

XOR Inverse 题面

题面翻译

给定长度为 n ($1 \leq n \leq 3 \times 10^5$) 的数列 $\{a_n\}$ ($0 \leq a_n \leq 10^9$)，请求出最小的整数 x 使 $\{a_n \oplus x\}$ 的逆序对数最少，其中 \oplus 是异或

输入格式

第一行 n ，第二行 $\{a_n\}$

输出格式

两个数，逆序对数和 x

题目描述

You are given an array a consisting of n non-negative integers. You have to choose a non-negative integer x and form a new array b of size n according to the following rule: for all i from 1 to n , $b_i = a_i \oplus x$ (\oplus denotes the operation [bitwise XOR](#)).

An inversion in the b array is a pair of integers i and j such that $1 \leq i < j \leq n$ and $b_i > b_j$.

You should choose x in such a way that the number of inversions in b is minimized. If there are several options for x — output the smallest one.

输入格式

First line contains a single integer n ($1 \leq n \leq 3 \cdot 10^5$) — the number of elements in a .

Second line contains n space-separated integers a_1, a_2, \dots, a_n ($0 \leq a_i \leq 10^9$), where a_i is the i -th element of a .

输出格式

Output two integers: the minimum possible number of inversions in b , and the minimum possible value of x , which achieves those number of inversions.

样例

```
9
10 7 9 10 7 5 5 3 5
```

```
4 14
```

思路

因为涉及到异或操作，当然也是位运算的一种，我们可以想到建立一棵 01trie。

在做这道题以前咱一次 01trie 都没写过，所以这里重点介绍一下

01trie 相似于 trie，每个节点至多有两个子节点，表示下一个位的值为 0 或 1。每个节点都有一个隐藏的值，当前位（这是由创建这个节点的时候递归的位决定的）。这棵树有一个性质，右节点代表的数的值一定比左节点的值大（我们定义右节点代表 1，于是这就是显而易见的了），这是极好的，很方便我们求逆序对。

于是我们根据 `a[i]` 中的每一个数建立一棵 01trie（实现见代码），用这颗 01trie 维护当前数的下标（毕竟要求逆序对），由于插入的下标是根据顺序递增的，它具有单调性，非常方便我们之后的操作（双指针求逆序对）。

对于每一个节点，它带来的总数对个数位左节点的大小乘以右节点的大小（显然），而逆序对可以根据如下代码求出（不做具体解释）：

```
ll ret = 0;
for(int i = 0; i < p[lchild].size(); i++){
    const ll lpos = p[lchild][i];
    while(num < p[rchild].size() && p[rchild][num] < lpos){
        num++;
    }
    ret += num;
}
```

`p[lchild/rchild]` 里存的是有严格上升单调性的下标，于是这段代码就不难理解了（由于有单调性，于是可以使用双指针）

最后我们统计答案，枚举每一位到底是异或（相当于交换左右节点，使得逆序对的个数变成(总数对个数-原逆序对个数)还是不异或更优，**每一位是独立的，于是贪心选择。**

代码

```
#define ll long long
#include<bits/stdc++.h>
using namespace std;
const int MAXN = 3e5 + 9;
int child[MAXN*31][2], tot = 1;
int a[MAXN];
vector<int> p[MAXN * 31];
ll f[33][2];
void insert(int u, int num, int pos, int wei){
    if(wei < 0) return ;
    const int c = (num>>wei) & 1;
    if(!child[u][c]) child[u][c] = ++tot;
    p[child[u][c]].push_back(pos);
    insert(child[u][c], num, pos, wei-1);
    return ;
}
void dfs(int u, int wei){
    const int lchild = child[u][0];
    const int rchild = child[u][1];
    if(lchild)dfs(lchild, wei-1);
    if(rchild)dfs(rchild, wei-1);
    if(!lchild && !rchild) return ;
    int num = 0;
    ll ret = 0;
    for(int i = 0; i < p[lchild].size(); i++) {
```

```
const ll lpos = p[lchild][i];
while(num < p[rchild].size() && p[rchild][num] < lpos){
    num++;
}
ret += num;
}
f[wei][0] += ret;
f[wei][1] += 1ll * p[lchild].size() * p[rchild].size() - ret;
return ;
}
ll answ,ansx;
int n;
int main(){
scanf("%d",&n);
for(int i = 1;i <= n;i++){
    scanf("%d",&a[i]);
    insert(1,a[i],i,30);
}
dfs(1,30);
for(int i = 30;i >= 0;--i){
    answ += min(f[i][0],f[i][1]);
    if(f[i][1] < f[i][0]){
        ansx |= (1ll<<i);
    }
}
printf("%lld %lld",answ,ansx);
return 0;
}
```