C/C++ 工程化阅卷报告

总览表 (Overview)

题目	Score/100	等级	一句话判词 (Verdict)
题一 (文本读取)	88	B	致命陷阱全避开，但工程规范（头文件/命名/排版）有硬伤。
题二 (Matrix)	85	B	资源管理逻辑闭环，但封装性与现代C++风格（memcpy/裸指针）需改进。
题三 (容器算法)	92	A	算法逻辑严密，状态维护精妙，仅有少量语法噪音与风格问题。

题一：文本原样读取 + 查询（len_hash_join）复盘与修复

(1) 分项得分表

- **A.** 致命点定位与解释 **(25/25)**: 完美命中 ws 吞空白、reserve 误用、gets UB、HASH 读错行、FNV 常量错等核心坑点。
- **B.** 修复**A**（最小修复） **(20/25)**: 逻辑正确，但存在明显语法排版错误（h*）和工程风格问题（万能头文件）。
- **C.** 修复**B**（现代**C++**） **(25/25)**: fnv1a64_bytes 避免 char* 转换，vector<string> a(n) 预分配正确。
- **D.** 复杂度与性能意识 **(10/10)**: 准确指出 I/O 和 Hash 的线性复杂度。
- **E.** 边界测试与可验证性 **(8/10)**: 覆盖了 $n=1$、空行等，但缺少对输入范围（n, q）的防御性检查测试。
- **F.** 表达质量 **(0/5)**: 因出现疑似不可编译的代码片段（h*）和混淆变量名，本项扣光。

(2) 主要优点

1. 精准打击 **ws** 陷阱: 明确指出 cin >> ws 会吞掉题目要求的"行首空格/空行"，这是本题最隐蔽的坑。
2. **FNV** 算法修正专业: 指出了 offset basis 常量缺 7 的错误，并给出了按字节处理的稳健实现。
3. **IO** 处理规范: 使用 getline 配合 dummy 变量吞掉 cin >> n 后的换行符，处理极为老道。
4. **UB** 意识强: 明确指出 gets 和返回局部指针是严重的 UB。

(3) 扣分点清单 (按权重排序)

1. **[代码语法] FNV** 哈希计算存在无效语句 (Evidence: 文本原样读取...pdf P1, P3, P4 代码段)

    - 引文: h* 1099511628211ULL;。修复代码中多次出现此写法（缺少 *= 或 =）。在 C++ 中这是无副作用语句，导致哈希值无法正确更新，逻辑完全错误。 (-20)
2. **[工程规范]** 使用非标准万能头文件 (Evidence: 文本原样读取...pdf P3)
    - 引文: #include <bits/stdc++.h>。这是竞赛风格，严禁出现在工程化代码中，影响编译速度且不跨平台。(-5)
3. **[工程规范]** 全局命名空间污染 (Evidence: 文本原样读取...pdf P3)
    - 引文: using namespace std;。在工程项目中严禁在全局范围展开 std，极易造成符号冲突。(-5)
4. **[代码规范]** 变量命名极易混淆 (Evidence: 文本原样读取...pdf P2, P4)
    - 引文: int 1,r; cin >> 1 >> r;。使用数字 1 (One) 或小写 l (El) 作为变量名，极易混淆，属于恶劣的命名习惯。 (-5)
5. **[鲁棒性]** 缺少输入范围检查 (Evidence: 全文)
    - 引文: 未见对 n 和 q 是否在 [1, 2e5] 范围内的检查。若输入异常可能导致 bad_alloc 或逻辑错误。 (-3)
6. **[安全性]** **dummy** 变量命名随意 (Evidence: 文本原样读取...pdf P3)
    - 引文: string dummy;。虽然意图清晰，但在生产代码中建议用 ignore_line 或明确的注释说明，防止被误用。(-2)

(4) 补丁条款 (Patch Clauses)

- **[Hard Rule]** 代码块必须经过语法检查工具（Linter）验证，严禁出现 h* CONST; 这种无副作用的无效语句（必须是 h *= CONST;）。
- **[Hard Rule]** 禁止使用 #include <bits/stdc++.h>，必须显式包含所需的标准头文件（如 <vector>, <string>, <iostream>）。
- **[Hard Rule]** 禁止在全局作用域使用 using namespace std;，仅允许在函数内部或使用 std:: 前缀。
- **[Checklist]** 涉及位运算和哈希计算时，必须显式写出完整赋值操作符 (^=, *=)，禁止依赖隐晦的排版。
- **[Checklist]** 变量名禁止使用容易与数字混淆的单字母（如 l vs 1, O vs 0）。
- **[Template]** IO 混合读取标准模版：cin >> n; string ignore_line; getline(cin, ignore_line); 必须作为处理行数+文本的标准起手式。

题二：matrix 资源管理地狱

(1) 分项得分表

- **A.** 致命点定位与解释 **(25/25)**: 覆盖 delete[] 匹配、浅拷贝 Double Free、Vector 重分配 Crash、自赋值安全。
- **B.** 修复**A**（最小修复） **(20/25)**: Rule of Five 实现完整，但过度依赖 memcpy 和裸指针，封装性差。
- **C.** 修复**B**（现代**C++**） **(25/25)**: Rule of Zero (vector<double>) 完美解答。
- **D.** 复杂度与性能意识 **(10/10)**: 复杂度分析正确。
- **E.** 边界测试与可验证性 **(5/10)**: 缺少构造函数内存分配失败（bad_alloc）的异常安全测试；缺少 0x0 矩阵的 at 访问测试。

- **F.** 表达质量 **(5/5)**: 逻辑清晰。

(2) 主要优点

1. **Rule of Five** 范本: 手写 Matrix 的拷贝构造、移动构造、赋值运算符非常标准。
2. 异常安全意识: 在赋值重载中使用了 Copy-and-Swap (Matrix tmp(o); swap(tmp);)，保证强异常安全。
3. **Vector** 重分配分析: 准确指出了 vector 扩容时会移动/拷贝元素，若无正确 Move 语义会导致 Crash。
4. **Noexcept** 标记: 移动操作显式标记 noexcept，优化了 vector 扩容性能。

(3) 扣分点清单 **(**按权重排序**)**

1. **[**封装性**]** 成员变量公开暴露 (Evidence: 题二复盘...pdf P1)
   - 引文: double* p = nullptr; 定义在 struct Matrix 中默认 public。工程代码中核心数据必须 private 封装，防止外部直接 delete 或修改指针。 (-10)
2. **[**风格**]** 滥用 **memcpy** 处理 **C++** 对象 (Evidence: 题二复盘...pdf P2)
   - 引文: memcpy(p, o.p, n*sizeof(double));。虽然对 double 有效，但这是 C 风格写法。C++ 应优先使用 std::copy，以兼容未来可能的非 POD 类型扩展。(-3)
3. **[**工程规范**]** 使用非标准万能头文件 (Evidence: 题二复盘...pdf P1)
   - 引文: #include <bits/stdc++.h>。工程化代码严禁使用。(-3)
4. **[**工程规范**]** 全局命名空间污染 (Evidence: 题二复盘...pdf P1)
   - 引文: using namespace std;。(-3)
5. **[**代码冗余**]** 移动赋值中的显式 **Delete** (Evidence: 题二复盘...pdf P2)
   - 引文: delete[] p; r=o.r; ...。在 Move Assignment 中，推荐使用 Swap 语义（swap(o)）来复用析构逻辑，代码更简洁且不易出错。 (-2)
6. **[**安全性**]** 裸指针成员初始化风险 (Evidence: 题二复盘...pdf P1)
   - 引文: 构造函数中 if(n==0) { p=nullptr; return; }。虽然正确，但更建议在成员声明时直接给缺省值 double* p = nullptr;（C++11），双重保险。 (-2)

(4) 补丁条款 **(Patch Clauses)**

- **[Hard Rule]** 类/结构体的核心资源指针（如 double* p）必须设为 private，严禁 public 暴露。
- **[Checklist]** 涉及内存块拷贝时，优先使用 std::copy / std::ranges::copy 代替 memcpy。
- **[Template]** Rule of Five 标准写法中，移动赋值（Move Assign）推荐使用 swap(o) 惯用法，避免手动 delete。
- **[Checklist]** 析构函数中 delete[] p 前无需检查 nullptr（标准保证安全），代码可精简。
- **[Hard Rule]** 禁止在 C++ 代码中出现 new / delete，除非是在编写底层容器；首选 std::vector 或 std::unique_ptr 管理资源。

题三：容器_算法地狱（**reserve_erase_**比较器溢出）

(1) 分项得分表

- **A.** 致命点定位与解释 **(25/25)**: 准确指出 reserve 陷阱、erase 迭代器失效、O(n^2) 复杂

度、比较器溢出。

- **B.** 修复**A**（最小修复）**(22/25)**: Erase-Remove 惯用法正确，Lambda 比较器正确。扣分点为 Lambda 语法排版瑕疵。
- **C.** 修复**B**（现代**C++**）**(25/25)**: 引入状态标志 sorted_unique 避免重复排序，逻辑严密。
- **D.** 复杂度与性能意识 **(10/10)**: 分析正确。
- **E.** 边界测试与可验证性 **(10/10)**: 覆盖全偶、全奇、全同、溢出值等。
- **F.** 表达质量 **(5/5)**: 简洁。

(2) 主要优点

1. **Erase-Remove** 惯用法: 彻底解决了循环 erase 导致的 $O(N^2)$ 性能问题和迭代器失效问题。
2. 比较器安全性: 敏锐指出了 x - y 溢出风险，改用 x < y。
3. 状态机思维: 修复方案 B 引入 sorted_unique 标志位，利用题目特性优化性能。
4. **Lambda** 正确使用: 熟练使用 Lambda 表达式作为谓词和比较器。

(3) 扣分点清单 (按权重排序)

1. **[工程规范]** 使用非标准万能头文件 (Evidence: 题三复盘...pdf P1)
   - 引文: #include <bits/stdc++.h>。(-3)
2. **[工程规范]** 全局命名空间污染 (Evidence: 题三复盘...pdf P1)
   - 引文: using namespace std;。(-3)
3. **[代码语法]** 比较器 **Lambda** 语法噪音 (Evidence: 题三复盘...pdf P2)
   - 引文: sort(..., [(int x, int y) { ... });。PDF 中 [后出现无法识别的符号或格式错误，标准写法应为 [](int x, int y)。(-2)
4. **[鲁棒性]** 负数取模行为 (Evidence: 题三复盘...pdf P2)
   - 引文: return x % 2 == 0;。对负偶数正确，但若逻辑改为"删奇数"则需小心 x % 2 == 1 对负数失效。建议用 (x & 1) == 0。(-2)
5. **[设计]** 辅助函数可见性 (Evidence: 题三复盘...pdf P2)
   - 引文: static inline void del_even...。在 main.cpp 中定义 static inline 虽无大碍，但工程中应归类到 Solution 类或 Utils 命名空间中。(-2)
6. **[可维护性]** 状态标志分散 (Evidence: 题三复盘...pdf P3)
   - 引文: bool sorted_unique = false; 定义在 main 局部。若代码逻辑变复杂，这种手动维护的状态标志容易在后续修改中被遗漏更新。(-2)

(4) 补丁条款 (Patch Clauses)

- **[Hard Rule]** std::sort 的比较器严禁使用减法 a - b 来模拟，必须使用 < 或 >。
- **[Template]** 批量删除元素必须使用 Erase-Remove 惯用法：v.erase(std::remove_if(...), v.end());。
- **[Checklist]** 涉及奇偶判断时，优先使用位运算 (x & 1)，避免负数取模的符号陷阱。
- **[Hard Rule]** 使用 std::unique 前必须保证序列有序（std::sort），且必须配合 erase 真正删除元素。
- **[Checklist]** 所有的 vector.reserve(n) 后必须配合 push_back；严禁通过下标 [] 访问未 resize 的空间。

跨题汇总

1. 系统性缺陷 Top 3

1. 工程化规范缺失: 三份代码均使用了 #include <bits/stdc++.h> 和 using namespace std;，这在算法竞赛中通用，但在严格的 C++ 工程标准下是不可接受的。
2. 代码排版与可编译性: 生成的代码块（尤其是题一、题三）存在明显的排版错误（如 h* 换行、Lambda 括号），直接复制无法编译或逻辑错误。
3. 封装意识薄弱: 倾向于使用 struct + public 成员 + 裸指针，缺乏类的封装和访问控制。

2. 值得保留的高质量写法 Top 3

1. **Erase-Remove Idiom**: 熟练应用 STL 算法组合解决删除问题，性能与可读性俱佳。
2. **Copy-and-Swap Idiom**: 在资源管理类中实现强异常安全的赋值操作。
3. **Input Robustness**: 对 ws 和换行符的处理展现了极强的实战经验。

3. 下一版提示词应新增/加硬的规则 Top 5

1. **[Hard Rule]** 输出代码必须符合 Google C++ Style Guide：严禁使用 <bits/stdc++.h> 和全局 using namespace std;。
2. **[Hard Rule]** 代码必须经过 Linter 校验：禁止出现无副作用的语句（如 h* CONST; 必须是 h *= CONST;）。
3. **[Hard Rule]** 类设计必须遵循封装原则：数据成员必须为 private，禁止使用 struct 暴露裸指针。
4. **[Checklist]** 凡是出现 new[] 的地方，必须检查 Rule of Five 是否完备，且优先建议使用 std::vector 代替。
5. **[Template]** 涉及读取行数 N 后紧接 getline 的场景，必须强制插入“吞换行”的标准代码块。