分治算法

分而治之：对半切开独立求解，让复杂度成对数坍塌

HGNU-ACM 实验室算法组

配套：分治三步与复杂度树 + 快速幂 + 归并延伸 + 倍增与快速选择 + 3 道真题

本章目录 | 前半

分治三步曲与主定理复杂度树

二分查找与归并分治延伸

快速幂：指数对半与二进制倍增

倍增构造与快速选择 Top-K

承接第 6 章《递推与递归》：从“拆出互不重叠的子块”出发，掌握主定理复杂度树、归并延伸、快速幂与快速选择。

本章目录｜后半

三大分治一本通真题精讲

常见错误、检查清单与练习安排

精讲《取余运算（mod）》《循环比赛日程表》《输出前 k 大的数》，并预告下一篇《搜索专题》。

承接第 6 章：普通递归 vs 分治算法到底差在哪？

回顾第 6 章《递推与递归》

在上一章，我们发现直接递归求 `fib(50)` 会超时，因为 `f(5)` 拆出 `f(4)` 和 `f(3)` 时，左右两边有大量重叠的子问题，所以必须靠 `memo` 缓存。

- 如果我们换一种拆分方式：把原问题严格切成互不重叠的两半（左半归左半、右半归右半，绝不重复计算同一块），情况会怎样？
- 比如把 10^5 个数对半劈成前 5×10^4 个和后 5×10^4 个，或者把次幂 2^{1000} 写成 $2^{500} \times 2^{500}$（2^{500} 只算一次！）——
- 这种“切开互不重叠子问题，各自解决后再拼合”的思想，就叫做分治（**Divide and Conquer**）！

本页要点

普通递归常出现子问题重叠；分治拆出的是互不重叠的独立子块，且通常每次直接砍半！

生活小例子与新术语大白话：分解、解决、合并

生活引入：两人合作理顺一副扑克牌

给你 52 张打乱的牌，一个人从头插到底要对比上千次；如果切两半，你和同桌各理好自己手里的 26 张，最后两人比着牌面大小归并成一副，速度会快得多！

分治算法标准三步曲（记住这三个词的大白话含义）：

1. 分解（**Divide**）：把当前大问题切分成若干个规模更小、结构完全相同、且互不重叠的子问题（通常对半切）。
2. 解决（**Conquer**）：如果子问题已经小到一眼能答（到达边界），直接给出答案；否则递归交给“小一号自己”解决。
3. 合并（**Combine**）：把各个子问题返回的局部答案，加工拼合成原问题的完整答案。

本页要点

任何分治代码都只有三块：切半（分解）$\rightarrow$ 递归（解决）$\rightarrow$ 拼合（合并）。

为什么不用暴力？两个“一眼超时”的规模对比

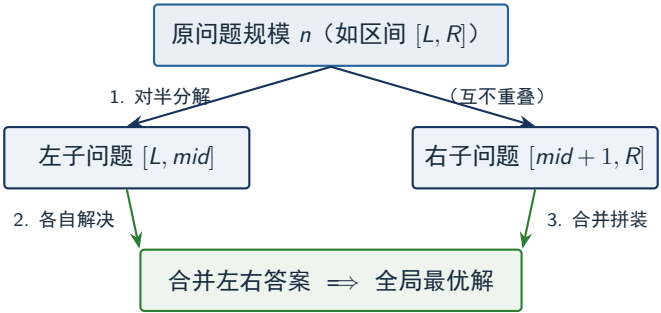
在算法题里，暴力一步步做和分治对半砍，速度差距是“几小时 vs 一瞬间”：

具体任务	暴力一步步做（会 **TLE**）	分治对半切开（秒过 **AC**）
算 $2^{10^9} \bmod 10007$	连乘 10^9 次 ≈ 10 秒（必超时）	每次指数砍半，仅需 $\log_2(10^9) \approx$ **30** 步！
给 10^5 个数排序 / 逆序对	双层循环对比 $\approx 5 \times 10^9$ 次（50 秒）	对半归并，共 17 层，约 1.7×10^6 次（10 毫秒）
从 10^5 个数里找第 k 大	全排序要 $O(n \log n)$ 步	每次扔掉一半，仅约 2×10^5 步（$O(n)$）

本页要点

当数据规模达到 10^5 甚至 10^9 时，只有把规模“每次砍半”的对数级思想才能稳过 1 秒时限！

TikZ 图解：分治三步曲的通用执行骨架



- 必须确认一件事：左半部分 $[L, mid]$ 和右半部分 $[mid+1, R]$ 完全不交集；跨中点的联系留在第 3 步“合并”里统一处理。

本页要点

画好分治图先看中间分界线 `mid`：左半、右半、跨过 `mid` 的合并，三部分盖住全部情况。

TikZ 图解：主定理直觉——三种分治递归树怎么算复杂度？

不用背复杂的数学公式，画出“每一层干了多少活”，把每层加起来就是总时间！

1. 单分支减半（仅走 1 支）
二分查找 / 快速幂

第 0 层：1 个，代价 1
↓
第 1 层：1 个，代价 1
↓
共 $\log_2 n$ 层 $\Longrightarrow$ $\mathbf{O(\log n)}$

2. 双分支合并（走 2 支）
归并排序 / 逆序对

第 0 层：$1 \times n = n$
↓
第 1 层：$2 \times \frac{n}{2} = n$
↓
共 $\log_2 n$ 层 $\Longrightarrow$ $\mathbf{O(n \log n)}$

3. 单分支减治（丢掉一半）
快速选择 Top-K

第 0 层：代价 n
↓
第 1 层：代价 $n/2$
↓
$n + \frac{n}{2} + \frac{n}{4} + \cdots < 2n$ $\Longrightarrow$ $\mathbf{O(n)}$

本页要点

记牢“主定理直觉”三张树： 每层只走 1 支花 $O(1)$ 是 $O(\log n)$； 拆 2 支合并花 $O(n)$ 是 $O(n \log n)$； 丢掉一半花 $O(n)$ 是 $O(n)$！

手算跟踪表：规模 $n = 16$ 时三种分治树的每层操作数

设原规模 $n = 16$，列出从第 0 层一路砍半直到规模 $n = 1$（终止条件）的精确工作量：

层数	子问题规模	单支快速幂 $O(1)$/层	双支归并 $O(n)$/层	单支减治 Top-K
第 0 层	16	1	$1 \times 16 = 16$	16
第 1 层	8	1	$2 \times 8 = 16$	8
第 2 层	4	1	$4 \times 4 = 16$	4
第 3 层	2	1	$8 \times 2 = 16$	2
第 4 层	1（终止）	1（返回）	$16 \times 1 = 16$	1（返回）
合计	5 层（$\log_2 16 + 1$）	共 5 次	共 80 次（$O(n \log n)$）	共 31 次（$< 2n$）

本页要点

看第 4 行：一旦当前规模缩小到 1 就立刻触底返回，再也不往下拆分！

正确性论证：怎么证明一个分治写法一定对？

和递归一样，分治靠结构归纳法三步确认正确性：

检验阶段	在分治中具体检查什么
1. 最小边界正确	当规模 $n = 1$（如 `L == R`）或 $p = 0$ 时，不继续拆分，直接返回的值是否百分百准确？
2. 归纳假设成立	既然 `mid` 严格在 $[L, R)$ 内，左右子区间规模都严格小于原区间，假定两次递归返回的结果已经准确。
3. 合并无漏无重	合并步是否把“左内部 + 右内部 + 跨过中点 `mid` 的情况”全部精准拼入最终结果？

本页要点

分治最怕忘掉“跨中点的那部分贡献”——左半对、右半对，合起来却漏了横跨左右的情况就会 **WA**！

模板代码：通用双分支分治递归骨架

```
void divide_conquer(int L, int R) {
    if (L >= R) return;                          // 1. 终止边界：只剩 0 或 1 个元素
    int mid = L + (R - L) / 2;                   // 2. 分解：取中点（防 L+R 溢出）
    divide_conquer(L, mid);                      // 3. 解决左半区间 [L, mid]
    divide_conquer(mid + 1, R);                  // 4. 解决右半区间 [mid+1, R]
    combine(L, mid, R);                          // 5. 合并：处理跨 mid 的联系
}
```

本页要点

为什么这样写： L+(R-L)/2 防溢出； 当 $L = 1, R = 2$ 时分成 $[1, 1]$ 和 $[2, 2]$ 绝不死循环；若写 $[L, mid - 1]$ 和 $[mid, R]$ 就会无限递归爆栈！

上机验证与常见分治退化陷阱

三组边界核对（用 L, R 代入上页模板）

- `L = 1, R = 1`：命中 `L >= R` 返回；`L = 1, R = 2`：分成 `[1,1],[2,2]`；`L = 1, R = 5`：分成 `[1,3],[4,5]`。

反事实追问：为什么有时写了分治还是 TLE？

问：假如求一模一样的子问题时写了两次 `solve(n/2) + solve(n/2)`（没存变量），复杂度会变多少？
答：节点数从 $\log_2 n$ 暴涨成 $2n-1$，原本 $O(\log n)$ 瞬间退化成 $O(n)$，遇 $n = 10^9$ 直接 **TLE**！

本页要点

分治绝不允许“对完全相同的子问题递归调用两次”，算好一次务必存到局部变量！

从二分到归并：单支减半 vs 双支合并

- 承接第 **4** 章《二分答案》：二分查找其实是最简单的减治分治——每次算好中点 `mid`，判断目标在左半还是右半，另一半直接扔掉，合并步为空，总复杂度 $O(\log n)$。
- 但有很多问题不能扔掉另一半，比如：求一个排列的逆序对总数（一本通 1237）。
- 生活场景：体育课按身高从矮到高排了一排队伍 `[2, 6, 3, 4, 5, 1]`，如果有个子高的站在了矮个子前面（即 $i < j$ 但 $a[i] > a[j]$），就算一对“逆序”。怎么快速数出一共有多少对？

本页要点

二分每次丢掉一半所以只要 $O(\log n)$；归并两边都要处理并在合并时顺手统计跨区间的答案！

暴力枚举为什么超时？

最直白的暴力做法

写两层 `for` 循环枚举每一对下标 (i, j)（$1 \le i < j \le n$），只要发现 $a[i] > a[j]$ 就让计数器 `ans++`。

- $n = 6$ 时只要比较 $\frac{6\times5}{2} = 15$ 次；
- 但一本通 1237 中 $n \le 10^5$！此时需要比较 $\frac{10^5\times(10^5-1)}{2} \approx \mathbf{5 \times 10^9}$ 次！
- 在 1 秒限制（约 10^8 次操作）下，暴力双循环要跑 50 秒，必定 **TLE**。
- 另外注意：最坏全逆序时，答案达到 5×10^9，已经超过 `int` 上限，计数器必须用 `long long`！

本页要点

暴力 $O(n^2)$ 在 $n = 10^5$ 必超时；而且逆序对总数最大约 5×10^9，必须开 `long long`！

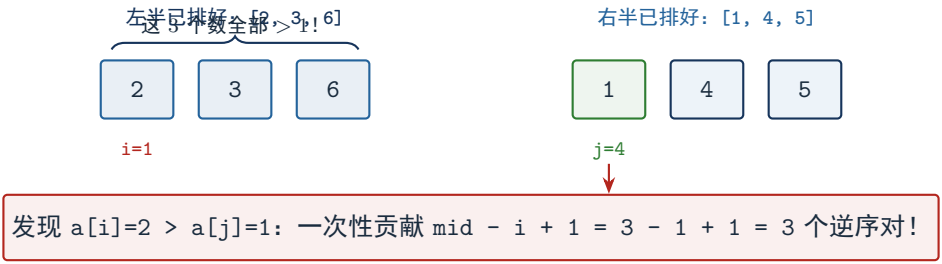
新术语大白话：归并排序与“批量数逆序对”

- 归并排序（**Merge Sort**）：先把数组对半切成左半和右半，分别排成从小到大有序，再用两个指针把两段有序数组合并成一整段有序数组。
- 为什么有序就能批量数逆序对？
  1. 左半内部全部已经从左到右升序，右半内部也已经升序；
  2. 合并时如果发现右半当前元素 `a[j]` 严格小于左半当前元素 `a[i]`（即 `a[i] > a[j]`）；
  3. 因为左半排好序了，所以左半从 `i` 一直到 `mid` 的所有数全都比 `a[j]` 大！
  4. 于是不用一个个去比，一次性直接加上 `mid - i + 1` 个逆序对！

本页要点

“两半分别有序”让我们不用一对一对比，发现 `a[i] > a[j]` 时一下子批量收割 `mid - i + 1` 对！

TikZ 图解：合并 [2，3，6] 与 [1，4，5] 统计跨界逆序对



本页要点

每当把右半的数 a[j] 放入临时数组，就说明左半还没放进去的所有数都在它前面且比它大！

手算跟踪表：合并 [2，3，6] 与 [1，4，5] 的双指针过程

设左半下标 $1 \sim 3$（mid=3），右半下标 $4 \sim 6$，初始跨界逆序对 cross = 0：

轮次	$i \le 3, j \le 6$	比较	选入	新增逆序	说明
1	$i=1, j=4$（真）	$2>1$	右 1	$+3$	$2,3,6$ 全比 1 大，$j \to 5$
2	$i=1, j=5$（真）	$2 \le 4$	左 2	$+0$	选左边不产生新逆序，$i \to 2$
3	$i=2, j=5$（真）	$3 \le 4$	左 3	$+0$	$i \to 3$
4	$i=3, j=5$（真）	$6>4$	右 4	$+1$	6 比 4 大，$j \to 6$
5	$i=3, j=6$（真）	$6>5$	右 5	$+1$	6 比 5 大，$j \to 7$
6	$j=7$（为假）	退出	收尾 6	$+0$	抄回得 [1..6]，跨界共 5 对

本页要点

再加上左半内部 1 对 $(6,3)$ 和右半内部 2 对 $(4,1),(5,1)$，合计 $1+2+5=8$ 对！

正确性论证：为什么“边排边数”不重不漏？

论证部分	为什么绝不会算重或算漏
1. 划分完备	任意一对下标 (p, q)（$p < q$）只可能属于三种情况之一：都在左半、都在右半、p 在左半且 q 在右半。三类完全互斥。
2. 排序不改跨界关系	左右半各自内部排序后，虽然左半内部打乱了原本先后，但左半每个数原本的下标都严格小于右半每个数！
3. 右出列即结算	右半元素 $a[j]$ 在被放进有序数组那一刻，左半尚未出列的恰好是所有满足 $a[i] > a[j]$ 的左半数，刚好被统计一次且仅一次。

本页要点

尽管左右内部排了序，但“所有左半元素原本都在所有右半元素左边”这一先后关系从未破坏！

模板代码：归并分治统计逆序对 (1/2)

```
long long merge_count(vector<int>& a, vector<int>& tmp, int L, int R) {
    if (L >= R) return 0;
    int mid = L + (R - L) / 2;
    long long ans = merge_count(a, tmp, L, mid)
                  + merge_count(a, tmp, mid + 1, R);
    int i = L, j = mid + 1, k = L;
    while (i <= mid && j <= R) {
        if (a[i] <= a[j]) tmp[k++] = a[i++];  // 相等先放左，不算逆序
        else { ans += mid - i + 1; tmp[k++] = a[j++]; } // 批量计入逆序
    }
```

本页要点

发现 a[i] > a[j] 时左半剩余 $[i..mid]$ 全比 a[j] 大，直接加 mid - i + 1！

模板代码：归并分治统计逆序对 (2/2)

```
    while (i <= mid) tmp[k++] = a[i++];        // 收尾左半剩余元素
    while (j <= R) tmp[k++] = a[j++];          // 收尾右半剩余元素
    for (int p = L; p <= R; ++p) a[p] = tmp[p]; // 写回原数组供上层使用
    return ans;
}
```

本页要点

临时数组 `tmp` 在主函数开好传引用，切忌在递归内新建数组导致超时或爆内存！

上机验证与反事实追问（等号陷阱！）

三组测试数据（输入序列 → 逆序对数）

- 升序边界：[1, 2, 3, 4] → 0；　样例：[2, 6, 3, 4, 5, 1] → 8；　含重：[3, 3, 2] → 2。

反事实追问

问：如果把 `if (a[i] <= a[j])` 改成没有等号的 `if (a[i] < a[j])`，面对 [3, 3, 2] 会出什么错？
答：左边 3 遇到右边相等的 3 时会错误走进 `else` 分支，把相等元素当成逆序加上去，导致重复数据全部 **WA**！

本页要点

严格大于才算逆序，所以 `a[i] == a[j]` 时务必优先取走左边 `a[i] <= a[j]`！

生活引入：手算 $3^{13} \bmod 100$，真的要连乘 12 次吗？

偷懒妙招

假设让你口算 3^{13} 的末两位（即 mod100）：
你不必从 3^1 乘到 3^{13}，只要先算出一半 $3^6 \equiv 29 \pmod{100}$，再用 $3^{13} = (3^6)^2 \times 3$ 就行了！
而算 3^6 时，又只要先算 $3^3 = 27$，再算 $27^2 = 729 \equiv 29$！

- 指数 $13 \to 6 \to 3 \to 1 \to 0$，一共只用了 4 步对半拆分，就把 13 次幂算完了！
- 若指数换成 10^9，每次对半砍，只需要 $\log_2(10^9) \approx$ **30** 步！

本页要点

对半拆指数公式：$b^p = (b^{\lfloor p/2 \rfloor})^2$（$p$ 为偶数），或 $(b^{\lfloor p/2 \rfloor})^2 \times b$（$p$ 为奇数）。

暴力连乘为什么会超时？

最直白的循环做法

```
long long ans = 1; for (int i = 1; i <= p; ++i) ans = ans * b % k;
```

- 当 $p \le 2^{31}-1 \approx 2.14 \times 10^9$ 时，这个循环要实打实运行 21 亿次！
- 评测机 1 秒只能执行约 10^8 次基本运算，连乘循环会跑超过 20 秒，必定 **TLE**。
- 而且还有一个隐藏坑：如果 k 接近 $2^{31}-1$ 甚至更大，两个取模后的数相乘可能会突破 `int` 甚至逼近 `long long` 上限！

本页要点

解决 $b^p \bmod k$ 的两把钥匙： 指数对半把 $O(p)$ 压成 $O(\log p)$； 乘积强转防溢出。

新术语大白话：同余乘法性质与分治快速幂

- 同余乘法性质：求积后再取余，等价于每做一次乘法就先取余再继续乘：

$$(a \times b) \bmod k = \big((a \bmod k) \times (b \bmod k)\big) \bmod k.$$

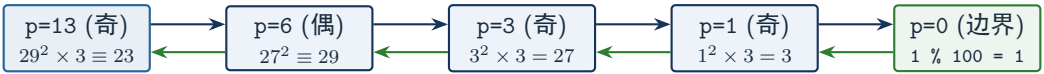
- 这样一来，中间结果永远被压在 $[0, k-1]$ 以内，绝不会像 3^{1000} 那样膨胀成天文数字。
- 快速幂（**Binary Exponentiation**）：利用指数二进制折半，结合同余取模，在 $O(\log p)$ 时间内求出 $b^p \bmod k$。

本页要点

记住：取余只能写在乘法/加法外面，千万不能“边算边对指数 p 取余 k”！

TikZ 图解：分治递归 qpow(3，13，100) 的折半与合并

向下分解：每次 p / 2，只调用 1 支！



向上合并：算一次 half*half，奇数补乘 3

本页要点

每层只递归一次 qpow(b，p/2，k) 并将结果存在 half 里，合并时只需做 half * half！

手算跟踪表：qpow(3，13，100）的完整返回链

下表按“从触底到返回主程序”的真实结算顺序列出每层数据：

当前调用	$p == 0$?	下层返回的 half	本层合并算式（mod100）	本层返回值
qpow(3, 0)	真（触底）	无	$1 \bmod 100$	**1**
qpow(3, 1)	假（奇数）	1	$(1 \times 1) \times 3 = 3$	**3**
qpow(3, 3)	假（奇数）	3	$(3 \times 3) \times 3 = 27$	**27**
qpow(3, 6)	假（偶数）	27	$27 \times 27 = 729 \equiv 29$	**29**
qpow(3, 13)	假（奇数）	29	$(29 \times 29) \times 3 = 2523 \equiv 23$	**23**（结束）

本页要点

第 1 行的 `1 % k` 千万别写成 `1`：如果输入 $k = 1$，任何数对 1 取余都必须是 0！

正确性论证与大模数乘法防溢出

检验项	为什么这样写绝对安全
基础边界	当 $p=0$ 时数学规定 $b^0=1$，对 k 取模即为 `1 % k`（兼容 $k=1\to 0$）。
归纳合并	若 `half` $=b^{\lfloor p/2\rfloor} \bmod k$，则偶数 $p=2m$ 时 $b^p=(b^m)^2$；奇数 $p=2m+1$ 时 $b^p=(b^m)^2\cdot b$。
防乘溢	当 k 超过 3×10^9（如 10^{12}）时，两个小于 k 的 `long long` 相乘仍会超过 9.2×10^{18}！强转成 128 位整数 `(__int128)a * b % mod` 可彻底根治溢出。

本页要点

口诀：底数先取模防首乘溢出，边界写 `1 % k` 防 $k=1$，乘法用 `__int128` 防爆 `long long`。

模板代码：分治递归快速幂

```
long long mul_mod(long long a, long long b, long long mod) {
    return (long long)((__int128)a * b % mod); // 128 位防两数相乘爆 long long
}
long long qpow(long long b, long long p, long long k) {
    if (p == 0) return 1 % k;                  // 边界：特判 k=1 时返回 0
    long long half = qpow(b, p / 2, k);        // 只递归算一次半次幂！
    long long res = mul_mod(half, half, k);    // 平方合并
    if (p & 1) res = mul_mod(res, b % k, k);   // 奇数补乘一个底数 b
    return res;
}
```

本页要点

为什么这样写： 第 6 行把子结果存进 half 避免二次递归； b % k 先缩小底数，p & 1 判奇偶。

拓展对比：迭代二进制快速幂

除了递归版，把指数 p 看成二进制（如 $13 = 8 + 4 + 1$），也能用循环省掉系统栈：

```
long long qpow_iter(long long b, long long p, long long k) {
    long long res = 1 % k;
    b = (b % k + k) % k;                    // 底数先取模且防负数
    while (p > 0) {
        if (p & 1) res = mul_mod(res, b, k);   // 二进制当前位是 1 就乘进答案
        b = mul_mod(b, b, k);                  // 底数自身倍增：b -> b^2 -> b^4 ...
        p >>= 1;                               // 指数右移一位（除以 2）
    }
    return res;
}
```

本页要点

递归版直观体现“分治合并”，迭代版零栈开销、常数更小；两种都要能默写！

上机验证与反事实追问（重复调用退化陷阱）

三组测试数据（输入 b p k → 预期输出）

- 普通样例：3 13 100 → 23；　　最小边界：2 0 1 → 0（验证 1 % k）；
- 大数防溢：123456789 987654321 1000000007 → 652541198（一瞬间秒出）。

反事实追问

问：假如偷懒不写 half 变量，直接写成 return mul_mod(qpow(b, p/2, k), qpow(b, p/2, k), k); 会怎样？
答：每层分裂成两个完全相同的调用！深度 $\log_2 p$ 层的二叉树共有 $2^{\lfloor \log_2 p \rfloor + 1} - 1 \approx 2p$ 个节点，时间复杂度瞬间从 $O(\log p)$ 退化成 $O(p)$，当 $p = 10^9$ 时直接 **TLE**！

本页要点

写两遍 qpow(b, p/2, k) 是快速幂最高频的 **TLE** 错法！

引入：两种特别的分治——倍增拼接与减治丢半

前面学的快速幂和归并是“从大往下拆（自顶向下）”，本节看另外两种在竞赛里极好用的分治姿势：

1. 倍增构造分治（自底向上翻倍）：比如循环比赛日程表，先填好 1×1 小表，再翻倍成 $2 \times 2 \to 4 \times 4 \to \cdots \to 2^m \times 2^m$。
2. 快速选择减治（只进一侧丢弃另一半）：比如输出前 k 大的数，每次用枢轴切成两半后，只走进包含第 k 大的那一侧，另一侧彻底不管，均摊复杂度只要 $O(n)$！

本页要点

分治不只是“先递归拆再合并”，按 2 的幂次翻倍拼表、只挑半边走也是纯粹的分治思想！

倍增构造原理：怎么用 $k \times k$ 小赛程拼出 $2k \times 2k$ 大赛程？

假设有 $2k$ 名选手（前 k 人编号 $1 \sim k$，后 k 人编号 $k+1 \sim 2k$），前 k 人内部赛程（左上 $k \times k$ 块）已经填好：

- 左下象限（后 k 人前 $k-1$ 天内战）：直接把左上象限每个人编号全部 $+k$！
- 右上象限（前 k 人后 k 天去打后 k 人）：直接把左下象限（即左上 $+k$）抄过来！
- 右下象限（后 k 人后 k 天去打前 k 人）：把左上象限原封不动抄过来！

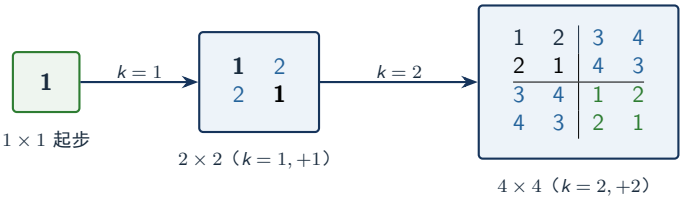
为什么一定满足“每天不撞车、每人不重样”？

左上每行每列是 $\{1 \dots k\}$ 的排列，加 k 后是 $\{k+1 \dots 2k\}$ 的排列；左右拼接后，每行每列恰好凑齐 $1 \sim 2k$ 全排列，绝无冲突！

本页要点

四象限密码：左上 $=$ 右下，左下 $=$ 右上 $=$ 左上全员 $+k$！

TikZ 图解：从 $1\times 1 \to 2\times 2 \to 4\times 4$ 的赛程倍增演变



本页要点

看 4×4 表的横竖分割线：蓝色块全部是左上角 $+2$，右下角绿字块完全复制左上角！

生活引入：只找年级前 5 名，干嘛把全校 1000 人全排序？

日常对比

老师想知道全校 1000 份试卷里最高的那 5 个分数。
做法 A：把 1000 份试卷按分数从高到低彻底排好序，要花 $O(n \log n)$ 时间；
做法 B：随便抽一份比如 85 分当分界线，把 > 85 分的扔左边、≤ 85 分的扔右边。
如果数一下左边刚好已经有了 12 份试卷，右边的 988 份试卷还需要理会吗？直接扔掉！

- 下一步只需要在左边那 12 份里继续找前 5 名就行，范围一下子缩水了将近 99%！

本页要点

“划分之后只进包含目标的一侧，另一侧彻底丢掉”，就是快速选择从 $O(n \log n)$ 降到 $O(n)$ 的秘诀！

新术语大白话：枢轴划分、减治与均摊 $O(n)$

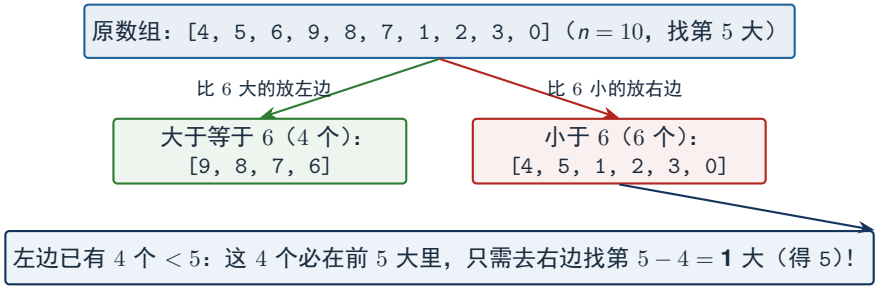
- 枢轴划分（**Partition**）：在当前区间 $[L, R]$ 内选一个参考数 `pivot`（枢轴），用双指针把大于（或小于）它的数挪到一边，其余挪到另一边，返回分界位置。
- 减治（**Prune and Search**）：与“两边都要递归”的归并不同，每次做完划分就根据左边元素个数与 k 的大小关系，剪掉半边不递归。
- 为什么平均复杂度是 $O(n)$？第 1 轮扫 n 个数，第 2 轮约扫 $n/2$ 个，第 3 轮约扫 $n/4$ 个……总扫描量：

$$n + \frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \cdots < \mathbf{2n} \implies \text{均摊} O(n)!$$

本页要点

归并因为两半都要算所以每层都是 n（共 $O(n \log n)$）；快速选择只走一半，等比级数求和小于 $2n$！

TikZ 图解：在 [4,5,6,9,8,7,1,2,3,0] 中找前 5 大（选 pivot = 6）



本页要点

若左边个数 $\geq k$，直接丢掉右边只搜左边；若左边个数 $< k$，左边全收下，去右边找第 $k -$ 左个数 大！

手算跟踪表：快速选择找第 $k = 5$ 大的区间收缩过程

在数组 [4,5,6,9,8,7,1,2,3,0]（下标 $0 \sim 9$）中找第 5 大（即降序下标 4）：

轮	区间 $[L, R]$	枢轴左段大小	比较结果	下一步动作
1	$[0,9]$（10 个）	选 6，≥ 6 有 4 个	$4 < 5$	左保留，去 $[4,9]$ 找第 1 大
2	$[4,9]$（6 个）	选 4，≥ 4 有 2 个	$2 \geq 1$	扔右半！进左段 $[4,5]$
3	$[4,5]$（$\{5,4\}$）	选 5，≥ 5 有 1 个	$1 \geq 1$	缩到单元素 $[4,4]$
4	$[4,4]$（$L == R$）	无需再分	触发终止	第 5 大为 **5**，前 5 大 $\{9..5\}$

本页要点

区间长度从 $10 \to 6 \to 2 \to 1$ 快速坍塌，根本不需要对剩下的数字做任何排序！

模板代码：C++ STL nth_element 找前 k 大

C++ 标准库 <algorithm> 内置了基于快速选择的 nth_element，平均复杂度正是 $O(n)$：

```
// 目标：把数组 a 中前 k 大的元素放到 [0, k) 位置，并按降序输出
nth_element(a.begin(), a.begin() + k, a.end(), greater<long long>()); // O(n) 划分
sort(a.begin(), a.begin() + k, greater<long long>());                 // O(k log k)
    排好
for (int i = 0; i < k; ++i) cout << a[i] << '\n';
```

本页要点

为什么这样写： 第 4 参传 greater<>() 挑最大（默认挑最小）； nth_element 后 [0,k) 内部乱序，需再局部 sort！

拓展代码：手写快速选择 quick_select 原理

```cpp
long long quick_select(vector<long long>& a, int L, int R, int k) {
    if (L == R) return a[L];                    // 区间缩成 1 个数直接返回
    long long pivot = a[L + (R - L) / 2];       // 选中间值作枢轴防退化
    int i = L - 1, j = R + 1;
    while (i < j) {
        do ++i; while (a[i] > pivot);           // 大的放左边
        do --j; while (a[j] < pivot);           // 小的放右边
        if (i < j) swap(a[i], a[j]);
    }
    if (k <= j) return quick_select(a, L, j, k); // 只进左半
    return quick_select(a, j + 1, R, k);         // 只进右半
}
```

本页要点

不要写死选 a[L] 当枢轴：遇有序输入会退化成 $O(n^2)$ **TLE**！

上机验证与反事实追问（重复元素与排序误区）

三组测试数据（测试 nth_element + 局部 sort）

- 普通样例：n=10，a=[4..0]，k=5 → 9，8，7，6，5；
- 含重复值：n=5，a=[5,1,5,3,5]，k=3 → 5，5，5；　最小边界：[-10,-20]，k=1 → -10。

反事实追问

问：如果用了 nth_element 后偷懒不写后面的 sort 直接打印 a[0..k-1]，会出什么问题？

答：nth_element 只保证前 k 个确实是最大的那 k 个，但它们内部顺序可能是乱的（如 7 9 8 6 5），直接输出必 **WA**！

本页要点

nth_element 不会去重，也不会把前 k 个排好序；需要有序输出一定要再 sort 前 k 个。

例 1 | 取余运算（mod）（一本通 1326）[基础]：先读懂要求

大白话题意

给定三个整数 b, p, k，计算 $b^p \bmod k$ 的值，并按指定格式输出整个算式和结果。

- 输入：一行三个整数 b, p, k，满足 $0 \le b, p \le 2^{31}-1$，$1 \le k \le 2^{31}-1$。
- 输出：一行字符串，严格按照 `b^p mod k=ans` 格式（注意 `mod` 前后各有一个空格，`=` 前后没有空格）。
- 特别说明：当 $p = 0$ 时规定 $b^0 = 1$；当 $k = 1$ 时任何数对 1 取余都是 0。

原题样例

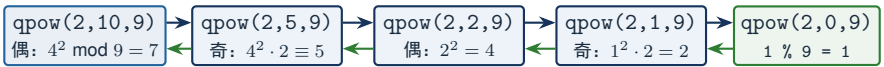
输入：`2 10 9` $\Longrightarrow$ 输出：`2^10 mod 9=7`。

本页要点

别忽略输入输出格式中的空格位置，且 $p = 0, k = 1$ 是原题专门强调的边界陷阱！

例 1 | 怎么想到的？范围分析与分治调用图解

- p 最大达 $2^{31}-1 \approx 2.14 \times 10^9$，暴力乘 p 次绝对 **TLE**；必须用快速幂分治在 31 步内解决。
- 输出还需要打印原本的 `b`，所以先另存 `base = (b % k + k) % k` 传给 `qpow`，不要覆盖 `b`！



本页要点

即使 b, k 均达到 $2^{31}-1$，在 `mul_mod` 里强转 `__int128` 也能彻底防止乘法溢出。

例 1 | 原题样例 2 10 9 手推表

从触底返回一步步核对样例 2^10 mod 9=7：

层数状态	p 奇偶	收到 half	计算步骤（mod9）	返回值
qpow(2, 0, 9)	边界 $p=0$	无	$1 \bmod 9$	1
qpow(2, 1, 9)	奇数	1	$1 \times 1 \times 2 = 2$	2
qpow(2, 2, 9)	偶数	2	$2 \times 2 = 4$	4
qpow(2, 5, 9)	奇数	4	$(4 \times 4) \times 2 = 32 \equiv 5$	5
qpow(2, 10, 9)	偶数	5	$5 \times 5 = 25 \equiv 7$	**7**（与样例一致）

本页要点

仅经过 5 步算完 $2^{10} \bmod 9 = 7$，输出时按原值拼出 2^10 mod 9=7。

例 1 | 核心步骤与相对快速幂模板的改动

按顺序执行	纸上得到	核对问题
1. 用 long long 读入 b, p, k	防止 $2^{31}-1$ 溢出 int	不要直接修改变量 b 的原值！
2. 算 base=(b%k+k)%k	把底数安全压到 $[0, k-1]$	保留原 b 用于最后打印算式
3. 调 qpow(base,p,k)	用 __int128 算得结果	边界写了 return 1 % k; 吗？
4. 按格式打印等式	b^p mod k=ans	mod 左右有空格，= 无空格吗？

本页要点

相对模板改动：核心 mul_mod 与 qpow 完全一致，主程序另开 base 传参并保留原始 b 用于格式输出。

例 1 | 完整可编译代码 (1/2)：防溢出乘法与快速幂函数

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

long long mul_mod(long long a, long long b, long long mod) {
    return (long long)((__int128)a * b % mod); // 防止中间相乘爆 long long
}
long long qpow(long long b, long long p, long long k) {
    if (p == 0) return 1 % k;                  // 边界特判：k=1 时 1%1=0
    long long half = qpow(b, p / 2, k);        // 折半递归一次并缓存
    long long res = mul_mod(half, half, k);    // 平方合并
    if (p & 1) res = mul_mod(res, b % k, k);   // 奇数补乘底数
    return res;
}
```

本页要点

这段 mul_mod + qpow 是万能工具函数，后续数论、矩阵快速幂均可直接复用。

例 1 | 完整可编译代码 (2/2)：主程序与格式输出（code/mod_pow.cpp）

```cpp
int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    long long b, p, k;
    if (cin >> b >> p >> k) {
        long long base = (b % k + k) % k;      // 保证底数非负且先缩小至 < k
        cout << b << '^' << p << " mod " << k << '=' << qpow(base, p, k) << '\n';
    }
    return 0;
}
```

本页要点

注意格式空格：b^p mod k=ans（mod 前后各一空格，= 无空格）；用 base 传参保留原 b！

例 1 | 易错点与 3 组自测边界

常见易错

- 易错 **1**（特判挂 **WA**）：`p==0` 写成 `return 1;`，输入 `2 0 1` 错误输出 `=1`（应为 `=0`）；
- 易错 **2**（调用两次 **TLE**）：没存 `half` 直接写 `qpow(b,p/2,k) * qpow(b,p/2,k)` 必超时；
- 易错 **3**（乘法溢出 **WA**）：$k = 2^{31} - 1$ 时乘积超 4×10^{18}，必须强转 `__int128`。

三组自测数据（输入 $\rightarrow$ 预期输出）

- 原题样例：`2 10 9` $\rightarrow$ `2^10 mod 9=7;`　特殊边界：`2 0 1` $\rightarrow$ `2^0 mod 1=0;`
- 极值防溢：`2147483647 2147483647 2147483646` $\rightarrow$ `...=1`。

本页要点

自测一定跑一次 `2 0 1`，能瞬间抓出所有忘写 `1 % k` 的人！

例 2 | 循环比赛日程表（一本通 1325）[基础]：先读懂要求

大白话题意

有 $N = 2^M$ 名选手（$1 \le M \le 10$），举行 $N-1$ 天循环赛，每名选手每天与其余 $N-1$ 人各打一场且无轮空。输出 $N \times N$ 表：第 0 列写自己编号，第 j 列（$1 \le j < N$）写第 j 天的对手编号。

- 输入：一个整数 M（$1 \le M \le 10$，$N \le 1024$）。　输出：N 行每行 N 个空格隔开的整数。

原题样例

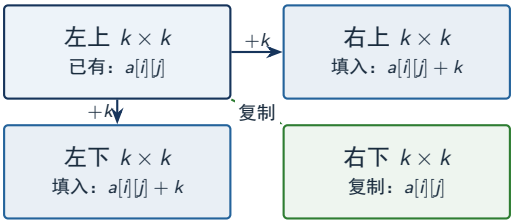
输入：3（$N = 8$） $\Longrightarrow$ 第 1 行 1 2 3 4 5 6 7 8，...，第 8 行 8 7 6 5 4 3 2 1。

本页要点

$M = 10$ 时输出超 10^6 个整数，必须关闭流同步且禁用 `endl`！

例 2 | 怎么想到的？四象限对称填充建模图解

暴力回溯填表在 $N = 1024$ 时会组合爆炸；而观察 2×2 到 4×4 的结构，直接由四个 $k \times k$ 象限拼成：



本页要点

只需要写三行赋值：左下 $= +k$、右上 $= +k$、右下 $=$ 左上，从小到大翻倍 M 轮即可！

例 2 | 原题样例 M=3（8×8）分步倍增手推表

从 1×1 起步，经过 $k=1 \to k=2 \to k=4$ 三轮倍增，第 3 轮直接得到原题样例的 8×8 整表：

轮次	得到的方阵（展示前 4 行或完整表）
起步 1×1	[1]
$k=1 \to 2 \times 2$	第 1 行：1 2 ｜ 第 2 行：2 1
$k=2 \to 4 \times 4$	第 1 行：1 2 3 4；第 2 行：2 1 4 3；第 3 行：3 4 1 2；第 4 行：4 3 2 1
$k=4 \to 8 \times 8$（原题样例）	左上保留上轮 4×4，左下/右上全加 4，右下复制左上：第 1 行 1 2 3 4 5 6 7 8；第 5 行 5 6 7 8 1 2 3 4；第 8 行 8 7 6 5 4 3 2 1

本页要点

任意抽查一列（如第 4 列）：$5,6,7,8,1,2,3,4$ 恰含 $1 \sim 8$ 且两两配对对称，毫无冲突！

例 2 | 核心步骤与相对分治思路的细节

按顺序执行	纸上得到	核对问题
1. 计算 $n = 1 \ll m$	确定正方形边长 2^m	用 0-indexed 开 $n \times n$ 矩阵了吗？
2. 初始化 `a[0][0] = 1`	填好 1×1 底座	注意第 1 个选手编号是 1（不是 0）！
3. 循环 `k = 1; k < n; k <<= 1`	完成 $2, 4, \ldots, n$ 阶填充	内层只扫 $0 \le i, j < k$ 吗？
4. 按行打印空格分隔	输出整张 $n \times n$ 赛程表	用 `'\n'` 不用 `endl` 了吗？

本页要点

总赋值次数约 $\frac{1}{3}n^2$；用 `vector<vector<int>> a(n, vector<int>(n, 0))` 按需分配，绝不数组开小！

例 2 | 完整可编译代码 (1/2)：初始化与规模定义

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

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int m;
    if (!(cin >> m)) return 0;
    int n = 1 << m;                              // 选手总数 N = 2^M
    vector<vector<int>> a(n, vector<int>(n, 0));
    a[0][0] = 1;                                 // 1*1 最小底座
```

本页要点

按 $n = 1 \ll m$ 动态分配 $n \times n$ 二维数组，兼容最大 $M = 10$（1024×1024）绝不越界！

例 2 | 完整可编译代码 (2/2)：四象限倍增与输出（code/schedule_table.cpp）

```
    for (int k = 1; k < n; k <<= 1)           // 规模翻倍：1, 2, 4, ... n/2
        for (int i = 0; i < k; ++i)
            for (int j = 0; j < k; ++j) {
                a[i + k][j] = a[i][j] + k;    // 左下 = 左上 + k
                a[i][j + k] = a[i][j] + k;    // 右上 = 左上 + k
                a[i + k][j + k] = a[i][j];    // 右下 = 复制左上
            }
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            cout << a[i][j] << (j + 1 == n ? '\n' : ' ');
    return 0;
}
```

本页要点

百万输出标配：关同步 + '\n' + 三目运算符控制行末无多余空格防 **PE**。

例 2 | 易错点与 3 组自测边界

常见易错

- 易错 **1**（数组开太小 **RE**）：手开固定数组 `int a[100][100]`，看到 $M \le 10$ 却忘了 $2^{10} = \mathbf{1024}$；
- 易错 **2**（输出超时 **TLE**）：在 $M = 10$ 输出 10^6 个整数时用了 `endl` 刷新缓冲区；
- 易错 **3**（象限对应错 **WA**）：把右下角 `a[i+k][j+k]` 错写成 `a[i][j] + k`，导致后 k 人变成自己跟自己或重复对打。

三组自测数据（输入 M → 预期输出首行与末行）

- 最小边界：`1` → 两行：`1 2` 与 `2 1`；
- 中等自测：`2` → 4×4 矩阵，首行 `1 2 3 4`，末行 `4 3 2 1`；
- 原题样例：`3` → 8×8 矩阵，首行 `1 2 3 4 5 6 7 8`，第 5 行 `5 6 7 8 1 2 3 4`。

本页要点

测一下 $M = 10$ 看程序能不能在 0.2 秒内瞬间跑完不卡顿，确认不会 **TLE**！

例 3 | 输出前 k 大的数（一本通 1235）[入门]：先读懂要求

大白话题意

给定 n 个整数（$2 \le n < 100000$，$|a_i| \le 10^8$），找出其中最大的 k 个数（$1 \le k < n$）并从大到小输出。注意：相同的数按出现次数分别计数，重复的元素必须重复输出！

- 输入：第 1 行是 n；第 2 行是 n 个整数；第 3 行才是 k（注意读入顺序！）。
- 输出：共 k 行，从大到小每行输出一个数。

原题样例

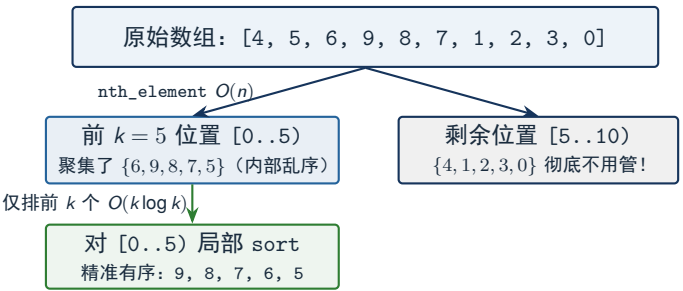
输入：10 / 4 5 6 9 8 7 1 2 3 0 / 5 $\Longrightarrow$ 输出：9，8，7，6，5（每行一个）。

本页要点

先读 n 个数组元素，最后才读 k！而且一定不能去重，有两个 9 就要输出两个 9。

例 3 | 怎么想到的？快速选择减治 + 局部排序建模图解

- 虽然直接全局 sort 的 $O(n \log n)$ 也能过 $n = 10^5$，但我们只需要前 k 个，没必要把剩下 $n - k$ 个垃圾数字也费劲排好序！



本页要点

两步组合拳：先 nth_element 在 $O(n)$ 内圈出前 k 大，再 sort 前 k 个按降序输出！

例 3 | 原题样例两阶段手推表

针对样例 n=10，a=[4,5,6,9,8,7,1,2,3,0]，k=5，记录数组状态演变：

执行阶段	数组 a[0..9] 状态变化	此时保证的性质
0. 读入完成	[4，5，6，9，8，7，1，2，3，0]	读完数组后最后读出 k = 5
1. nth_element 结束	前 5 项可能是 [7，9，8，6，5]；后 5 项 [4，1，2，3，0]	前 5 项每个都 $\geq$ 后 5 项任何数，但内部未必降序
2. sort 前 k 项	前 5 项变成 [9，8，7，6，5]；后 5 项不变	a[0..4] 严格从大到小排列
3. 循环输出 0..k-1	逐行打印 9，8，7，6，5	当 i == 5 时退出，与样例一致

本页要点

特别注意第 1 步到第 2 步的区别：nth_element 之后前 k 个可能还是乱序的，必须再排！

例 3 | 核心步骤与相对模板的改动

按顺序执行	纸上得到	核对问题
1. 读 n，开数组读 n 个数	装满 vector<long long> a(n)	不要急着读 k！k 在最后一行
2. 读入 k	得到要输出的数量	$k < n$ 时 a.begin() + k 合法吗？
3. 调 nth_element(..., greater<>())	前 k 大聚集在 [0, k)	传了 greater<long long>() 吗？
4. 调 sort(..., greater<>())	仅对 [0, k) 降序排列	只输出了前 k 项吗？

本页要点

相对模板改动：按题面先读数组最后读 k，并用 long long 存储；a.begin() + k 刚好划出前 k 个最大数。

例 3 | 完整可编译代码 (1/2)：按题面顺序读入

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

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int n, k;
    if (!(cin >> n)) return 0;
    vector<long long> a(n);
    for (auto &x : a) cin >> x;                    // 先读完 n 个整数
    cin >> k;                                      // 最后才读入第 3 行的 k!
```

本页要点

写惯了“先读 n 和 k 再读数组”的同学，在这题最容易因为先读了 k 导致整个数组错位！

例 3 | 完整可编译代码 (2/2)：快速选择与局部排序（code/top_k.cpp）

```
    nth_element(a.begin(), a.begin() + k, a.end(), greater<long long>());
    sort(a.begin(), a.begin() + k, greater<long long>());
    for (int i = 0; i < k; ++i) cout << a[i] << '\n';
    return 0;
}
```

本页要点

两步组合拳：`nth_element` 以 $O(n)$ 将前 k 大聚到 `[0, k)`（不删重），随后 `sort` 排前 k 项降序输出！

例 3 | 易错点与 3 组自测边界

常见易错

- 易错 **1**（读错顺序 **WA**）：写成 `cin >> n >> k;`，把第 1 个数组元素当成了 `k`；
- 易错 **2**（方向写反 **WA**）：忘传 `greater<long long>()`，输出了最小的 k 个数；
- 易错 **3**（输出乱序 **WA**）：只写了 `nth_element` 就直接输出，前 k 个数没有按从大到小排好；
- 易错 **4**（多余去重 **WA**）：自己调用了 `unique` 去重，违背“重复元素需重复输出”的题意。

三组自测数据（输入 → 预期输出）

- 原题样例：`10 / 4 5 6 9 8 7 1 2 3 0 / 5` → `9, 8, 7, 6, 5`；
- 重复值测试：`5 / 5 1 5 3 5 / 3` → `5, 5, 5`；
- 最小与负数：`2 / -10 -20 / 1` → `-10`。

本页要点

交题前测一组“全重复元素”和一组“全负数”，确认不去重且输出最大那个！

分治算法高频错误对照表

题目 / 场景	常见错误写法	正确做法与原因
快速幂重复递归	写 `qpow(b,p/2)*qpow(b,p/2)`	退化成 $O(p)$ 指数次调用必 **TLE**，必须先存 `half`
快速幂 $k=1$ 特判	边界写 `if(p==0) return 1;`	模 1 时任何数都是 0，写成 `return 1 % k;`
大模数乘法溢出	写 `(a % k) * (b % k)` 用 `long long`	若 $k > 3\times10^9$ 两数相乘溢出，强转 `(__int128)a * b % k`
归并逆序对等号	合并写 `if (a[i] < a[j])`	相同元素不算逆序，必须带等号 `<=` 先出左半
`nth_element` 用法	忘了写第 4 参数 `greater<>()`	默认挑前 k 小；且挑完后要再局部 `sort` 前 k 项
大数据输出刷新	日程表 1024×1024 用 `endl`	一百多万字符配合 `endl` 会直接 **TLE**，改 `'\n'`

本页要点

分治提速的两大前提：算过的一半绝不重算，只进一侧时果断丢掉另一半！

看懂评测结果：CE / WA / RE / TLE / MLE 定位指南

结果	本章最常见元凶	快速排查步骤
CE	忘了 `#include <algorithm>` 或错写 `__int128`	确认使用 64 位 C++17 编译；补全头文件
WA	$k = 1$ 返回了 1、乘法溢出、`nth_element` 后没 `sort`	测 2 0 1、大模数、重复/逆序样例
RE	固定数组开小（如日程表开 100 而非 1024）、死递归	检查 $n = 2^M$ 上限；检查 $L = 1, R = 2$ 能否缩小
TLE	快速幂写两次递归、日程表用 `endl`、归并新建数组	确认每层只调一次 `qpow`；临时数组开在外部
MLE	递归每层按值传参复制整个 `vector`	递归函数传数组必须加引用 `vector<int>&`

本页要点

记住一条隐藏性能坑：递归函数传 `vector` 忘了加引用 `&`，每层拷贝一次直接 **TLE/MLE**！

提交前检查清单（六必查）

1. 分治递归中相同的子结果是否只递归求一次并存到局部变量（绝不写两次 `f(n/2)*f(n/2)`）？
2. 分治底线边界（如 $p = 0$ 或区间长度为 1）是否覆盖了 $k = 1$、$n = 1$ 等最小数据？
3. 大整数乘法取模时是否用（`__int128`）防止两个 10^9 以上数字相乘溢出？
4. 使用 `nth_element` 找第 k 大时，是否写明了 `greater<>()` 并对前 k 项补做了 `sort`？
5. 棋盘/网格类倍增分治是否验证过 $M = 1$ 到 $M = 10$（1024×1024）数组大小够不够？
6. 百万输出或 10^5 级读入时，是否关闭流同步并使用 `'\n'`？

最后一步

对快速幂测一组 $b = 2, p = 0, k = 1 \to 0$，对日程表测 $M = 10$ 耗时，确认秒出结果再提交！

本章全部练习与完成标准

题号	题目名称	难度	核心思路与代码路径
1326	取余运算（mod）	[基础]	分治快速幂 + __int128 + 1%k，code/mod_pow.cpp
1234	2011	[基础]	找 mod10000 循环节（周期 500）或快速幂
1325	循环比赛日程表	[基础]	从 1 × 1 倍增四象限填充，code/schedule_table.cpp
1327	黑白棋子的移动	[提高]	归约成 $n-1$ 规模，底座 $n=4$ 查固定构造
1235	输出前 k 大的数	[入门]	nth_element + 局部 sort，code/top_k.cpp
1237	求排列的逆序数	[基础]	归并统计跨界逆序 mid-i+1，开 long long

本页要点

完成标准：能画主定理复杂度树；能默写快速幂与归并逆序对模板；独立 AC 以上 6 题。

课后安排与下一篇预告

- 复习：合上课件，动手推导“为什么快速幂写两遍 `qpow(b, p/2)` 会退化成 $O(p)$”，自己画出调用节点树。
- 练习：独立写出《求排列的逆序数》（1237）和《黑白棋子的移动》（1327），重点核对 `long long` 与递归底座 $n = 4$。
- 下一篇预告：正式开启《03_ 搜索专题》第 1 章《DFS 回溯与剪枝优化》——当问题无法切半独立求解，必须穷举试探时，如何用深度优先搜索与“及时刹车（剪枝）”拿到满分！

本章一句话

大问题对半一分为二，算好一半绝不算第二遍，分而治之让复杂度成对数级坍塌。

谢谢！