1. 排序

1.1 快速排序

1.1.1 快速排序

题意

将给定的长度为n $(1 \leq n \leq 1e5)$的数列从小到大排序.

思路

选定数x,让$\leq x$的数在x的左边,$\geq x$的数在x的右边.

双指针i和j分别往中间移动,i移至第一个$\geq x$的位置,j移至第一个$\leq x$的位置,交换$num[i]$和$num[j]$,直至两指针相遇或互相穿过.

可以证明每次将序列分成两部分的长度的期望是$\dfrac{n}{2}$,则平均时间复杂度$O(n\log n)$.最坏时间复杂度$O(n^2)$几乎达不到.

快排是不稳定排序.

代码

```
const int MAXN = 1e+5 + 5;
int nums[MAXN];

void quick_sort(int q[], int l, int r) {  // 快排
    if (l >= r) return;  // 边界

    int i = l - 1, j = r + 1, x = q[l + 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]);
    }

    quick_sort(q, l, j), quick_sort(q, j + 1, r);  // 递归排两侧
}

int main() {
    quick_sort(nums, 1, n);
}
```

1.1.2 快速选择算法

题意

给定长度为n的整数数列和整数k,求数列从小到大排序后的第k个数.

代码

```
const int MAXN = 100005;
int n, k;
int nums[MAXN];

int quick_sort(int l, int r, int k) {
    if (l == r) return nums[l];  // 边界

    int x = nums[l], i = l - 1, j = r + 1;
    while (i < j) {
        while (nums[++i] < x);  // i移到第一个>=x
        while (nums[--j] > x);  // j移到第一个<=x
        if (i < j) swap(nums[i], nums[j]);
    }

    int cnt = j - l + 1;  // j左边有几个数
    if (k <= cnt) return quick_sort(l, j, k);  // 递归排左边
    else quick_sort(j + 1, r, k - cnt);  // 递归排右边
}

int main() {
    cout << quick_sort(1, n, k);
}
```

1.2 归并排序

1.2.1 归并排序

题意

将给定的长度为$n\ \ (1\leq n\leq 1e5)$的数列从小到大排序.

思路

将序列从中间分成两部分,递归排序$left$和$right$,最后合并两个有序的数组.

用双指针维护,初始时两指针分别在两有序数组的最小值处.比较两个指针所指的数哪个更小,小的放进答案数组中,相应的指针后移一位.重复该过程直到某个指针移到序列末尾,再将另一个序列在另一个指针后面的部分放进答案数组中.若两指针所指的数相等,则对第一个指针进行操作.

将序列从中间分成两部分,能分出$\log_2 n$层.两个指针移动的距离之和为n,故合并两序列的复杂度$O(n)$,故总复杂度$O(n\log n)$.

归并排序是稳定排序.

代码

```cpp
const int MAXN = 1e5 + 5;
int nums[MAXN];
int tmp[MAXN];  // 归并排序中暂时存有序序列的数组

void merge_sort(int q[], int l, int r) {
  if (l >= r) return;

  int mid = l + r >> 1;
  merge_sort(q, l, mid), merge_sort(q, mid + 1, r);

  // 归并过程
  int k = 0, i = l, j = mid + 1;  // 双指针i、j初始时指向两序列的起点,即最小数
  while (i <= mid && j <= r) {
    if (q[i] <= q[j]) tmp[k++] = q[i++];
    else tmp[k++] = q[j++];
  }

  while (i <= mid) tmp[k++] = q[i++];  // 左边没放完
  while (j <= r) tmp[k++] = q[j++];  // 右边没放完

  for (i = l, j = 0; i <= r; i++, j++) q[i] = tmp[j];  // 把排序后的结果放回原数组
}

int main() {
  for (int i = 1; i <= n; i++) cin >> nums[i];

  merge_sort(nums, 1, n);
  for (int i = 1; i <= n; i++) cout << nums[i] << ' ';
}
```

1.2.2 逆序对的个数

题意

给定一个有$n\ \ (1\leq n\leq 1e5)$个数的数列,输出其中逆序对的个数.

思路

按归并的思想,将逆序对分成三类:①两数同时出现在左半边;②两数同时出现在右半边;③两数分别在两个半边.

考虑修改归并排序使其能顺便求出逆序对的个数.对应以上三类,有:

①为merge_sort(l,mid).

②为merge_sort(mid+1,r).

③对序列$right$中的每个数,考察$left$中有几个数大于该数.因$left$单调增,若$left$中某一数$>right$中的当前数,则$left$中该数及其后面的数都$>right$中的当前数,即产生$(mid-i+1)$个逆序对.

当一个序列倒序时逆序对最多,此时有$(n-1)+(n-2)+\cdots+1=\dfrac{n(n-1)}{2}$,代入$n=1e5$,约5e9,故要开long long.

代码

```cpp
const int MAXN = 1e5 + 5;
int nums[MAXN];
int tmp[MAXN];  // 归并排序中暂时存有序序列的数组

ll merge_sort(int q[], int l, int r) {
  if (l >= r) return 0;

  int mid = l + r >> 1;
  ll res = merge_sort(q, l, mid) + merge_sort(q, mid + 1, r);

  // 归并过程
  int k = 0, i = l, j = mid + 1;  // 双指针i、j初始时指向两序列的起点,即最小数
  while (i <= mid && j <= r) {
    if (q[i] <= q[j]) tmp[k++] = q[i++];
    else {
      res += (ll)(mid - i + 1);
      tmp[k++] = q[j++];
    }
  }

  while (i <= mid) tmp[k++] = q[i++];  // 左边没放完
  while (j <= r) tmp[k++] = q[j++];  // 右边没放完

  for (i = l, j = 0; i <= r; i++, j++) q[i] = tmp[j];  // 把排序后的结果放回原数组

  return res;
}

int main() {
  for (int i = 1; i <= n; i++) cin >> nums[i];

  cout << merge_sort(nums, 1, n);
}
```

1.2.3 超快速排序

题意

给定一个包含n个相异元素的整数序列,每次操作可交换相邻两元素.问将序列升序排列至少需多少次操作.

有多组测试数据.每组测试数据第一行输入整数$n\ \ (1\leq n\leq 5e5)$.接下来n行每行输入一个整数$a_i\ \ (0\leq a_i\leq 999999999)$.输入$n=0$时表示输入结束,该序列无需处理.数据保证所有测试数据的n之和不超过5e5.

对每组测试数据,输出将序列升序排列至少需多少次操作.

思路

用该操作进行排序类似于冒排.注意到每次操作至多会减少一对逆序对,故答案即原序列中逆序对的数量.

代码

```cpp
const int MAXN = 5e5 + 5;
int n;
ll a[MAXN], tmp[MAXN];

ll merge_sort(int l, int r) {
  if (l == r) return 0;

  int mid = l + r >> 1;
  ll res = merge_sort(l, mid) + merge_sort(mid + 1, r);

  int i = l, j = mid + 1, k = 0;
  while (i <= mid && j <= r) {
    if (a[i] <= a[j]) tmp[k++] = a[i++];
    else {
      res += (ll)mid - i + 1;
      tmp[k++] = a[j++];
    }
  }

  while (i <= mid) tmp[k++] = a[i++];
  while (j <= r) tmp[k++] = a[j++];

  for (int i = l, j = 0; i <= r; i++, j++) a[i] = tmp[j];

  return res;
}

int main() {
  while (cin >> n, n) {
    for (int i = 0; i < n; i++) cin >> a[i];

    cout << merge_sort(0, n - 1) << endl;
  }
}
```

1.3 逆序对

1.3.1 Inversion Counting

原题指路: https://codeforces.com/problemset/problem/911/D

题意 $(2\ \mathrm{s})$

给定一个$1\sim n\ \ (1\leq n\leq 1500)$的排列$a=[a_1,\cdots,a_n]$. 有$m\ \ (1\leq m\leq 2\mathrm{e}5)$个操作, 每个操作输入两个整数$l,r\ \ (1\leq l\leq r\leq n)$, 表示翻转$a[]$的区间$[l,r]$. 每次操作后, 输出$a[]$的逆序对数的奇偶性.

思路

考察翻转长度为len的区间对$a[]$的逆序对数的奇偶性的影响, 设该区间中的有序数对有$cnt=C_{len}^2$个.

有如下情况:

逆序对数	顺序对数	cnt	翻转是否改变$a[]$逆序对数的奇偶性
奇	奇	偶	否
奇	偶	奇	是
偶	奇	奇	是
偶	偶	偶	否

故当且仅当cnt为奇数时, 翻转区间会改变$a[]$的逆序对数的奇偶性. 因$n \leq 1500$, $O(n^2)$暴力求初始时$a[]$的逆序对数的奇偶性后, 每次操作检查翻转区间的cnt值的奇偶性即可.

代码

```
void solve() {
    int n; cin >> n;
    vector<int> a(n + 1);
    for (int i = 1; i <= n; i++) cin >> a[i];

    int inv = 0;  // 逆序对
    for (int i = 1; i <= n; i++) {
        for (int j = i + 1; j <= n; j++)
            inv += a[i] > a[j];
    }

    int parity = inv & 1;  // 逆序对数的奇偶性
    CaseT {
        int l, r; cin >> l >> r;

        int len = r - l + 1;
        if (len * (len - 1) / 2 & 1) parity ^= 1;

        cout << (parity ? "odd" : "even") << endl;
    }
}

int main() {
    solve();
}
```