T1

从 a_1 开始，按照

$$a_{n+1} = \begin{cases} \frac{a_n}{2}, & if\ a_n\ is\ even, \\ 3 \times a_n + 1, & if\ a_n\ is\ odd. \end{cases}$$

不断计算，直到计算到了 a_m 或第一次进入循环为止。进入循环后的和可以简单计算。

实测在题目范围内，经过很小的步数就能进入循环，且 a_i 不会爆 $long\ long$。

考察知识点

数学 枚举

T2

题目分析

首先对于同一种值，显然持续取队头队尾是最优的。

但是现在有多种值，我们考虑如何合理地取值是最优的。

具体的，我们设初始有一个长度为 n 的值全为 1 的序列 b。

考虑转化题意，那么你操作一次 $[l,r]$ 这个区间，就相当于等价获得 $\sum_{i=l}^{r} b_i$ 的价值，同时，做完此操作后，你需要将 $b_l \leftarrow 0$，$b_r \leftarrow 0$，同时显然一个位置只能操作一次。

这是一个经典的贪心，按照右端点从小到大排序后依次做这些区间即可，这是一个单点加区间求和的过程，可以使用树状数组或线段树解决。

做法：

如果某个流派的符文石在阵列中的位置是 $i_1 < i_2 < \cdots < i_k$，那么最优的配对方式是依次将首尾配对：$(i_1, i_k), (i_2, i_{k-1}), \ldots$。这样我们就能把同一流派的石头成对配好，并确定它们的消除顺序。

对于四个位置 $a < b < c < d$，若配对为 (a,c) 与 (b,d)，无论先后顺序，总能量贡献是一样的。而若配对为 (a,d) 与 (b,c)，则必须先执行 (a,d)，再执行 (b,c)，才能保证最优。

因此，只要保证所有这样的 (a,d) 都在 (b,c) 前面即可。所以按左端点从小到大 / 按右端点从大到小 / 按左右端之间距离从大到小.....都是最优策略之一。

确定最优策略后，考虑怎么计算能量和：每次移除一对符文石后，后面所有石头的下标都会减小。可以用树状数组来动态维护“当前有效位置”。

时间复杂度 $O(n \log n)$。

考察知识点

贪心 简单数据结构

T3

题目分析

首先，我们发现我们需要分别考虑 $a_i \le b_i$，$a_i \ge b_i$ 这两种情况。

那么我们看这个题面，首先我们期望得到的是一路往目标水杯走，不漏水的情况容易处理，我们考虑漏水的情况。但是不一定做得到这种情况，为啥呢？因为另一边可能会漏水，于是我们只需要考虑漏水会漏到哪里。

那么显然漏水的最大高度一定 $<$ 这一段路径的酒管的最高高度，于是我们直接暴力扩展直到找到一个 $\ge$ 这一段路径的酒管的最高高度就停止，于是，我们就得到了一个暴力做法。

考虑优化，发现这段代码的前一半只需要找到一段区间 前缀 / 后缀 第一个 $< now$ 的数即可，这一段容易二分处理，在此不再赘述。

后一半，发现问题等价于求原序列的一段区间的 前缀 / 后缀 max 之和，即

$$\sum_{i=l}^{r}\left(\max_{j=l}^{i} a_j\right) \quad \text{与} \quad \sum_{i=l}^{r}\left(\max_{j=i}^{r} a_j\right).$$

做法：

考虑一次实验，假设 $a < b$（另一种情况对称）。

可以发现：在酒桶 $i\,(a \le i < b)$ 中，酒面必须达到 $\max_{j=i}^{b-1} h_j$；在酒桶 $i\,(i < a)$ 中，酒面必须达到

$$\begin{cases} H, & \max_{j=i}^{a-1} h_j < H, \\ 0, & otherwise, \end{cases}$$

其中 $H = \max_{j=a}^{b-1} h_j$。

可以离线，把询问挂在 b 上。$\sum_i \max_{j=i}^{b-1} h_j$ 可以用单调栈维护，$\sum_i [\max_{j=i}^{a-1} h_j < H]$ 可以在单调栈上二分求出。

也有其他 $online$ 做法。

时间复杂度 $O(n \log n)$。

考察知识点

贪心 倍增 数据结构 单调栈

T4

题目分析

我们要让 $\sum_{i=1}^{n}|a_i-(b_i\oplus x)|$ 这个式子的值最小。

考虑刻画一下想要做出这个问题我们需要啥，设 b_i 为满足 $a_j,b_j\oplus x$ 这两个数最高的不同的二进制位为 2^i 的 j 的个数，发现事实上只需要从高到低依次考虑每个二进制位，即我们只需要让 b 序列倒过来之后的字典序最小即可。

如何求出最高的不同的二进制位呢？我们只需要求出 $a_i\oplus(b_i\oplus x)$ 这个数的 1 所在最高的二进制位即可。

由于异或有结合率，所以 $(a_i\oplus b_i)\oplus x$，设 $c_i=a_i\oplus b_i$，我们对 c_i 建一个 $trie$ 树，都预处理出这一位取了 $0/1$ 时对子树内部的贡献。

在 $trie$ 上进行一遍 dfs 即可。

做法

比较 a_i 和 $b_i\oplus x$ 的大小关系，只需要看 a_i 和 $b_i\oplus x$ 最高位不同的位置，也就是看 $a_i\oplus(b_i\oplus x)$ 最高的 1 的位置。

注意到 $a_i\oplus(b_i\oplus x)=(a_i\oplus b_i)\oplus x$。记 $c_i=a_i\oplus b_i$。从高到低位考虑，若 x 这位是 0，那么对于 c_i 这位是 1 的那些 i，a_i 和 $b_i\oplus x$ 的大小关系就确定了，其贡献也随之确定。x 这位是 1、c_i 这位是 0 同理。否则继续考虑下一位。

对 c_i 建立 01-$trie$，对于 $trie$ 上每个节点，都对更低位预处理出 x 该位取 $0/1$ 时，子树内会带来多少贡献。

在 $trie$ 上 dfs，dfs 到一个节点就表示当前在考虑 x 在这个节点的子树内的情况。每走到一位，就分别考虑 x 这一位是 0 和 1 带来的新贡献。如果选了一边，就加入另一边子树的贡献。这样 dfs 到每个节点时，就对每个更低位计算出了 x 这位取 $0/1$ 时的贡献。最终到达叶子或空节点时，可以快速处理。

时间复杂度 $O(n\log^2 V)$。

代码

```
#include <bits/stdc++.h>
using namespace std;
#define ls u<<1
#define rs u<<1|1
#define ll long long
#define ull unsigned long long
#define ii pair<int,int>
#define vv vector<vector<int>>
#define fi first
#define se second
#define endl '\n'
#define debug(x) cout << #x << ": " << x << endl
#define pub push_back
```

```
#define pob pop_back
#define puf push_front
#define pof pop_front
#define lb lower_bound
#define ub upper_bound
#define i128 __int128
#define Time 1.0*clock()/CLOCKS_PER_SEC
#define pcnt(x) __builtin_popcount(x)
#define mem(a,goal) memset(a,(goal),sizeof(a))
#define vfind(v,x) lower_bound(all(v),x)-v.begin()
#define deal(v) sort(all(v));v.erase(unique(v.begin(),v.end()),v.end())
#define rep(x,start,end) for(int x=(start)-((start)>(end));x!=(end)-((start)>(end));
((start)<(end)?x++:x--))
#define all(x) (x).begin(),(x).end()
#define sz(x) (int)(x).size()
ll rd()
{
    ll a=0; int f=0; char p=getchar();
    while(!isdigit(p)) { f |= p=='-'; p = getchar(); }
    while(isdigit(p)) { a = (a<<3) + (a<<1) + (p^48); p = getchar(); }
    return f ? -a : a;
}
const int INF = 998244353;
const int P   = 1e9+7;
const int N   = 1e6+5;

int n;
int a[N],b[N];
int trie[N][2],tot;
ll dt[N][31][2];
ll sum[N];
ll ans=1e18;
void dfs(int u,int k){
    if(!trie[u][0]){
        ll res=sum[u];
        for(int i=k;i>=0;--i)
            res+=min(dt[u][i][0],dt[u][i][1]);
        ans=min(ans,res);
        return ;
    }
    for(int p=0;p<2;++p){
        sum[trie[u][p]]+=dt[u][k][p]+sum[u];
        for(int i=0;i<k;++i)
            for(int j=0;j<2;++j)
                dt[trie[u][p]][i][j]+=dt[u][i][j];
        dfs(trie[u][p],k-1);
    }
}
int main()
{
    n=rd();
    for(int i=1;i<=n;++i)   a[i]=rd();
```

```
    for(int i=1;i<=n;++i)   b[i]=rd();

    for(int i=1;i<=n;++i){
        int now=0;
        for(int k=30;k>=0;--k){
            if(!trie[now][0])   trie[now][0]=++tot;
            if(!trie[now][1])   trie[now][1]=++tot;
            int pa=a[i]>>k&1;
            int pb=b[i]>>k&1;

            if(pa==0){
                int mask=(1<<k)-1;
                int son=trie[now][pb^1];
                sum[son]+=1<<k;
                sum[son]-=a[i]&mask;
                for(int j=k-1;j>=0;--j){
                    for(int p=0;p<2;++p){
                        dt[son][j][p]+=(p^(b[i]>>j&1))*(1<<j);
                    }
                }
            }else{
                int mask=(1<<k)-1;
                int son=trie[now][pb];
                sum[son]+=1<<k;
                sum[son]+=a[i]&mask;
                for(int j=k-1;j>=0;--j){
                    for(int p=0;p<2;++p){
                        dt[son][j][p]-=(p^(b[i]>>j&1))*(1<<j);
                    }
                }
            }

            now=trie[now][pa^pb];
        }
    }
    dfs(0,30);
    cout<<ans;
    return 0;
}
```

考察知识点

贪心 $trie$ 动态规划 预处理 思维