问题

给定序列与操作：

- **DEL_EVEN**：删除所有偶数，保持剩余元素相对顺序
- **DEDUP_SORT**：去重并升序
- **KTH k**：输出当前第 k 小（1-index），不存在输出 `NA` 保证：第一次出现 **KTH** 之前至少执行过一次 **DEDUP_SORT**（因此任意 KTH 时序列应保持升序）。

给定代码存在 RE / 输出怪 / TLE 等问题。

回答

1) 题意一句话

维护一个整数序列，支持“删偶数、去重排序、查第 k 小”，并保证 **KTH 时必须是升序序列**。

2) 致命点清单（≥5条）

1. **reserve≠resize（直接 RE）**：`reserve(n)` 只扩容量不改 `size()`，此时 `a[i]` 越界写入 → 未定义行为/崩溃。
2. **erase 迭代器失效（必炸/漏删）**：`a.erase(it)` 后 `it` 及其后迭代器失效，你再 `++it` 可能跳元素或崩。
3. **逐个 erase 复杂度灾难（O(n^2) 级 TLE）**：vector 中间 erase 每次搬移 O(n)，在循环里删很多元素会退化到平方级。
4. **比较器 `** 可能溢出（排序 UB/乱序）**：当 x=1e9,y=-1e9，x-y` 超 int 范围；比较器不满足严格弱序会导致 sort 行为未定义/结果怪。
5. **“KTH 前必须升序”的逻辑**：题面保证 KTH 之前执行过 DEDUP_SORT；正确实现应维持不变式：
6. DEDUP_SORT 后序列升序且去重；
7. 之后 DEL_EVEN 只做“删除”，删除不会破坏已升序（子序列仍升序）；
8. 若你的实现因比较器/erase 乱序 → KTH 取 `a[k-1]` 就错。
9. **KTH 里把 a 当“第 k 小”用**：只有在序列已升序时 `a[k-1]` 才等于第 k 小；未维护不变式就会输出错误。

---

3) 修复方案 A｜最小修复 （贴近原写法，先保证不 RE + 不 TLE + 题意正确）

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

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

```
    int n;
    if (!(cin >> n)) return 0;

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

    int q; cin >> q;
    while (q--) {
        string op; cin >> op;
        if (op == "DEL_EVEN") {
            // 修：erase-remove 一次线性删除，避免 O(n^2)
            a.erase(remove_if(a.begin(), a.end(), [](int x){ return x % 2 ==
0; }), a.end());
        } else if (op == "DEDUP_SORT") {
            // 修：安全比较器
            sort(a.begin(), a.end(), [](int x, int y){ return x < y; });
            a.erase(unique(a.begin(), a.end()), a.end());
        } else { // KTH
            int k; cin >> k;
            if (k < 1 || k > (int)a.size()) cout << "NA\n";
            else cout << a[k - 1] << "\n";
        }
    }
    return 0;
}
```

---

4) 修复方案 B｜推荐现代 C++（状态维护更清晰：DEDUP_SORT 幂等化，避免重复 sort）

思路：第一次 DEDUP_SORT 做“排序+去重”，之后序列始终保持“已升序且去重”，后续再调用 DEDUP_SORT 直接跳过即可（因为只有删除操作，不会引入乱序/重复）。

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

static inline void del_even(vector<int>& a){
    a.erase(remove_if(a.begin(), a.end(), [](int x){ return (x & 1) == 0; }),
a.end());
}

static inline void dedup_sort(vector<int>& a){
    sort(a.begin(), a.end());           // 默认就是 x<y
```

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

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

    int n;
    if(!(cin >> n)) return 0;

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

    bool sorted_unique = false;

    int q; cin >> q;
    while(q--){
        string op; cin >> op;
        if(op == "DEL_EVEN"){
            del_even(a);
            // 删除不破坏升序/去重，不需要改标记
        }else if(op == "DEDUP_SORT"){
            if(!sorted_unique){
                dedup_sort(a);
                sorted_unique = true;
            }
            // 若已是 sorted_unique，DEDUP_SORT 幂等：什么都不做
        }else{ // KTH
            int k; cin >> k;
            // 题面保证 KTH 前有 DEDUP_SORT；这里仍保守防御
            if(!sorted_unique){
                dedup_sort(a);
                sorted_unique = true;
            }
            if(k < 1 || k > (int)a.size()) cout << "NA\n";
            else cout << a[k-1] << "\n";
        }
    }
    return 0;
}
```

---

5) 复杂度

- **DEL_EVEN**：O(m) 时间，O(1) 额外空间（m=当前长度）。
- **首次 DEDUP_SORT**：O(m log m) 时间；之后若幂等跳过则 O(1)。KTH：O(1)。

6) ≥6 个边界/反例测试（输入要点 → 期望）

1. **全偶**：n=5: 2 4 6 8 10；DEL_EVEN 后空；KTH 1 → `NA`。
2. **全奇且已升序**：1 3 5 7；DEDUP_SORT 后不变；KTH 4 → 7。
3. **全相同**：7 7 7 7；DEDUP_SORT 后只剩一个 7；KTH 2 → `NA`。
4. **极值溢出压测**：-1e9, 1e9, 0；DEDUP_SORT 后应为 -1e9 0 1e9（旧比较器可能乱）。
5. **反复 DEL_EVEN**：混合奇偶多次删；序列长度单调不增且保持升序（若已排序）。
6. **DEDUP_SORT 后 KTH**：先乱序输入，先 DEL_EVEN 若干，再 DEDUP_SORT，再 KTH 多次，应按升序索引输出。
7. **交替操作**：DEDUP_SORT → DEL_EVEN → DEDUP_SORT(应幂等) → KTH。

---

7) 可复用小模板（≤6行）

```
template<class T, class Pred>
void erase_if(vector<T>& v, Pred p){
    v.erase(remove_if(v.begin(), v.end(), p), v.end());
}
```