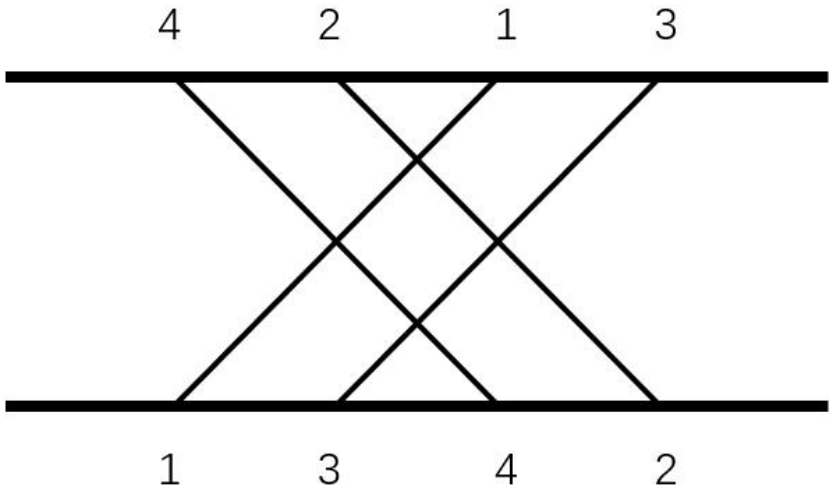
（今天是1月14号，114是个非常神奇的数字，真是太巧了。）

A.调整航线

题目大概是这个样子：

给出一个n与两行各n个的1~n的一种排列，求通过对某一种排列的两相邻值交换进而让两个排列中相等的值一一对应的最小操作数。



这种情况下就至少要4个操作。

题目保证 n <= 10^5。

分析：

既然 n <= 10^5，则一定能想到正解一定是 O(nlogn) 或 O(n)的。且既然题目要求两个排列一一对应，则对于两行输入分别产生的 a 和 b 数组，我们可以进行一下一种操作：

对于 b 数组的每一个值，让他的值为该值在 a 数组中的出现下标。

啊，知道你们可能听不懂这句话，所以“贴心”的白翊扬给你们举个例子：

就拿样例来说，现在 a 数组={4,2,1,3}；b 数组={1,3,4,2}；

如果要让 b 数组每一个位置的值为这个值在 a 数组中的下标（为了让你们理解，下标从 1 开始）。b 数组的第一项 1，它在 a 数组的下标为 3，所以 b 数组第一项修改为 3，其他的也以此类推。

所以处理后的 b 数组={3,4,1,2}；

懂了吗?

现在可以进行下一步操作了。

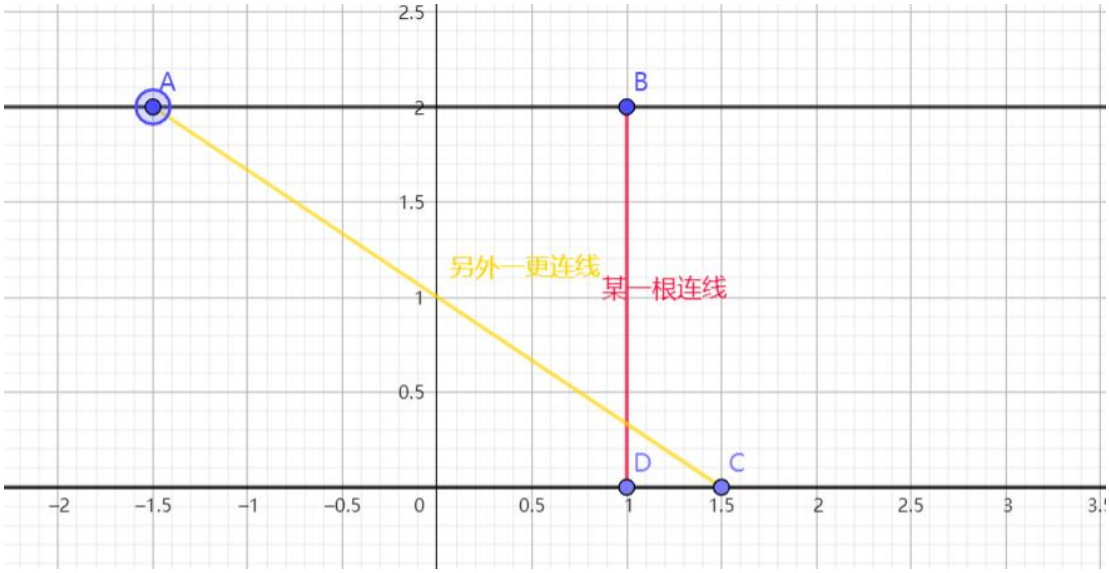
现在对于 b 数组中的一个数，**他的下标和值分别代表某个数**（这个数是多少我们并不关心）在 b 和 a 数组中的出现位置。（就像上面例子中 b 数组下标为 1 的地方是 3，就意味着上面的第 3 个连接着下面的第 1 个）

通过对样例与题目的分析，我们发现，**所有连接线所产生的交点即为答案。**

那什么情况下两条线会产生交点呢?

当然是对于一条线在两端的端点，另一条线两端的交点分别在其两侧，**就是 b 数组某个点的下标和值两个值对应着一大一小**。

我知道你们可能还是听不懂，所以送你们张图。



在这张图中，A、C 分别在 B、D 的两侧，因为我们把两个连接点送进一个数组，也就是 C 点的下标和值一个比 D 点对应的值大，另一个比 D 点对应的值小。

懂了吗?

下标与值一大一小，这不就是**逆序对**吗？？！

所以这就是个求逆序对的题目~。

注：记得开 long long。

核心代码（处理数组的值与下标的一段）：

```cpp
int main(){
    // x是个map，存储在a数组中某个数的下标
    int n;
    scanf("%d",&n);
    for(int i = 0;i < n;i++){
        scanf("%d",&a[i]);
        x[a[i]] = i + 1;
    }
    for(int i = 0;i < n;i++){
        scanf("%d",&b[i]);
        b[i] = x[b[i]];
    }
    pai(0,n - 1); //  归并排序找逆序对
    cout << ans<< endl;

    return 0;
}
```

A. 逆序对翻转问题

题目大意如下：

给一个 1~n 的排列，m 次询问，每次询问区间前反转这个区间（如： {1,2,3}→{3,2,1}）。

再询问问这个排列（不是区间，是整个排列！）中逆序对数的奇偶。

（注意前面反转的区间是会影响之后的反转的，是强制在线的题目）

保证 n,m < 10^5。

分析：

首先，求逆序对肯定要归并排序，但是你看看数据范围，如果每一次询问都来一次，肯定受不了。

分析可得，对每一次询问一定要 O（logn）或者 O（1），你可能想不到什么 O（logn）的解法，所以一定要追求 O（1）。

因为数据两两不相同，所以一段长为 x 的序列中的 x*（x - 1）/2 个对，一定只存在逆序对和顺序对，倒置就是将区间中这两个的数量颠倒了。

想一想，这一段中的数，相对于后面的数还是靠前；相对于前面的数还是靠后，所以它不会对前后产生影响。

如果这个区间内的对数是偶数，那逆序对的奇偶性不会改变，反之亦然。

所以对每一组询问，我们判断一下区间对数的奇偶性，在开个 bool 存答案，遇到奇数就取反，这样就可以在只归并排序一次的情况下实现。

核心代码（奇偶性的判断）：

```
guib(0,n - 1); //归并排序
scanf("%d",&m);
int check = ans & 1; //原本的奇偶性
for(int i = 0;i < m;i++){
    int f,l;
    scanf("%d %d",&f,&l);
    int len = l - f + 1;
    long long kn = len * (len - 1) / 2; //区间内涵的对数
    if(kn & 1){ //对数是奇数，奇偶性翻转
        check = !check;
    }
    if(check){
        cout << "odd" << endl;
    }else{
        cout << "even" << endl;
    }
}
```

C.恢复合影

题目大概是这样的：

给出 n 与 5 个 1~n 的排列，保证一个数至多只在一张照片中会随机挑一个位置放置（不在正确的位置），求正确的排列

保证 n <= 10^5 。

分析：

从两个人开始：在正确的排列中的任意两个人 x 和 y，假设 x 在 y 的左侧，则在这五次照片中，x 最多只有 2 次可能在 y 的右侧（1.x 跑到 y 的右侧 2.y 跑到 x 的左侧）其余都是 x 在 y 的

左侧，因此这 5 次照片中超过 3 次出现 x 在 y 的左侧，那正确顺序 x 一定在 y 的左侧。

既然如此，这不就是个排序规则嘛。

那就很简单了，自定义个排序规则去 sort。

核心代码（排序规则）：

```
bool cmp(int x, int y) {
    int left = 0; // 记录 5 次照片中 x 出现在 y 左侧的次数
    for (int i = 1; i <= 5; i++)
        if (pos[x][i] < pos[y][i]) left++;
    return left >= 3; // 大于三次，一定是正确的顺序
}
```

D. Mas 的异或操作

题目大概也就是这么个样子：

给出 n，k，x 与一个长为 n 的序列，执行 k 次操作，对于每一次操作：

先以升序排列，在将序列中下标为奇数的（下标从 1 开始）的值替换为这个数异或 x（C++中的异或为‘^’）。

输出的这个序列中的最大与最小值。

题目保证 n,k <= 10^5;x,每一个元素 <= 1000;

分析：

首先这个题目理论上必须要你模拟完所有的 k 轮，你应该可以看出来。

从数据范围出发,n,k <= 10^5 就注定了时间复杂度为O（k * 一个不那么大的数）。既然对每一轮都要排序，而且会参与异或操作的所有数都不超过 1000（2^10），也就是结果一定不超过 2^10。这种情况下**计数排**序无疑是最快的（理论上接近线性，在这种题目里可以 压缩到常数级）。

但是 O（k * n）的时间复杂度任然不够，但是想一想，如果我们**直接对桶做操作**，那一轮仅需 1024 次操作，注意一些细节，是能过的。

那怎么针对桶进行操作呢?

显而易见的是，**一个桶里的所有数在操作后是一样的**，所以我们只要知道两个东西:**操作结果、有几个数要操作（那也就知道啦=了有几个数不需要操作）**。

操作结果直接异或即可，需要操作的数的个数要参考桶里第一个数的奇偶性，这也简单，找个变量初始值为奇数，一边遍历一边修改即可。这样可以把一部分的数扔到新的桶里，剩下的留在这。

还有需要注意的是：为了防止干扰，**对桶的操作要重新开一排桶**。结束之后把新的一排桶赋值到原来的并清空即可开始下一轮。

温馨提示：1.尽量用位运算;

2.时间能省就省。

核心代码（桶的遍历与重置）：

```
for(int i = 0;i < k;i++){
    int pos = 0;
    for(int j = 0;j < 1055;j++){
        if(t[j]){
            int ans = j ^ x;
            int sl;
            if(pos % 2 == 1){
                sl = t[j] / 2;
            }else{
                sl = (t[j] + 1) / 2;
            }
            pos += t[j];
            t2[ans] += sl;
            t2[j] += (t[j] - sl);
        }
    }
    cz(); //就是t1与t2两个桶的重置工作
}
```