基础组排序-题解 $q\omega q$

排序的题目集难度略大，深感抱歉，这里补上题解。

1 重载运算符和 `sort` 函数

对于一个 `struct` 我们可以重载其 `<` 运算符，这样就可以使用 `sort` 进行排序。

```
struct stu {
    int id, score;
    // 这是一个以 score 为第一关键字降序
    // id 为第二关键字升序的 <
    // 这是在结构体中的写法
    bool operator<(const stu &rhs) const {
        if (score != rhs.score) return score > rhs.score;
        return id < rhs.id;
    }
    // 这样 stu a{1,2}, b{2,1}; a < b; 就可以直接比较
    // 结果是一个 bool 类型
    // 等价于调用 a 的 < 运算符
    // 即 a<b ==> a.operator<(b)
} student[N];

// 也可以在结构体之外写
// 此时需要传入两个参数
bool operator<(const stu &lhs, const stu &rhs) const {
    // lhs 表示在 < 号左边的元素
    // rhs 表示在 > 号右边的元素
    // 当然变量名可以任意 但是要注意顺序
    return lhs.score > rhs.score || (lhs.score == rhs.score && lhs.id < rhs.id);
}

// 在使用 sort 函数的时候 如果重载了 < 号
// 那么比较的时候会直接使用重载的 < 比较
sort(student + 1, student + 1 + n); // 表示排序 student[1...n] 左闭右开区间

// 如果没有或者我们希望使用自定义的比较
// 我们需要传入一个函数
bool cmp(const stu &lhs, const stu &rhs) const {
    // lhs 表示在 < 号左边的元素
    // rhs 表示在 > 号右边的元素
    // 当然变量名可以任意 但是要注意顺序
    // 第一个参数在左 第二个参数在右
    return lhs.score > rhs.score || (lhs.score == rhs.score && lhs.id < rhs.id);
}

sort(student + 1, student + 1 + n, cmp); // 表示排序按照 cmp 规定的顺序排序
```

~~PPT 中讲到了 `lambda` 表达式，主要是我用的太习惯了，几乎写 `sort` 都没有使用过 `cmp` 函数，几乎全部都是用 `lambda` 表达式实现，所以课上忘记讲传函数作为参数了。感兴趣可以去了解什么是 `lambda` 表达式。~~

2 【模板】排序

快速排序、归并排序、桶排序和 `sort` 函数均可以通过这道题。

- 快速排序有最坏的情况，最后一组数据把基准 `x=q[l]` 的做法卡掉了，请选择中位数或者随机选择基准

```
void q_sort(int l, int r) {
    if (l >= r) return;
    int x = q[(l + r) / 2], i = l - 1, j = r + 1;
    while (i < j) {
        do i++; while (q[i] < x);
        do j--; while (q[j] > x);
        if (i < j) swap(q[i], q[j]);
    }
    q_sort(l, j);
    q_sort(j + 1, r);
}
```

- 归并排序

```
void merge_sort(std::vector<int> &a, int l, int r) {
    if (l >= r) return;
    int mid = (l + r) >> 1;
    merge_sort(a, l, mid), merge_sort(a, mid + 1, r);
    std::vector<int> b(r - l + 1);
    int i = l, j = mid + 1, k = 0;
    while (i <= mid && j <= r) {
        if (a[i] > a[j]) {
            b[k++] = a[j++];
        } else {
            b[k++] = a[i++];
        }
    }
    while (i <= mid) b[k++] = a[i++];
    while (j <= r) b[k++] = a[j++];
    for (int i = l; i <= r; ++i) a[i] = b[i - l];
}
```

- 堆排序

```
void adjustHeap(std::vector<int> &a, int i, int n) {
    int t = a[i];
    for (int j = i * 2; j <= n; j *= 2) {
        if (j < n && a[j] < a[j + 1]) ++j;
        if (t >= a[j]) break;
        a[i] = a[j];
        i = j;
    }
    a[i] = t;
}

void heap_sort(std::vector<int> &a) {
    int n = a.size() - 1;
    for (int i = n / 2; i >= 1; --i) adjustHeap(a, i, n); // 建堆
    for (int i = n; i > 1; --i) {
        std::swap(a[1], a[i]);
        adjustHeap(a, 1, i - 1);
```

```
    }
}
```

3 奇偶排序

做法非常多，但是我们可以这样规定比较函数：如果奇偶性相同，按照数值顺序，否则奇数更小。

```
bool cmp(int x, int y) {
    if (x % 2 == y % 2) return x < y;
    return x % 2 > y % 2;
}
```

4 考试排名

四关键字比较，只需要把总成绩和语文数学总成绩求出来比较即可。

```
struct stu {
    int id;
    int chinese_score;
    int math_score;
    int english_score;

    int score() const {
        return chinese_score + math_score + english_score;
    }

    int score1() const {
        return chinese_score + math_score;
    }

    bool operator<(const stu &rhs) const {
        if (score() != rhs.score()) return score() > rhs.score();
        if (score1() != rhs.score1()) return score1() > rhs.score1();
        if (chinese_score != rhs.chinese_score) return chinese_score >
rhs.chinese_score;
        return id < rhs.id;
    }
};
```

5 去重排序

可以使用 `unique` 函数。给出一种非 `STL` 的做法。

```
// 首先需要排序
int unique(int *a, int n) {
    int cnt = 0;
    for (int i = 1; i <= n; ++ i) {
        // 如果一个元素是第一个元素或者和前一个元素不同
        // 那么它一定是排序后同类元素的第一个
        if (i == 1 || a[i] != a[i - 1]) {
            a[++ cnt] = a[i];
        }
    }
    // 不同的元素存储在 a[1...cnt]
    return cnt; // 返回一共有多少不同的元素
}
```

6 模拟EXCEL排序

按照题意模拟即可。

```cpp
int cmptype;

struct record {
    std::string id, name;
    int score;

    bool operator<(const record &b) const {
        if (cmptype == 1) return id < b.id;
        if (cmptype == 2) {
            if (name != b.name) return name < b.name;
            return id < b.id;
        }
        if (score != b.score) return score < b.score;
        return id < b.id;
    }
};
```

7 [USACO07DEC] Bookshelf B

降序排序，每次取最大的，直到不能取为止。

8 第k大数

需要 $O(N)$ 的快速选择算法才能通过。~~虽然把快读+sort也放过去了.~~

1. 我们选择一个基准元素x，将数组分成两部分，左边的元素都小于等于x，右边的元素都大于等于x
2. 如果左边的元素个数大于等于k，那么我们在左边找第k大数
3. 否则我们在右边找第$k-num$大数，其中num是左边元素的个数

```cpp
void quick_select(std::vector<int> &a, int l, int r, int k) {
    // 查询a中[l,r]区间中第k大的元素
    if (l >= r) return;
    int i = l - 1, j = r + 1, x = a[rand() % (r - l + 1) + l];
    while (i < j) {
        do ++i; while (a[i] > x); // 降序 所以在左边找小数
        do --j; while (a[j] < x); // 在右边找大数
        if (i < j) std::swap(a[i], a[j]);
    }
    // 快排划分为两个区间 [l,j] 和 [j+1,r]
    // 左区间元素个数为 j-l+1
    if (j - l + 1 >= k) quick_select(a, l, j, k);
    else quick_select(a, j + 1, r, k - (j - l + 1));
}

void solve() {
    int n, k;
    std::cin >> n >> k;
    std::vector<int> a(n);
    for (int &i : a) std::cin >> i;
    quick_select(a, 0, n - 1, k);
```

```
    std::cout << a[k - 1] << '\n';
}
```

9 逆序对

```
using LL = long long; // 逆序对最多会有 n*(n-1) 个 从而需要使用 LL

LL merge_sort(std::vector<int> &a, int l, int r) {
    if (l >= r) return 0;
    int mid = (l + r) >> 1;
    LL res = merge_sort(a, l, mid) + merge_sort(a, mid + 1, r);
    std::vector<int> b(r - l + 1);
    int i = l, j = mid + 1, k = 0;
    // 在每次合并的时候，我们考虑左区间元素大于右区间元素的情况
    while (i <= mid && j <= r) {
        if (a[i] > a[j]) {
            // 由于 a[i...mid] a[j...r]都是有序的
            // 从而 a[i...mid] 中的任意元素和 a[j] 形成逆序对
            // 考虑合并的过程会发现这样可以覆盖所有的逆序对
            res += mid - i + 1;
            b[k++] = a[j++];
        } else {
            b[k++] = a[i++];
        }
    }
    while (i <= mid) b[k++] = a[i++];
    while (j <= r) b[k++] = a[j++];
    for (int i = l; i <= r; ++i) a[i] = b[i - l];
    return res;
}

void solve() {
    int n;
    std::cin >> n;
    std::vector<int> a(n);
    for (int &i : a) std::cin >> i;
    std::cout << merge_sort(a, 0, n - 1) << '\n';
}
```