1. 题目概览

- **要做什么**：输入 `n` 个整数，输出**去重后的升序序列**。
- **输入/输出**：`n` + `n` 个整数；输出去重、排序后的结果（用空格分隔）。
- **题型感觉**：典型「读入数组 → 排序 → 去重」的小题～

---

2. 解题思路整理（C++ 视角）

- 用一个 `vector<int>` 把 `n` 个数读进来。
- `sort(a.begin(), a.end())` 排序。
- 用"去重三连"：
- `auto it = unique(a.begin(), a.end());`
- `a.erase(it, a.end());`
- 输出 `a`。

一个很干净的结构示意：

```
vector<int> a(n);
for (int i = 0; i < n; ++i) cin >> a[i];

sort(a.begin(), a.end());
a.erase(unique(a.begin(), a.end()), a.end());

for (int x : a) cout << x << ' ';
```

---

3. 头文件与库函数整理

推荐别用 `bits/stdc++.h`（有些 OJ/编译器不支持），这题常用：

- `<iostream>`：`cin / cout`
- `<vector>`：`std::vector`
- `<algorithm>`：`std::sort / std::unique`

---

4. 代码可以改进的地方（你 RE 的真正原因）

你这句是核心坑：

```
vector<int> a;
a.reserve(n);
cin >> a[i];
```

`** 只是“预留容量 capacity”，并不会让 vector 真的拥有 **` **个元素。**

- `reserve(n)` 之后：
- `a.capacity() >= n` ✅（能装下 n 个元素）
- `a.size() == 0` ❌（当前元素个数还是 0）
- `operator[]` 下标访问要求：`0 <= i < a.size()`。
- 所以你 `a[i]` 等价于访问不存在的元素 → **越界** → OJ 很容易 **RE**。

reserve vs resize（最关键的区别）

- `reserve(n)`：
- **只改 capacity，不改 size**
- 不会创建元素
- 常用于配合 `push_back`，避免频繁扩容
- `resize(n)`：
- **直接把 size 变成 n**
- 会创建/补齐元素（对 `int` 会补 0）
- 之后你才能安全用 `a[i]`

你这题的两种正确写法

✅写法 1：直接开好大小（最直观）

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

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

    int n; cin >> n;
    vector<int> a(n);
    for (int i = 0; i < n; ++i) cin >> a[i];

    sort(a.begin(), a.end());
    a.erase(unique(a.begin(), a.end()), a.end());

    for (int x : a) cout << x << ' ';
```

```
    return 0;
}
```

✅写法 2：reserve + push_back（你想用 reserve 的正确姿势）

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

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

    int n; cin >> n;
    vector<int> a;
    a.reserve(n);
    for (int i = 0; i < n; ++i) {
        int x; cin >> x;
        a.push_back(x);
    }

    sort(a.begin(), a.end());
    a.erase(unique(a.begin(), a.end()), a.end());

    for (int x : a) cout << x << ' ';
    return 0;
}
```

> 小夸一下：你后面 `sort + unique + erase` 这一套写得非常标准！RE 只是前面读入阶段的小坑，很多人都会踩的，不是你不会用 vector～

---

5. 本题可以沉淀的小套路 / 小模板（C++ 框架）

这类题基本就是：**读入 → 排序 → unique 去重 → 输出**。

极简骨架：

```cpp
int n; cin >> n;
vector<int> a(n);
for (int i = 0; i < n; ++i) cin >> a[i];

sort(a.begin(), a.end());
```

```
a.erase(unique(a.begin(), a.end()), a.end());

for (int x : a) cout << x << ' ';
```

---

6. 加练

练习题 A（同难度）：去重但保持原顺序

题意：输入 `n` 个整数，输出“第一次出现”的元素序列（去重，但不排序，保持原输入顺序）。

思路：用 `unordered_set<int> seen` 记录出现过的数；没见过就加入答案 `vector<int> ans`。

参考代码：

```
#include <iostream>
#include <vector>
#include <unordered_set>
using namespace std;

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

    int n; cin >> n;
    unordered_set<int> seen;
    vector<int> ans;
    ans.reserve(n);

    for(int i = 0; i < n; ++i){
        int x; cin >> x;
        if(seen.insert(x).second){
            ans.push_back(x);
        }
    }

    for(int x: ans) cout << x << ' ';
    return 0;
}
```

相似点：都是“去重”。 **不同点**：这题不排序，重点是“保持顺序”。

练习题 B（更难一点、更像小工程）：在线维护去重升序集合

题意：

- 第一行 `m`，接下来 `m` 行操作：
- `1 x`：加入数字 `x`
- `2 x`：删除数字 `x`（若不存在则忽略）
- `3`：输出当前集合的升序序列（一行）

思路拆解：

- 用 `set<int> s` 自动保持“去重 + 升序”。
- 插入/删除都是对 `set` 操作。
- 查询输出就遍历 `set`。

代码框架：

```
#include <iostream>
#include <set>
using namespace std;

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

    int m; cin >> m;
    set<int> s;

    for(int i = 0; i < m; ++i){
        int op; cin >> op;
        if(op == 1){
            int x; cin >> x;
            s.insert(x);
        }else if(op == 2){
            int x; cin >> x;
            s.erase(x);
        }else if(op == 3){
            for(int x: s) cout << x << ' ';
            cout << '\n';
        }
    }
    return 0;
}
```

升级点：从“一次性处理数组”升级成“多次操作、一直维护一个结构”，更贴近工程里维护数据状态的感觉～