C++ 编程语言与 STL · 第八章（承接第七章结构体，衔接算法基础篇）

STL 实战与高效查找

从 $O(n)$ 线性扫描到 $O(\log n)$ 精准定位

HGNU-ACM 实验室算法组

配套：6 类高频容器与查找工具 + 5 道完整例题

（code/stl_ex0..4.cpp）

本章目录

为什么学 STL：容器地图与 vector

pair、严格弱序与离散化

数组二分神器：lower_bound 与 upper_bound

动态查找前驱后继：set 与 multiset

栈、队列与优先队列 priority_queue

键值映射与位加速：map、unordered_map 与 bitset

经典例题五步精讲（覆盖全部 code/*.cpp）

常见错误、检查清单与练习安排

本页要点

学完第七章“结构体与 sort”后，本章把容器、二分查找、有序集合与堆装进你的竞赛工具箱。

先想一个生活问题：在排队名单里随时找邻居

生活场景：按身高排队并随时插新人

班上已有 10^5 名同学按身高从小到大排好队，接下来有 10^5 次现场提问：

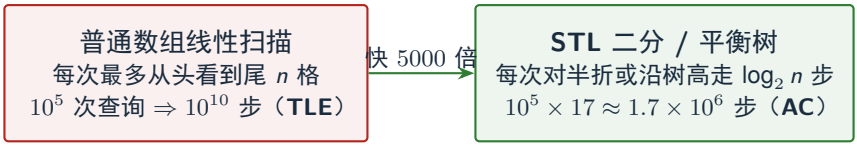
- 有时问：身高 ≥ 175 的第一位同学站在第几个位置？
- 有时新来一名身高 172 的同学，先问队里比他矮的最高是谁、比他高的最矮是谁，再把他插进队伍。

- 新术语——**STL**（标准模板库）：C++ 自带的现成工具箱，里面装好了会自己变长的数组、自动排序的集合、堆和字典。
- 第七章我们学过用结构体数组 + `sort` 给静态名单排序；本章解决“边变边查”和“极快查找”的问题。

本页要点

STL 不是偷懒工具，而是把经过严格测试的高效数据结构直接交给你使用的竞赛武器。

暴力做法 $O(n)$ 为什么会 TLE?



- 评测机 1 秒大约能执行 $10^7 \sim 10^8$ 次简单操作；超过 10^8 次通常就会超时（**TLE**）。
- 每次插入新同学时，普通数组要把后面的人全部往右挪一格，同样会跑到 10^{10} 步。

本页要点

看到数据规模 $n, q \geq 10^5$ 时，绝不能每次线性从头扫描，必须换用 $O(\log n)$ 工具。

STL 容器选型一张图

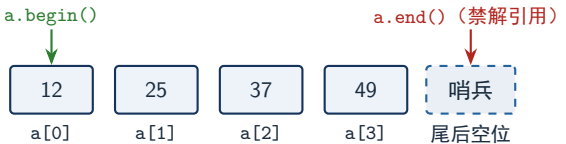
需求场景	首选 STL 容器	单次操作耗时	典型题目用途
长度不固定 / 邻接表	`vector<T>`	尾插均摊 $O(1)$	动态数组、图论存边
一端 / 两端进出	`stack/queue/deque`	进出均摊 $O(1)$	括号匹配、BFS、滑窗
随时取最大或最小值	`priority_queue<T>`	压入/弹出 $O(\log n)$	合并果子、动态中位数
动态有序找前驱后继	`set / multiset`	增删查 $O(\log n)$	营业额统计、近邻查询
字符串或大下标计数	`map / unordered_map`	$O(\log n)$ / 均摊 $O(1)$	词频统计、离散映射
超长 $0/1$ 开关批量算	`bitset<N>`	整行移位 $O(N/64)$	0-1 背包可行性

本页要点

读题先分清“静态二分”“动态查邻居”“只取最值”还是“按键计数”，选对容器题就解了一半。

从定长数组到 vector：三大新术语大白话

- 定长痛点：全局数组长度写死，多组数据一旦忘清空就会串数据（**WA**）。
- 动态数组 `vector<T>`：长度随 `push_back` 自动变长，支持 `assign` 一行清空。
- 迭代器（**Iterator**）：指向容器格子的“智能光标”，`*` 取值，`++` 移下一格。
- 左闭右开 `[begin, end)`：`begin()` 指第 0 格，`end()` 指末位后的空位。



本页要点

任何 STL 算法都认左闭右开 `[begin, end)`；元素个数等于 `end() - begin()`。

手算跟踪：vector 增删变化与 size() 下溢边界

观察初始为空的 vector<int> a 连续执行操作后的状态（每步都要核对 size()）：

步骤	执行操作	操作后 a	a.size() 与边界注意
1	初始定义 vector<int> a;	[]	size() = 0（此时 a.size()-1 下溢！）
2	a.push_back(10);	[10]	size() = 1，末尾加入 10
3	a.push_back(25);	[10, 25]	size() = 2，a.back() = 25
4	a.pop_back();	[10]	size() = 1，弹出尾元素 25
5	a.assign(3, 0);	[0, 0, 0]	size() = 3，原内容清空并重设 3 个 0

- 当 a 为空时，a.size() 的类型是无符号整数 size_t，计算 a.size() - 1 会回绕成约 1.8×10^{19}！

本页要点

循环写 i < (int)a.size()，绝对不要在无强转的情况下写 i <= a.size() - 1。

模板代码：vector 开数组、二维邻接表与安全遍历

```
int n = 4;
vector<int> a(n, 0);                  // 开长度为 n 的数组，全部初值为 0
a.push_back(42);                      // 尾插元素 42，长度自动变为 n+1
vector<vector<int>> g(n + 1);         // 二维动态数组：图论存每个点的邻居
g[1].push_back(3);                    // 给 1 号点添加一条连向 3 号点的边
g.assign(n + 1, {});                  // 多组数据时一行清空并重设大小
for (int i = 0; i < (int)a.size(); ++i) {
    cout << a[i] << ' ';              // 先强转 (int) 再比较，防空数组下溢
}
```

为什么这样写？

用 `g.assign(n + 1, {})` 可以一次性把旧数据的长度和内容全部重置；遍历时加 `(int)` 或改用范围循环 `for (int x : a)`，彻底消灭无符号下溢。

上机验证与易错对照：vector 专项

上机自测三组数据

1. 普通：$n = 3$ 依次尾插 4,7,9 $\rightarrow$ 4 7 9；
2. 最小：$n = 0$ 安全循环 $\rightarrow$ 0 次不崩；
3. 重置：插 3 项后 assign(2,5) $\rightarrow$ 5 5。

追问：空数组写 `i <= a.size()-1` 会怎样？
答：无符号下溢成 1.8×10^{19}，越界 **RE**！

常见错误对照

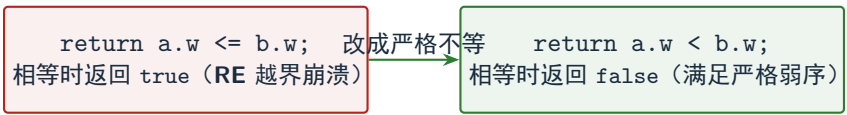
- **RE**：空向量算 `a.size()-1` 或 `pop_back()`；
- **RE**：未指定长度直接 `cin >> a[i]`；
- **WA**：多组数据忘用 `assign` 清空旧值。

本页要点

用 vector 记住三条：先开大小再下标访问、循环强转 (int)、多组用 assign。

pair 二元组与“严格弱序”唯一铁律

- 承接第七章结构体：上一章我们用 struct 打包多个字段；如果只绑两个字段（如区间端点、次数与单词），直接用 STL 自带的 pair<T1，T2>，免写结构体定义。
- 默认排序规则：sort 排 pair 时，先按 first 升序，first 相等再按 second 升序。
- 新术语——严格弱序：自定义比较器 cmp(a，b) 回答的是“*a* 是否严格排在 *b* 前面”。当 *a* 与 *b* 相等时，必须返回 false！



本页要点

自定义 sort 比较器只写 < 或 >，绝不写 <= 或 >=；用 pair 时默认升序即可。

离散化引入：值域 10^9 但只有 4 个数，桶数组开不下

生活场景：给跨度极广的学号/坐标重新编排座位

给出 4 个数 $a = [10^9, 5, 10^9, -7]$，你想用桶数组 `cnt[x]` 统计每个数出现的次数。

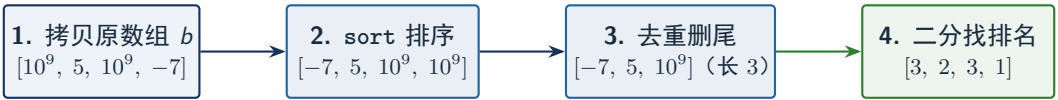
- 数值从 -7 跨到 10^9：数组下标既不能为负，也开不了 10^9 个格子（直接 **MLE** 爆内存）！
- 可实际上总共只有 3 个不同的数值：$-7 < 5 < 10^9$。

- 新术语——离散化：在不改变数值大小先后顺序的前提下，把大范围的数一对一映射成 $1, 2, \ldots, m$ 的小排名下标。
- 映射后：$-7 \mapsto 1,\ 5 \mapsto 2,\ 10^9 \mapsto 3$，只需要开大小为 3 的数组就能统计！

本页要点

只要题目只关心数值的“大小关系”或“是否相等”，而不关心具体差值，就可以离散化。

TikZ 状态图：离散化四步流程（排序 → 去重 → 查排名）



- 为什么要先拷贝一份 $b = a$ 再排序？因为我们要保留原数组 a 的输入顺序，只把 b 当作“去重词典”。
- `unique` 的关键动作：它只会把相邻重复元素往后挪，并返回新结尾迭代器，并不会缩短数组，必须配合 `erase` 剪掉尾巴！

本页要点

口诀：拷贝 b → `sort` 让相同值挨在一起 → `erase(unique)` → `lower_bound` 查排名。

纯手工跟踪表：离散化在 $a = [10^9, 5, 10^9, -7]$ 上的每一步

步骤	执行操作	数组 b 状态（或结果）	说明
1	拷贝 `b = a`	$[10^9, 5, 10^9, -7]$	保留原数组 a 不动
2	`sort(b.begin(), b.end())`	$[-7, 5, 10^9, 10^9]$	相同的 10^9 紧挨在一起
3	`it = unique(...)`	$[-7, 5, 10^9 \mid$ 脏数据$]$	`it` 指在下标 3，尾巴还在！
4	`b.erase(it, b.end())`	$[-7, 5, 10^9]$（长度 3）	剪掉无效后缀，词典完工
5	查每个 $a[i]$ 的排名 $+1$	a 变为 $[3, 2, 3, 1]$	下标 $0, 1, 2$ 加 1 变成 $1, 2, 3$

- 第 3 步如果不做第 4 步 `erase`，词典 b 的长度为 4 且第 4 格存着无意义数值，后续二分就会查错！

本页要点

记住 `unique` 只“搬家不缩容”，少了 `b.erase(...)` 必定留下脏尾巴。

模板代码与为什么对：离散化三行核心

```
vector<long long> b = a;                              // 1. 拷贝一份建去重词典
sort(b.begin(), b.end());                             // 2. 排序让相同值紧挨
b.erase(unique(b.begin(), b.end()), b.end());         // 3. 删除末尾多余后缀
for (int i = 0; i < (int)a.size(); ++i)               // 4. 映射为 1~b.size()
    a[i] = lower_bound(b.begin(), b.end(), a[i]) - b.begin() + 1;
```

为什么一定对？（三条性质）

1. 严格递增：排序去重后 $b[0 \dots m-1]$ 严格递增无重复；
2. 保序性：若 $x < y$，在 b 中下标满足 $\mathrm{pos}(x) < \mathrm{pos}(y)$；
3. 范围有界：排名加 1 后严格落在 $[1, m]$（$m \le n$）内。

上机验证与易错对照：离散化专项

上机自测三组数据

1. 含负数与重复：[100,-5,100,0] → [3,1,3,2]；
2. 全相同边界：[7, 7, 7] → [1, 1, 1]；
3. 单元素边界：[9] → [1]。

反事实追问：忘了写 sort 直接调 unique 会怎样？
答：分散两头的相同值删不掉，词典无序，二分全错（**WA**）！

常见错误对照

- **WA**：忘了 b.erase(...)，残留后缀破坏单调性；
- **WA**：直接把原数组排序去重，丢了原对应关系；
- **WA**：题目需真实差值却做了离散化，差值已丢失。

本页要点

离散化只保留“谁大谁小、是否相等”，不保留“相差多少”；用前先看题目要求。

生活场景与暴力痛点：在有序书架上查编号

生活场景：按编号排好序的 2×10^5 本书

图书馆书架上已经有 $n = 2 \times 10^5$ 本书按编号从小到大（允许同号）排好。接下来有 $q = 2 \times 10^5$ 次查询：

- 问编号 ≥ 5 的第一本书在哪；或者问编号刚好在 $[5, 8]$ 之间一共有几本书。

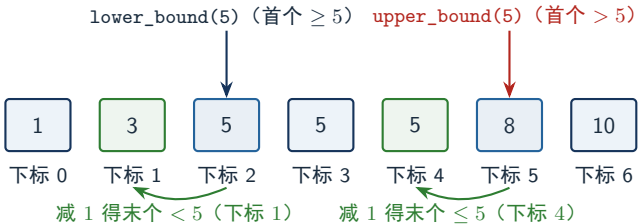
- 暴力做法为什么 **TLE**：每次从第 1 本书挨个往后找，单次最多看 2×10^5 本；$q = 2 \times 10^5$ 次一共要看 4×10^{10} 本，远远超过 1 秒 10^8 次上限，必超时（**TLE**）！
- 二分思路：利用“已经有序”的条件，每次直接看正中间那本书，一刀砍掉一半；2×10^5 本书最多只需看 $\log_2(2 \times 10^5) \approx 18$ 本！

本页要点

数组已经有序时，用 STL 二分查找可以把单次查询从 $O(n)$ 降到 $O(\log n)$。

核心定义与 TikZ 状态图：二分四问的几何位置

- `lower_bound(begin, end, x)`：在有序左闭右开区间中，返回指向第一个 $\geq x$ 元素的迭代器。
- `upper_bound(begin, end, x)`：返回指向第一个严格 $> x$ 元素的迭代器；都找不到时返回 `end()`。



本页要点

找 $\geq$ / $>$ 靠函数直接指；找 $<$ / $\leq$ 只需要从对应的结果往左退一格！

二分四问速查对照表

设非降序 `vector<int> a(n)`，`all(a)` 表示 `a.begin(), a.end()`：

查询目标	标准 STL 写法（得到下标）	示例（$x = 5$）	不存在时的结果
第 1 个 $\geq x$	`lower_bound(all(a), x) - a.begin()`	2（值 5）	`p == n`
第 1 个 $> x$	`upper_bound(all(a), x) - a.begin()`	5（值 8）	`p == n`
最后 1 个 $\leq x$	`(upper_bound(all(a), x) - a.begin()) - 1`	4（值 5）	`p == -1`
最后 1 个 $< x$	`(lower_bound(all(a), x) - a.begin()) - 1`	1（值 3）	`p == -1`

- 迭代器本身不是整数下标，必须减去 `a.begin()` 才得到 0-based 下标。
- 取值 `a[p]` 前必须先检查 `p < n`（向右不越界）或 `p >= 0`（向左不越界）。

本页要点

先查坐标，再判边界；`p == n` 说明没有后继，`p == -1` 说明没有前驱。

纯手工跟踪：lower_bound 在 $a=[1,3,5,5,5,8,10]$ 上找 $x=6$

在半开区间 $[l,r)$ 上找第一个 ≥ 6 的位置（初始 $l=0, r=7$，$mid=\lfloor (l+r)/2 \rfloor$）：

轮次	$l<r$?	$[l,r)$	mid	$a[mid]<6$?	下一步动作与原因
1	真（$0<7$）	$[0,7)$	3	$a[3]=5<6$ 真	$l\leftarrow 4$（左半全 <6，舍弃）
2	真（$4<7$）	$[4,7)$	5	$a[5]=8<6$ 假	$r\leftarrow 5$（5 可能是答案，保留）
3	真（$4<5$）	$[4,5)$	4	$a[4]=5<6$ 真	$l\leftarrow 5$（下标 4 也 <6，舍弃）
4	假（$5<5$）	$[5,5)$	-	不判断	退出循环，返回 $l=5$（值 8）

- 请重点看第 4 轮：当 l 与 r 相遇变成 $[5,5)$ 空区间时，条件 $l<r$ 为假，循环准时退出。

本页要点

手算跟踪一定要写到最后一次“条件为假”的退出步，这能验证二分绝不死循环。

为什么 lower_bound 一定对？（循环不变量）

不变量阶段	要检查的数学断言（区间 $[0, l)$ 严格 $< x$，且 $[r, n)$ 均 $\geq x$）
初始化	进入循环前 $l = 0, r = n$，$[0, 0)$ 为空、$[n, n)$ 为空，两条性质天然成立
保持	若 $a[mid] < x$，因非降序故 $[0, mid]$ 全 $< x$，设 $l = mid + 1$ 后不变量保持；否则 $a[mid] \geq x$，$[mid, n)$ 全 $\geq x$，设 $r = mid$ 后不变量保持
终止	循环结束时 $l == r$，两段区间在 l 处无缝拼接，故 l 必为第一个 $\geq x$ 的下标

- 极端情况 **1**（全 $< x$）：r 从未减小，结束时 $l = r = n$（对应 `a.end()`）。
- 极端情况 **2**（全 $\geq x$）：l 从未增大，结束时 $l = r = 0$（对应 `a.begin()`）。

本页要点

循环不变量把“二分为什么不漏解、不死循环”说得清清楚楚。

模板代码：区间 $[L, R]$ 计数与安全查询前驱

```cpp
// 在非降序数组 a 中统计落在 [L, R] 的元素个数
int cnt = upper_bound(a.begin(), a.end(), R)
        - lower_bound(a.begin(), a.end(), L);
auto it = lower_bound(a.begin(), a.end(), L); // 指向第一个 >= L
if (it == a.begin()) {
    cout << "没有比 L 小的数\n";              // 已在开头，禁止往左退！
} else {
    cout << "小于 L 的最大数 = " << *prev(it) << '\n'; // 安全退一格
}
```

为什么 upper_bound(R) - lower_bound(L) 正好是区间个数？

因为最后一个 $\le R$ 的位置是 upper_bound(R) - 1，第一个 $\ge L$ 的位置是 lower_bound(L)，两者包含的元素个数等于 $(\text{right} - 1) - \text{left} + 1 = \text{right} - \text{left}$！

上机验证与易错对照：lower_bound / upper_bound

上机自测（$a = [1, 3, 5, 5, 8]$）

1. 查 $[5, 8]$ → 数量 3，小于 5 最大数 3；
2. 查 $[1, 1]$ → 数量 1，无小于 1 的数；
3. 查 $[99, 100]$ → 数量 0，小于 99 最大数 8。

追问：`upper_bound(R)` 写成 `lower_bound(R)` 会怎样？
答：等于 R 的元素全漏掉（少计 **WA**）！

常见错误对照

- **WA**：未 `sort` 就二分，位置随机错乱；
- **RE**：`it == a.begin()` 时取 `*prev(it)` 崩；
- **TLE**：对 `set/map` 用全局 `lower_bound` 退化 $O(n)$！

本页要点

记住三句话：先排序再二分、左退先判 `begin()`、右查先判 `end()`。

生活场景与暴力痛点：一边插队一边问“左右邻居”

生活场景：不断有新成员报到的身高队列

陆续来 $n = 10^5$ 名同学，第 i 名同学身高是 x_i。他报到入队前，先要回答两个问题：

- 已经入队的同学里，比他矮的最高是谁（严格前驱）？比他高的最矮是谁（严格后继）？

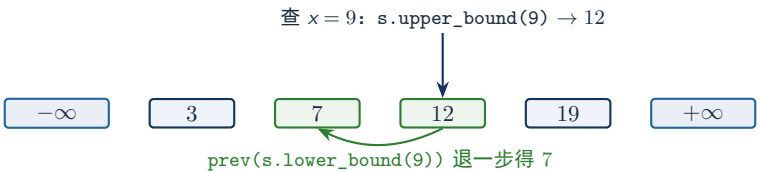
- 为什么 `vector` 不够用：如果用普通数组保持有序，每次把新数插进中间都要把后面的元素全部向右挪一格，单次 $O(n)$，10^5 次共约 5×10^9 步，必超时（**TLE**）！
- 新术语——`set`（平衡二叉搜索树 / 红黑树）：内部自动把元素排好序，插入、删除、查前驱后继全都只需 $O(\log n) \approx 17$ 步！

本页要点

静态查用 `vector + sort + lower_bound`；需要“边插入/删除边查前驱后继”就换 `set`。

核心定义与 TikZ 状态图：set 中的哨兵护栏法

- `set<T>`：自动去重且从小到大排列的集合；`multiset<T>`：允许相同元素重复出现的有序集合。
- 新术语——哨兵（**Sentinel**）：预先往集合里塞入 $-\infty$ 和 $+\infty$ 两个极大/极小的假值，把所有真实数据夹在中间，这样往前退一格或往后找时永远不会越界崩溃！



本页要点

两行初始化 `set<long long> s = {-INF, INF};`，彻底消灭 `begin()/end()` 越界崩溃！

set / multiset 四问速查表（含哨兵判定）

动态查找目标	set / multiset 成员函数	加 $-\infty, +\infty$ 哨兵后判定
第 1 个严格大于 x（后继）	`it = s.upper_bound(x)`	若 `*it == INF` 则无后继
第 1 个大于等于 x	`it = s.lower_bound(x)`	若 `*it == INF` 则无解
第 1 个严格小于 x（前驱）	`it = prev(s.lower_bound(x))`	永不越界，若 `*it == -INF` 则无前驱
第 1 个小于等于 x	`it = prev(s.upper_bound(x))`	永不越界，若 `*it == -INF` 则无解

- 成员函数铁律：必须写 `s.lower_bound(x)`，绝不能写成全局函数 `lower_bound(s.begin(), s.end(), x)`！
- 若题目数值范围为 $|x| \le 10^9$，哨兵 `INF` 可取 `4e18`（远超真实数值且不溢出 `long long`）。

本页要点

用点号调用 `s.lower_bound(x)` 走红黑树 $O(\log n)$；写错成全局函数会退化成 $O(n)$！

纯手工跟踪表：依次先查后插 $7 \to 3 \to 12 \to 9$

集合初始仅含哨兵 $\{-\infty, +\infty\}$，每次读入 x 先查前后继，再把 x 插入集合（若为哨兵则输出 -1）：

步	x	prev(lower)	upper	输出	插入后集合（不含哨兵）
0	-	-	-	-	$\emptyset$（内部仅 $\{-\infty, +\infty\}$）
1	7	$-\infty$	$+\infty$	-1 -1	$\{7\}$
2	3	$-\infty$	7	-1 7	$\{3, 7\}$
3	12	7	$+\infty$	7 -1	$\{3, 7, 12\}$
4	9	7	12	7 12	$\{3, 7, 9, 12\}$

- 如果第 1 步没有哨兵，集合为空时 `s.lower_bound(7)` 等于 `s.begin()`，调用 `prev` 当场崩溃！

本页要点

先查询、再插入，可以避免当前元素 x 自己干扰严格前驱与后继的结果。

模板代码与原理：set 哨兵前驱后继与 multiset 单删

```
const long long INF = 4e18;
set<long long> s = {-INF, INF};                 // 哨兵护栏：彻底消灭越界
auto it_gt = s.upper_bound(x);                  // 成员函数 O(log n)：首个 > x
auto it_lt = prev(s.lower_bound(x));            // 退一格：最大严格 < x
// 若用 multiset 存重复值，只想删掉“一个” x 必须先 find 再按迭代器删：
auto del = ms.find(x);
if (del != ms.end()) ms.erase(del);             // 错写 ms.erase(x) 会删光全部 x!
```

为什么全局 lower_bound 在 set 上会超时，而成员函数不会？

因为 vector 内存连续支持下标直接跳半程，而 set 的节点分散在树上，迭代器只能一步步 ++it；全局函数只能傻走 $O(n)$ 步，只有成员函数 s.lower_bound(x) 会从树根往下跳 $O(\log n)$ 步！

上机验证与易错对照：set / multiset 专项

上机自测三组数据（先查后插）

1. 普通：4 / 7 3 12 9 → -1 -1，-1 7，7 -1，7 12；
2. 最小：1 / 5 → -1 -1；
3. 全等：3 / 5 5 5 → 三行皆 -1 -1。

追问：改用全局 lower_bound(s.begin(), s.end(), x) 会怎样？
答：红黑树退化成 $O(n^2)$ 扫描直接 **TLE**！

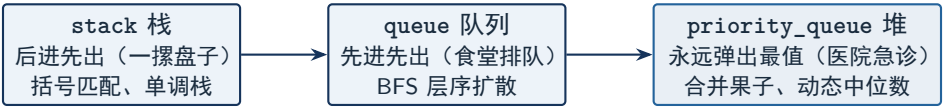
常见错误对照

- **RE**：无哨兵写 prev(s.lower_bound(x)) 退穿崩；
- **WA**：multiset 想单删却写 ms.erase(x) 全清空；
- **TLE**：用 ms.count(x) 统计高频值退化成 $O(n^2)$。

本页要点

记住四件事：加哨兵、成员函数二分、multiset 用迭代器单删、别用 count 查高频值。

生活场景选型：按顺序进出 vs 急诊每次取最值



- 暴力做法为什么 **TLE**：如果有 10^5 轮“加一个数并取出当前最小值”，每轮扫一遍数组需要 10^5 步，总计约 5×10^9 步，必超时（**TLE**）！
- 双端队列 `deque`：左右两端都能 $O(1)$ 进出，常用于滑动窗口最值与 0-1 BFS。

本页要点

只按进出顺序用 `stack` / `queue`；随时要拿“当前最大/最小值”就用 `priority_queue`。

核心定义与 TikZ 状态图：完全二叉树与小根堆上浮

- 新术语——堆（`priority_queue`）：用一棵完全二叉树（从上到下、从左到右填满的树）存数据，只保证父节点永远比子节点更优，因此每次只需沿着树高走 $\log_2 n \approx 17$ 步就能维护最值！



1. `push(5)`　2. `push(2)` 上浮　3. `push(8)` 留右　4. `pop()` 后 `top=5`

本页要点

堆顶永远是当前极值，但树中其他位置并不保证完全排好序；因此堆不支持下标遍历。

纯手工跟踪表：小根堆依次执行 5 步操作

维护小根堆 pq（每次保证 pq.top() 是当前堆里的最小值）：

步骤	执行操作	堆内元素	当前 top()	调整动作与边界检查
0	初始空堆	$\{\}$	禁止访问！	pq.empty() 为真，取顶会 **RE**
1	pq.push(5)	$\{5\}$	5	只有一个元素，直接当堆顶
2	pq.push(2)	$\{2,5\}$	2	$2<5$，一路上浮替换堆顶
3	pq.push(8)	$\{2,5,8\}$	2	$8>2$，停在右儿子不动
4	pq.pop()	$\{5,8\}$	5	弹出 2，8 补顶后与 5 交换下沉
5	pq.pop()	$\{8\}$	8	弹出 5，仅剩 8 为堆顶

- 每次插入或弹出后，堆顶都立即恢复成剩余元素的最小值。

本页要点

调用 pq.top() 或 pq.pop() 之前，永远先确认 !pq.empty()。

模板代码：大根堆、小根堆与反直觉的比较器

```cpp
priority_queue<int> big_pq;                          // 大根堆：top() 最大
priority_queue<int, vector<int>, greater<int>> small_pq; // 小根堆

struct Task {
    int id, cost;
    // 堆的 < 问“自己是否不如对方优先”：cost 大不如小，故写 >
    bool operator<(const Task &o) const {
        return cost > o.cost;                        // cost 小的浮到堆顶
    }
};
priority_queue<Task> task_pq;                        // 小根任务堆
```

为什么堆的比较方向恰好和 sort 相反？

在 sort 里 a < b 表示“*a* 放前面”；而在 priority_queue 里，a < b 为真表示“*a* 优先级低于 *b*，请让 *b* 浮到堆顶”，所以想要小的浮顶必须写 cost > o.cost！

为什么堆顶一定是最值？以及什么时候堆不够用

- 堆不变量（正确性）：每次 push 把新元素放末尾再一路向上比较交换，每次 pop 把末尾搬到堆顶再和较优儿子向下交换。因为任意父亲都比两个儿子优，所以全树最值一定停在树根（top()）。
- 堆的局限性（不能直接删中间值）：堆里只保证父子大小，找不到某一个指定值的下标，因此 priority_queue 没有 find 或 erase！
- 如果题目要求“删除任意指定值”怎么办？
  1. 改用 multiset（既能找最值 *ms.begin() / *ms.rbegin()，又能 erase(ms.find(x))）；
  2. 或者用“懒删除”：另开一个数组/堆记录待删值，等它冒到堆顶时再一起 pop() 扔掉。

本页要点

只取删堆顶用 priority_queue（常数更小）；要删任意中间元素就换 multiset。

上机验证与易错对照：priority_queue 专项

上机自测三组数据（小根堆）

1. 普通：入堆 5，2，8 全弹出 → 2 5 8；
2. 最小：只入 9 → top()=9，弹出后为空；
3. 全等：4，4，4 → 弹出三次均为 4。

追问：cost > o.cost 错写成 cost < o.cost 会怎样？
答：变大根堆，每次弹出最大 cost（**WA**）！

常见错误对照

- **CE**：小根堆漏写中间的 vector<int>；
- **RE**：空堆未判 !pq.empty() 直接取 top()；
- **WA**：按 sort 的习惯写反结构体 operator<。

本页要点

记住小根堆口诀：greater<int> 配三参数，结构体里反向写 >。

生活场景与暴力痛点：给字符串和超大坐标计数

生活场景：统计 10^5 个英文单词或学号的出现次数

输入 $n = 10^5$ 个单词，如 `"apple"`、`"cat"`、`"dog"`，统计每个单词出现了几次。

- 普通数组为什么不行：数组下标只能是 $0, 1, 2, \ldots$ 的整数，写不出 `cnt["apple"]`；就算把单词全存下来，每次从头对比是否出现过，共需 $10^5 \times 10^5/2 \approx 5 \times 10^9$ 步，必超时（**TLE**）！
- 新术语——键值对（`key -> value`）：`map<string, int>` 就像一本自动按字典序排好的查词字典，把“单词（键）”映射到“次数（值）”，单次查改仅 $O(\log n)$！

本页要点

当数组下标变成字符串、负数或高达 10^9 的整数时，用 `map` 当“超级数组”。

核心定义与 TikZ 状态图：mp[k] 查询的隐藏副作用

- `map<K, V>`：底层也是红黑树，键自动升序排列，单次增删查严格 $O(\log n)$。
- `unordered_map<K, V>`：底层是哈希桶（把键算成散列编号直接存格子里），均摊 $O(1)$，但遇到构造卡哈希数据时最坏可能退化成 $O(n)$！
- `bitset<N>`：把 N 个 $0/1$ 开关压进 64 位整数里批量位运算，速度提升 64 倍！



本页要点

只查询键是否存在时，千万别写 `if (mp[k] > 0)`，它会顺手塞进 `mp[k] = 0`！

纯手工跟踪表：map 词频统计与 count() 安全查询

依次处理单词序列 ["cat", "dog", "cat", "apple"]，最后查询 "bird" 是否出现：

步骤	执行操作	内部有序状态（键升序）	size()	说明
1	freq["cat"]++	cat:1	1	不存在则先建 0 再加 1
2	freq["dog"]++	cat:1, dog:1	2	按字典序自动放在 cat 后
3	freq["cat"]++	cat:2, dog:1	2	找到已有键，次数加 1
4	freq["apple"]++	apple:1, cat:2, dog:1	3	自动排到最前面
5	freq.count("bird")	apple:1, cat:2, dog:1	3	返回 0，不插入新键！

- 如果第 5 步误写了 freq["bird"] > 0，size() 会从 3 偷偷变成 4，后续遍历整个 map 时就会多印出一行 bird 0！

本页要点

修改计数用 freq[s]++；只判断存在与否必须用 freq.count(s) 或 freq.find(s)。

模板代码：map 安全计数与 bitset 布尔背包加速

```cpp
map<string, int> freq;                  // 单词 -> 出现次数（按键升序）
freq["cat"]++;                          // 累加计数
if (freq.count("bird")) {               // 只读查询：不会插入 0
    cout << freq["bird"] << '\n';
}
bitset<50005> can;                      // 每一位表示该总和能否凑出
can[0] = 1;                             // 总和 0 一定可达
for (int w : weights) {
    can |= (can << w);                  // 整行左移 w 位再并集，快 64 倍
}
```

本页要点

计数首选 map；遇到“只问能否凑出和 S”的布尔递推，bitset 可直接提速 64 倍。

map vs unordered_map 选型与 bitset 适用条件

容器	是否有序	时间复杂度	比赛使用建议
`map<K, V>`	按键升序，支持二分	严格 $O(\log n)$	最稳首选，绝不超时
`unordered_map`	无序，不能二分	均摊 $O(1)$，最坏 $O(n)$	常数小，但需防卡哈希数据
`bitset<N>`	固定 N 位 $0/1$ 串	整行运算 $O(N/64)$	需编译期固定上限 N，只存布尔

- 适用条件提醒：`bitset<N>` 的长度 N 必须是编译期常量（如 `50005`），不能写变量 `bitset<n>`；若需要动态长度的整数统计，仍用 `vector` 或 `map`。

本页要点

普通竞赛题 $n \le 2 \times 10^5$ 时优先选 `map`，稳健不翻车；真卡常再换 `unordered_map`。

上机验证与易错对照：map / unordered_map / bitset

上机自测三组数据

1. 词频：["b","a","b"] → a:1, b:2；
2. 空查：count("x") → 0，size() 不变；
3. bitset：[2,3] → can[5]=1，can[4]=0。

追问：写 if (mp[k] > 0) 查 10^5 个新键会怎样？
答：塞进 10^5 个垃圾 0 节点，触发 **MLE** 或 **WA**！

常见错误对照

- **WA/MLE**：用 mp[k] 判存在，破坏 size()；
- **TLE**：unordered_map 遇卡哈希退化成 $O(n^2)$；
- **CE**：bitset<n> 误用运行时变量作长度。

本页要点

用 map 谨记：只读用 count，修改用 []，稳妥优先不盲目用 unordered_map。

例 0 | [入门] 查找第一个不小于：先读懂要求

大白话题意（对应 题目/补充题目/二分/查找第一个不小于）

给你一排已经从小到大（非递减）排好队的整数 $a_1, a_2, \ldots, a_n$（$n \le 2 \times 10^5$）。接下来有 q 次询问（$q \le 2 \times 10^5$），每次给出一个整数 x，请你找出第一个满足 $a_i \ge x$ 的位置编号 i；如果所有数都比 x 小，输出 $n+1$。

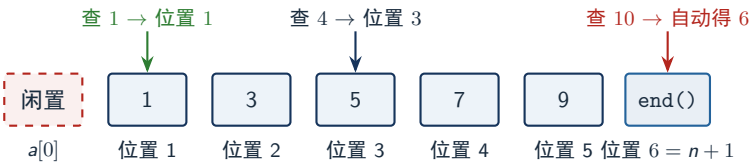
1. 输入：第 1 行两个整数 n, q；第 2 行 n 个非递减整数；接下来 q 行每行一个询问 x。
2. 处理：每次定位第一个 $\ge x$ 的 1-based 下标。
3. 输出：每次输出一个位置编号并换行。

原题样例

输入 `5 3 / 1 3 5 7 9 / 4 / 1 / 10` ⇒ 依次输出 3、1、6。

例 0 | 怎么想到的与 TikZ 建模图

- 暴力为什么不行：每次从头扫一遍寻找 $a_i \geq x$，最多要看 $n \times q = 4 \times 10^{10}$ 次，必超时（**TLE**）。
- 性质与模型：序列已保证非递减，直接用 `lower_bound`；为了对齐题目 1-based 编号，开 $n+1$ 格空出 $a[0]$，从 `a.begin() + 1` 开始找！



本页要点

查找不到时 `lower_bound` 返回 `a.end()`，减去 `a.begin()` 正好自动等于 $n+1$，零特判！

例 0 ｜样例手推跟踪表（序列 $a = [-, 1, 3, 5, 7, 9]$）

在半开下标区间 $[1, 6)$ 上执行 `lower_bound(a.begin()+1, a.end(), x)`：

询问 x	区间 $[l, r)$ 二分过程	停在的迭代器	减去 `a.begin()` 输出
4	$[1,6) \xrightarrow{a[3]=5\geq 4} [1,3) \xrightarrow{a[2]=3<4} [3,3)$	`a.begin() + 3`	**3**（对应值 5）
1	$[1,6) \xrightarrow{a[3]\geq 1} [1,3) \xrightarrow{a[2]\geq 1} [1,2) \xrightarrow{a[1]\geq 1} [1,1)$	`a.begin() + 1`	**1**（对应值 1）
10	$[1,6) \xrightarrow{a[3]<10} [4,6) \xrightarrow{a[5]<10} [6,6)$	`a.end()`	**6**（恰好等于 $n+1$）

- 即使全比 x 小（询问 10），循环终止时 $l = r = 6$，依然不需要写额外的 `if` 特判。

本页要点

把 $a[0]$ 空出来只当作占位起跑线，查找范围就严格对齐 $1 \sim n$。

例 0 | 核心步骤表与相对模板改动

按顺序计算	纸上得到	核对问题
1. 开 n+1 长数组	a[1..n] 存序列	数值范围到 $\pm 10^9$，用 long long 更稳
2. 跳过 0 号位查找	查找 [a.begin()+1, a.end())	若从 a.begin() 找，负数查询会误停在 $a[0]=0$！
3. 指针相减得下标	pos = it - a.begin()	全部小于 x 时是否正好得 $n+1$？

相对基础模板改了哪里？

基础二分模板从 a.begin() 开始找并得 0-based 下标；本题将数组多开 1 格、左起点改成 a.begin() + 1，并直接用 it - a.begin() 同时搞定 1-based 编号和未找到输出 $n+1$。

例 0 | 核心代码（对应完整源码 code/stl_ex0_first_ge.cpp）

```cpp
int main() {
    ios::sync_with_stdio(false); cin.tie(nullptr);
    int n, q; if (!(cin >> n >> q)) return 0;
    vector<long long> a(n + 1);              // 开 n+1 格，空出 a[0]
    for (int i = 1; i <= n; ++i) cin >> a[i]; // 读入 1-based 非递减序列
    while (q--) {
        long long x; cin >> x;
        int pos = lower_bound(a.begin() + 1, a.end(), x) - a.begin();
        cout << pos << '\n';                 // 找不到自动输出 n+1
    }
    return 0;
}
```

本页要点

头文件与 using namespace std; 见 code/stl_ex0_first_ge.cpp。

例 0 | 易错对照与三组自测边界数据

常见错误对照

- **WA**：误用 `upper_bound`（会跳过等于 x 的位置，例如查 5 错输出位置 4）；
- **WA**：从 `a.begin()` 开始找，询问负数 -10 时停在闲置的 $a[0]=0$；
- **TLE**：用 `endl` 换行导致 2×10^5 次刷新缓冲区超时。

三组必测边界数据（实测通过）

1. 最小边界：`1 1 / 5 / 5` $\rightarrow$ `1`；
2. 大量重复：`4 1 / 2 2 2 5 / 2` $\rightarrow$ 首个 `1`；
3. 负数与越界：`3 2 / -5 0 6 / -10 9` $\rightarrow$ `1` 和 `4`。

本页要点

提交前拿“负数、全相等、比最大值还大”这三组数据测一遍，不出错再交。

例 1 | [基础] 离散化与区间频次统计：先读懂要求

大白话题意

给出 n 个可能含负数、重复、绝对值高达 10^9 的整数 $a_0, \ldots, a_{n-1}$（$n, q \le 2 \times 10^5$）。随后有 q 次询问，每次给出闭区间 $[L, R]$，求序列中有多少个数满足 $L \le a_i \le R$（同时需支持对原值建立去重词典以便离散化映射）。

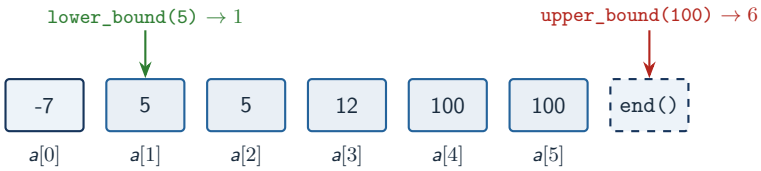
1. 输入：第 1 行 n, q；第 2 行 n 个整数；接下来 q 行每行两个整数 L, R。
2. 处理：先建去重有序词典 `b`，再将 `a` 排序后对每次询问算 $[L, R]$ 内的元素个数。
3. 输出：每次输出一个非负整数并换行。

小样例

输入 `6 3 / 100 5 100 -7 12 5 / 5 100 / 0 4 / 12 12` $\Rightarrow$ 输出 5、0、1。

例 1 | 怎么想到的与 TikZ 建模图

- 暴力为什么不行：若开桶统计需开到 10^9 爆内存（**MLE**）；若每次遍历 a 数个数要 $O(nq)$ 必超时（**TLE**）。
- 思路转化：先用 `sort + unique + erase` 把去重词典 b 建好，再把原数组 a 排序；于是 $[L, R]$ 内的元素会连续挤在一起，区间长度就是 `upper_bound(R) - lower_bound(L)`！



本页要点

排序后“落在 $[L, R]$ 内的元素”一定是一整段连续区间，右开边界减左闭边界即个数：$6 - 1 = 5$！

例 1 | 样例手推跟踪表（$a = [100, 5, 100, -7, 12, 5]$）

词典去重后 $b = [-7, 5, 12, 100]$；原数组排序后 $a = [-7, 5, 5, 12, 100, 100]$（长 6）：

询问 $[L, R]$	`lower_bound(L)` 下标	`upper_bound(R)` 下标	相减结果（元素列表）
$[5, 100]$	1（值 5）	6（`end()`）	$6-1 = \mathbf{5}(5, 5, 12, 100, 100)$
$[0, 4]$	1（值 $5 \geq 0$）	1（值 $5 > 4$）	$1 - 1 = \mathbf{0}$（无元素命中）
$[12, 12]$	3（值 12）	4（值 $100 > 12$）	$4 - 3 = \mathbf{1}$（只有 12）

- 注意第 2 组 $[0, 4]$：0 和 4 都不在数组里，但两个函数同时停在下标 1，相减正好自动得到 0！

本页要点

当区间里没有任何元素时，`upper_bound(R)` 和 `lower_bound(L)` 指向同一位置，相减为 0。

例 1 | 核心步骤表与相对模板改动

按顺序计算	纸上得到	核对问题
1. 建去重词典 b	排序 + 去重后的不同值集合	unique 之后有没有写 b.erase?
2. 排序原数组 a	非降序排列的 $a[0..n-1]$	不能直接在去重后的 b 上数个数（会丢重复）!
3. 区间端点作差	upper(r) - lower(l)	右端点用的是 upper_bound 而不是 lower_bound?

相对基础模板改了哪里?

在离散化去重词典模板之上，保留包含全部重复值的原数组 a 并对其排序，再用 upper_bound(r) - lower_bound(l) 一次算出闭区间 $[l, r]$ 内所有重复元素总个数。

例 1 | 核心代码（1/2）：读入与去重词典（code/stl_ex1_bounds.cpp）

```
int main() {
    ios::sync_with_stdio(false); cin.tie(nullptr);
    int n, q; if (!(cin >> n >> q)) return 0;
    vector<long long> a(n);                          // 原数组，保留重复元素
    for (int i = 0; i < n; ++i) cin >> a[i];
    // 1. 排序 + 去重建立离散化词典 b
    vector<long long> b = a;
    sort(b.begin(), b.end());
    b.erase(unique(b.begin(), b.end()), b.end());
```

本页要点

第 (1/2) 段先把原数组读入，并复制出一份排好序且去重的词典 b。

例 1 | 核心代码（2/2）：排序与区间频次查询（code/stl_ex1_bounds.cpp）

```
    // 2. 在排序后的原数组 a 上查询落在 [l, r] 内的元素个数
    sort(a.begin(), a.end());
    while (q--) {
        long long l, r;
        cin >> l >> r;
        int cnt = upper_bound(a.begin(), a.end(), r)
                - lower_bound(a.begin(), a.end(), l);
        cout << cnt << '\n';
    }
    return 0;
}
```

本页要点

第 (2/2) 段直接在有序原数组 a 上作差计数，若想算“不同值个数”则改在 b 上查。

例 1 | 易错对照与三组自测边界数据

常见错误对照

- **WA**：错在去重数组 `b` 上做区间计数，把重复元素漏数了；
- **WA**：右端点错写成 `lower_bound(r)`，导致等于 r 的数全被漏掉；
- **WA**：询问 $L > R$ 时未注意（若题面允许 $L > R$ 需先判 `if (l > r) cnt = 0`）。

三组必测边界数据（实测通过）

1. 单元素：`1 1 / 7 / 7 7` $\rightarrow$ `1`；
2. 全相同：`4 1 / 5 5 5 5 / 5 5` $\rightarrow$ `4`；
3. 全越界：`3 1 / 1 3 8 / 10 20` $\rightarrow$ `0`。

本页要点

分清“统计总出现次数（查原数组 `a`）”与“统计不同值种类（查词典 `b`）”！

例 2 | [基础] 动态最近邻查询：先读懂要求

大白话题意（对应经典题“营业额统计”的核心前置步骤）

有 n 个人（$n \le 10^5$）依次进入房间，第 i 个人手里拿着数字 x_i（$|x_i| \le 10^9$）。在他进入房间时（还没把 x_i 放进集合前），立刻输出房间里已有数字中严格小于 x_i 的最大数与严格大于 x_i 的最小数；某侧不存在则输出 -1。

1. 输入：第 1 行整数 n；第 2 行 n 个整数 $x_1, \ldots, x_n$。
2. 处理：每读一个 x_i，先在有序集合里查前驱和后继，再把 x_i 插入集合。
3. 输出：共 n 行，每行两个整数（严格前驱、严格后继）。

小样例

输入 `4 / 7 3 12 9` $\Rightarrow$ 依次输出 `-1 -1`、`-1 7`、`7 -1`、`7 12`。

例 2 | 怎么想到的与 TikZ 建模图

- 暴力为什么不行：每来一个数就扫一遍前面已入房间的所有数找最近邻，总计 $n(n-1)/2 \approx 5 \times 10^9$ 次，必超时（**TLE**）。
- 建模思路：用 `set<long long>` 动态维护当前集合，预先塞入 $-\infty$ 和 $+\infty$ 两个哨兵，确保 `prev(s.lower_bound(x))` 永不穿出左边界！



本页要点

真实数据永远被 $-\infty$ 与 $+\infty$ 夹在中间，一行代码就能同时搞定“空集合”和“最左元素”边界！

例 2 | 样例手推跟踪表（输入 $[7, 3, 12, 9]$）

初始 `s = {-INF, +INF}`，每步严格执行“先查后插”：

步数	读入 x	`*it_lt`（前驱）	`*it_gt`（后继）	输出	本步结束后集合（不含哨兵）
1	7	$-\infty$	$+\infty$	`-1 -1`	$\{7\}$
2	3	$-\infty$	7	`-1 7`	$\{3, 7\}$
3	12	7	$+\infty$	`7 -1`	$\{3, 7, 12\}$
4	9	7	12	`7 12`	$\{3, 7, 9, 12\}$

- 在第 4 步查 9 时，`s.lower_bound(9)` 停在 12，往左退一格是 7；`s.upper_bound(9)` 也停在 12。

本页要点

当查出的前驱为 `-INF` 或后继为 `INF` 时，用三目运算符还原输出 `-1`。

例 2 | 核心步骤表与相对模板改动

按顺序计算	纸上得到	核对问题
1. 初始化放哨兵	`s.insert(-INF);` `s.insert(INF);`	`INF = 4e18` 是否远大于 $\lvert x_i \rvert \le 10^9$？
2. 先用成员函数查	`it_gt, it_lt`	千万别在插入 x 之后才查！
3. 转换哨兵并插入	遇哨兵输出 `-1`，然后 `s.insert(x)`	成员函数是否写成 `s.upper_bound`？

相对基础模板改了哪里？

在 `set` 哨兵前驱后继模板基础上，增加了“把哨兵值 `-INF/INF` 还原成 `-1`”的输出格式转换，并严格把 `s.insert(x)` 放在输出之后。

例 2 | 核心代码（1/2）：哨兵初始化（code/stl_ex2_set_neighbor.cpp）

```cpp
int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int n;
    if (!(cin >> n)) return 0;

    const long long INF = 4e18;                 // 远超 10^9 的正负哨兵
    set<long long> s;
    s.insert(-INF);                             // 左哨兵：防止 prev 退穿开头
    s.insert(INF);                              // 右哨兵：兜底无后继的情况
```

本页要点

第 (1/2) 段用 4e18 设好正负哨兵，避免后续 prev 越界产生 RE。

例 2 | 核心代码（2/2）：先查后插循环（code/stl_ex2_set_neighbor.cpp）

```
    for (int i = 0; i < n; ++i) {
        long long x; cin >> x;
        auto it_gt = s.upper_bound(x);       // 成员函数 O(log n)：首个 > x
        auto it_lt = prev(s.lower_bound(x)); // 退一格：最后一个 < x
        long long prev_lt = (*it_lt == -INF) ? -1 : *it_lt;
        long long next_gt = (*it_gt == INF) ? -1 : *it_gt;
        cout << prev_lt << ' ' << next_gt << '\n';
        s.insert(x);                         // 先回答查询，最后再插入 x
    }
    return 0;
}
```

本页要点

第 (2/2) 段全程使用 s．成员函数二分，总复杂度 $O(n \log n)$ 稳过 10^5！

例 2 | 易错对照与三组自测边界数据

常见错误对照

- **TLE**：写成全局 `upper_bound(s.begin(), s.end(), x)`，退化成 $O(n^2)$；
- **RE**：无哨兵直接 `prev(s.lower_bound(x))`，遇最小值崩溃；
- **WA**：先 `s.insert(x)` 再查，会干扰严格前驱或后继。

三组必测边界数据（实测通过）

1. 单元素：`1 / 10` → `-1 -1`；
2. 全相等：`3 / 5 5 5` → 三行均为 `-1 -1`；
3. 严格递减：`3 / 9 6 3` → `-1 -1`, `-1 9`, `-1 6`。

本页要点

做完本题可顺手挑战同思路竞赛题《营业额统计》（题目/算法提高篇/.../营业额统计）。

例 3 | [提高] 对顶堆维护动态中位数：先读懂要求

大白话题意

依次读入 n 个整数（$n \le 10^5$）。每当读入到奇数个元素（即第 $1,3,5,\ldots$ 个数刚进来时），立即输出前 i 个数从小到大排序后的中位数（正中间第 $(i+1)/2$ 小的那个数）。

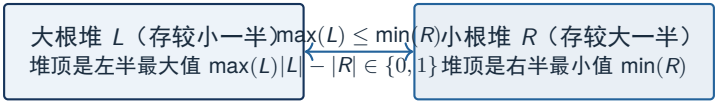
1. 输入：第 1 行整数 n；第 2 行 n 个整数。
2. 处理：边读边维护前 i 个数的中位数。
3. 输出：每到奇数步输出一行当前中位数。

小样例

输入 `5 / 1 9 2 8 5` $\Rightarrow$ 第 $1,3,5$ 步分别输出 `1`、`2`、`5`。

例 3 | 怎么想到的与 TikZ 建模图

- 暴力为什么不行：每来一个数就把前 i 个数重新 `sort` 一遍，总复杂度 $O(n^2 \log n)$，$n = 10^5$ 必超时（**TLE**）。
- 新术语——对顶堆：把已读的数切两半：较小的一半放大根堆 L（堆顶是左半最大），较大的一半放小根堆 R（堆顶是右半最小），中间接缝处的 `L.top()` 就是中位数！



本页要点

两条不变量：左半所有数 $\leq$ 右半所有数，且左堆个数永远等于右堆或比右堆多 1！

例 3 | 样例手推跟踪表（输入 $[1,9,2,8,5]$）

每次新数先根据与 `L.top()` 大小入堆，再检查是否需要搬运堆顶以维持 $|L|-|R|\in\{0,1\}$：

步数 i	新数 x	插入后 L（大根）	插入后 R（小根）	平衡检查与搬运	奇数步输出
1	1	$\{1\}$	$\{\}$	$\lvert L\rvert=1,\lvert R\rvert=0$，合法	**1**
2	9	$\{1\}$	$\{9\}$	$9>1$ 入 R，$\lvert L\rvert=\lvert R\rvert=1$	-
3	2	$\{1,2\}$	$\{9\}$	$2>1$ 原本入 R？错！看下↓	-

等等！第 3 步手推里藏着关键一跳：仔细看 $x=2$！

第 3 步读入 $x=2$ 时，因为 $2>L.\text{top}()=1$，所以 2 先进入 $R=\{2,9\}$；此时 $|R|=2>|L|=1$ 违反了尺寸不变量，触发平衡动作：把 R 的堆顶 2 搬回 L！于是 $L=\{1,2\},R=\{9\}$，输出 `L.top() = 2`！同理第 4 步 8 入 $R=\{8,9\}$，第 5 步 $5\le 2$ 假 $\rightarrow$ 入 $R=\{5,8,9\}$ 再把 5 搬回 $L=\{1,2,5\}$，输出 5！

例 3 | 核心步骤表与相对模板改动

按顺序计算	纸上得到	核对问题
1. 决定新数进哪边	`L.empty() || x <= L.top()` 入 L	先判 `L.empty()` 了吗？空堆取顶会 RE！
2. 检查尺寸差是否失衡	哪边多就搬它的 `top()` 去对面	左多 2 个移去右；右比左多 1 个移去左
3. 奇数步输出中位数	`if (i % 2 == 1) cout << L.top()`	左右数量相等时总数为偶数，不输出

相对基础模板改了哪里？

把一个大根堆和一个小根堆“背靠背”组合起来，每插入 1 个元素最多只触发 1 次堆顶搬运，就地在 $O(\log n)$ 内维护出了动态中位数。

例 3 | 核心代码（1/2）：双堆定义与入堆（code/stl_ex3_median_heap.cpp）

```cpp
int main() {
    ios::sync_with_stdio(false); cin.tie(nullptr);
    int n; if (!(cin >> n)) return 0;
    priority_queue<int> left_max;                          // 大根堆：较小一半
    priority_queue<int, vector<int>, greater<int>> right_min; // 小根堆
    for (int i = 1; i <= n; ++i) {
        int x; cin >> x;
        if (left_max.empty() || x <= left_max.top())
            left_max.push(x);                              // <= 左顶放左边
        else
            right_min.push(x);                             // 否则放右边
```

本页要点

第 (1/2) 段定义对顶双堆并分流入堆，`left_max.empty()` 是防 RE 关键。

例 3 | 核心代码（2/2）：尺寸平衡与输出（code/stl_ex3_median_heap.cpp）

```
        // 维持 size(left_max) == size(right_min) 或 +1
        if (left_max.size() > right_min.size() + 1) {
            right_min.push(left_max.top()); left_max.pop();
        } else if (right_min.size() > left_max.size()) {
            left_max.push(right_min.top()); right_min.pop();
        }
        if (i % 2 == 1) cout << left_max.top() << '\n'; // 奇数步输出
    }
    return 0;
}
```

本页要点

第 (2/2) 段每次最多搬运一次堆顶，总复杂度 $O(n \log n)$ 稳过 $n = 10^5$！

例 3 | 易错对照与三组自测边界数据

常见错误对照

- **RE**：漏写 `left_max.empty()` 直接写 `x <= left_max.top()`，第 1 步就崩溃；
- **WA**：只写了左边太长往右挪，漏写右边太长往左挪，遇到递增序列中位数永远卡在第 1 个数；
- **WA**：把右边错写成大根堆，破坏 $\max(L) \le \min(R)$。

三组必测边界数据（实测通过）

1. 严格递增：`5 / 1 2 3 4 5` $\rightarrow$ `1, 2, 3`；
2. 严格递减：`5 / 5 4 3 2 1` $\rightarrow$ `5, 4, 3`；
3. 全相等值：`3 / 7 7 7` $\rightarrow$ `7, 7`。

本页要点

自测对顶堆必测“严格递增”和“严格递减”两组，专门检验左右双向搬运！

例 4 | [基础] 统计词频与多关键字排序：先读懂要求

大白话题意

依次读入 n 个英文单词（$n \le 10^5$），统计每个不同单词出现的总次数，然后按出现次数从多到少输出；若两个单词出现次数相同，按字典序从小到大输出（每行输出“单词空格次数”）。

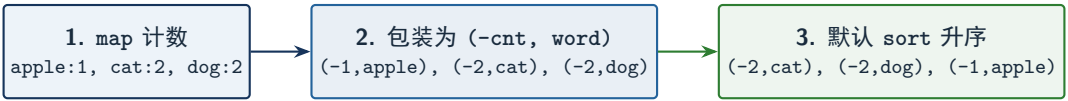
1. 输入：第 1 行整数 n；接下来给出 n 个不含空格的字符串单词。
2. 处理：统计每个字符串出现次数，并按“次数降序、单词升序”排序。
3. 输出：若干行，每行格式为 `word cnt`。

小样例

输入 `5 / cat dog cat apple dog` ⇒ 依次输出 `cat 2`、`dog 2`、`apple 1`。

例 4 | 怎么想到的与 TikZ 建模图

- 暴力为什么不行：每读一个单词就扫数组查之前是否出现过是 $O(n^2)$ 会 **TLE**；而只用 map 只能按“单词字母序”排，没法直接按“次数”排！
- 建模巧招：先用 map<string, int> 计数；再把每条记录包装成 pair(-次数, 单词) 放进 vector——用负次数把“次数降序”直接变成默认升序，免写自定义比较器！



本页要点

(-次数, 单词) 配默认升序 sort，一次同时实现“第一关键字降序、第二关键字升序”！

例 4 | 样例手推跟踪表（输入 cat dog cat apple dog）

阶段	操作内容	中间数据状态	说明
1	freq[s]++ 读完 5 词	apple:1, cat:2, dog:2	map 自动按键字典序合并
2	转为 (-cnt, word)	(-1,apple), (-2,cat), (-2,dog)	存进 vector<pair<int, string>>
3	调用默认 sort	(-2,cat), (-2,dog), (-1,apple)	$-2 < -1$；同为 -2 时 cat < dog
4	遍历输出 word 与 -neg_c	cat 2 / dog 2 / apple 1	记得把负次数取反还原成正数！

- 为什么把 -cnt 放在 pair 的第一个位置（first）？因为 pair 总是先比 first 再比 second！

本页要点

多关键字排序口诀：谁优先级高谁放 first，需要降序就在前面加负号！

例 4 | 核心步骤表与相对模板改动

按顺序计算	纸上得到	核对问题
1. map 统计词频	每个单词对应总次数	字符串作为键时默认区分大小写
2. 压入 pair 列表	items.emplace_back(-cnt, word)	第一关键字必须是 -cnt，别写反！
3. 排序并取反输出	先 sort 再输出 word 和 -neg_c	输出时有没有把 -neg_c 还原成正数？

相对基础模板改了哪里？

把 map 词频统计模板和 pair 多关键字排序结合：用结构化绑定 for (const auto &[w, c] : freq) 导出数据并利用负号反转主关键字排序方向。

例 4 | 核心代码（对应完整源码 code/stl_ex4_map_sort.cpp）

```cpp
int main() {
    ios::sync_with_stdio(false); cin.tie(nullptr);
    int n; if (!(cin >> n)) return 0;
    map<string, int> freq;                      // 记录单词出现次数
    for (int i = 0; i < n; ++i) { string s; cin >> s; freq[s]++; }
    vector<pair<int, string>> items;
    items.reserve(freq.size());                 // 预留空间减少扩容
    for (const auto &[w, c] : freq) items.emplace_back(-c, w);
    sort(items.begin(), items.end());           // 先按 -c，再按 w 升序
    for (const auto &[nc, w] : items) cout << w << ' ' << -nc << '\n';
    return 0;
}
```

本页要点

完整源码见 code/stl_ex4_map_sort.cpp，用负次数免写自定义比较器。

例 4 | 易错对照与三组自测边界数据

常见错误对照

- **WA**：错把 (word, -cnt) 放进 pair，变成先按字母排、次数根本没排上去；
- **WA**：输出时忘了把 nc 取反成 -nc，结果打印出负数次数；
- **CE**：结构化绑定 auto &[w, c] 需要 -std=c++17 编译标准。

三组必测边界数据（实测通过）

1. 全同单词：3 / a a a → a 3；
2. 同频字典序：2 / z a → a 1 后 z 1；
3. 混排词频：5 / b a b c b → b 3 / a 1 / c 1。

本页要点

如果不想使用负数技巧，也可以用第七章学过的结构体 + 严格弱序 lambda 来写比较器。

STL 高频错误对照表

题号 / 场景	常见错误写法	正确做法
vector 遍历	`i <= v.size() - 1`（空数组下溢）	强转 `i < (int)v.size()`
sort 比较器	`[](auto a, auto b){return a<=b;}`	严格弱序只写 < 或 >
例 0 / 例 1	未排序或起点写错，lower/upper 混用	先 sort；1-based 从 `a.begin()+1` 查
例 2 (set)	`lower_bound(s.begin(), s.end(), x)`	成员函数 `s.lower_bound(x)` 配哨兵
multiset 例 3 / 例 4	想删 1 个却写 `ms.erase(x)` 空堆直接 `top()` 或 `mp[k] > 0` 判存在	改用 `ms.erase(ms.find(x))` 单删 先判 `!pq.empty()`；只读用 `mp.count(k)`

本页要点

STL 程序 90% 的报错集中在“无符号 `size()`”“迭代器边界”“比较器写 <=”这三处。

看懂评测结果：按 CE / WA / RE / TLE 快速定位

- **CE**（编译错误）：检查是否漏写头文件、小根堆是否漏了三参数中的 `vector<T>`、结构化绑定是否开启了 `-std=c++17`。
- **RE**（运行错误）：检查空容器调用 `top()/pop()/back()`、无哨兵调用 `prev(s.begin())`、`vector` 下标越界、或比较器写了 `<=`。
- **TLE**（超时）：检查是否对 `set/map` 误用了全局 `lower_bound`、是否用 `ms.count(x)` 统计高频值、大数据量是否忘了关同步与用 `'\n'`。
- **WA**（答案错误）：检查 `lower_bound` 与 `upper_bound` 是否写反、`discrete` 去重后是否漏了 `erase`、`priority_queue` 比较方向是否写反。

本页要点

遇到报错先看评测结论标签，再按对应清单逐行回查，不要盲目乱改。

提交前检查清单（六必查）

1. 用 `lower_bound` / `upper_bound` 之前，数组是否已经保证非降序？
2. 是否分清了“第一个 $\geq$ / $>$”和“最后一个 $\leq$ / $<$（往左退一格）”？
3. 用 `set` / `multiset` 查前驱后继时，是否加了正负哨兵或判了 `!= begin()`？
4. `priority_queue` 默认是大根堆，自定义 `operator<` 方向是否与 `sort` 相反？
5. 当 $n, q \geq 10^5$ 时，是否写了 `ios::sync_with_stdio(false); cin.tie(nullptr);` 与 `'\n'`？
6. 涉及和、坐标或乘积超过 2×10^9 的变量，是否都换成了 `long long`？

最后一步本地自测

先手算验证题目样例，再自己造三组边界数据（$n = 1$、全部相等、全小于或全大于目标值）测通过再提交！

本章全部练习与完成标准

1. 数组二分与离散化：
   - 例 0 | 查找第一个不小于（见 题目/补充题目/二分/查找第一个不小于/problem.pdf，源码 code/stl_ex0_first_ge.cpp）；
   - 例 1 | 离散化与区间频次统计（code/stl_ex1_bounds.cpp）；扩展练习：查找最接近的元素（题目/基础算法/第七章分治算法/查找最接近的元素/problem.pdf）。
2. 有序集合前驱后继：
   - 例 2 | 动态最近邻查询（code/stl_ex2_set_neighbor.cpp）；经典进阶：营业额统计（题目/算法提高篇/四、数据结构（提高篇）/营业额统计/problem.pdf）。
3. 堆与键值映射：
   - 例 3 | 对顶堆动态中位数（code/stl_ex3_median_heap.cpp）；例 4 | 词频多关键字排序（code/stl_ex4_map_sort.cpp）；经典进阶：合并果子（题目/数据结构基础/第三章树/合并果子 (fruit)/problem.pdf）。

完成标准

不看讲义能默写二分四问与 set 哨兵前驱后继；独立编译跑通全部 5 份 code/*.cpp 并自行构造 3 组边界样例验证无误。

课后安排、下一篇预告与本章一句话

- 课后复习：在空白编辑器里不看笔记默写“离散化三行”和“set 哨兵前驱后继 6 行代码”。
- 刻意试错：把 code/ 下 5 份程序里的 <=、erase、全局 lower_bound 故意改错一次，亲眼观察 CE / WA / RE / TLE 报错现象。
- 下一篇预告：学完《C++ 编程语言与 STL》全部 8 章后，正式进入《02_ 基础算法与思维》第 1 章——系统学习时间复杂度估算、一维/二维前缀和、差分与双指针！

本章一句话

能用 STL 在 $O(\log n)$ 搞定的查找与动态维护，绝不手写 $O(n^2)$ 暴力循环。

谢谢！