典序去重输出
1176	谁考了第 k 名	[基础]	结构体绑定学号与成绩，按成绩降序取第 k 位
1179	奖学金	[基础]	三关键字严格弱序排名（scholarship.cpp）
1180	分数线划定	[基础]	$\lfloor 1.5m \rfloor$ 划定分数线 + 双关键字排序筛选
1310	车厢重组	[基础]	相邻交换最少步数 = 逆序对（carriage_reorder.cpp）
1311	求逆序对	[提高]	$n \le 10^5$ 含重复值与 long long（inversion_count.cpp）

本页要点

完成标准：能不看模板手写严格弱序比较器与归并求逆序对，并独立 AC 全部 7 题。

课后练习路线（从入门到提高）

1. 第一阶段（巩固桶排与去重）：先 AC《明明的随机数》（1184）与《单词排序》（1185），比较“开桶”和“sort + unique”两种去重写法。
2. 第二阶段（练熟结构体排序）：独立写出《谁考了第 k 名》（1176）、《奖学金》（1179）和《分数线划定》（1180），重点构造同分平局自测数据。
3. 第三阶段（攻克归并逆序对）：独立默写《车厢重组》（1310）和《求逆序对》（1311），构造 $n = 10^5$ 全逆序数据检查是否爆 int。

复习动作

合上课件，把归并合并 [2，5，8] 与 [3，6] 的过程画在草稿纸上，口述为什么 $5 > 3$ 时一次就能加 mid - i。

课后安排与下一章预告

- 源码对照：把 `code/` 下全部源码自己重打并编译运行一遍，重点测全相等和全逆序极端用例。
- 下一章预告：第 3 章《高精度运算》——这一章我们见识了 $n = 10^5$ 时逆序对总数达到 5×10^9，靠 `long long` 保住了不溢出；但如果题目直接给你两个 200 位的数字，连 19 位上限的 `long long` 也装不下了怎么办？下一章我们用倒序数组手写加减乘除竖式，彻底打破整数位数天花板！

本章一句话

写好严格弱序让 `sort` 排万物，值域小用桶 $O(n + V)$，借归并有序性在 $O(n \log n)$ 数清每一对逆序。

谢谢！