P2119 [NOIP2016 普及组] 魔法阵

枚举长度 $len = X_d - X_C$，统计在值域上选择两个长度为 $2len, len$ 区间、且间隔超过 $6len$ 的方案数，用前缀和或者阶段DP的思想统计。

复杂度 $O(n^2/c)$，本题中常数 $c = 9$。

U76030 Zjl的彩色弹改(color)

设 $f[l]$ 为以 l 为左端点的合法区间（包含所有颜色）中右端点最小值

$f[l]$ 单调且区间颜色数可以快速维护，做双指针。

```
int cnt, vis[MAXN];//cnt=子段中颜色数量，vis[x]=颜色x出现次数
void add(int x){
    vis[a[x]]++;
    if(vis[a[x]]==1) ++cnt;
}

void del(int x){
    vis[a[x]]--;
    if(vis[a[x]]==0) --cnt;
}
//M = 颜色总数
ll ans = 1e18;
for(int l=1,r=0;l<=N;l++){
    while(r < l) add(++r);
    while(cnt<M && r<N) add(++r);
    if(cnt==M) ans = min(ans, sumv[r]-sumv[l-1]);
    del(l);
}
cout<<ans;
```

U76011 选择客栈2

相同颜色的位置下标一起考虑。

- $f1[l]$ 为以 l 为左端点的合法区间（包含 k 家 $\le p$ 的咖啡店 ）中右端点最小值
- $f2[l]$ 为以 l 为左端点的合法区间（距离 $\le d$ ）中右端点最大值

$f1, f2$ 都可以双指针或者二分维护。

```cpp
//U76011 选择客栈2
ll work(int c){
    ll ans = 0;
    int len = adj[c].size();
    for(int l=0,r1=0,r2=0;l<len;++l){
        while(l == r1 || r1 < len && s[adj[c][r1]] - s[adj[c][l]-1] < k)
++r1;
        while(r2+1 < len && adj[c][r2+1] <= adj[c][l]+d) ++r2;
        if(r1 <= r2 && adj[c][r2] <= adj[c][l]+d) ans += r2 - r1 + 1;
    }
    return ans;
}
```

U234197 区间最值求和

算法1

单调栈找前驱 $l[i]$（最近的 $\geq$）、后继 $r[i]$（最近的 $>$），$a[i]$ 的贡献为 $(r[i]-i)\times(i-l[i])$。

```cpp
for(int i=1;i<=n;i++){
    while(st.size() && a[st.top()]<a[i]){
        r[st.top()] = i;
        st.pop();
    }
    if(st.size()) l[i] = st.top();
    st.push(i);
}
```

算法2

建笛卡尔树，根据子树 sz 也可以找出前驱、后继信息

st表或者单调栈建树，$a[i]$ 的贡献为 左右子树 $sz+1$ 的乘积。

```cpp
stk[++top] = 0;
a[0] = 1e9;
rep(i, 1, n) {
    while(top && a[stk[top]] < a[i]) {
        ls[i] = stk[top];
        --top;
    }
    rs[stk[top]] = i;
    stk[++top] = i;
}
```

算法3

并查集+涨水

按 $a[i]$ 升序考虑，用 vis 标记已经考虑的位置，若 $a[i]$ 旁边的 $i-1, i+1$ 位置已经考虑过（说明 $< a[i]$），则合并集合、同时计算 $a[i]$ 的贡献。

```
#include<bits/stdc++.h>
#define MAXN 1000006
using namespace std;
typedef long long ll;

int n;
int a[MAXN], id[MAXN], vis[MAXN];
int fa[MAXN], sz[MAXN];
int findr(int x){
    if(fa[x]==x) return x;
    else return fa[x] = findr(fa[x]);
}

bool cmp(const int& i, const int& j){
    return a[i] < a[j];
}

ll ans = 0;
void merge(int u, int v){
    int rv = findr(v);
    ans += (ll)a[u] * sz[u] * sz[rv];
    fa[rv] = u;
    sz[u] += sz[rv];
}

int main(){
    ios::sync_with_stdio(0);
    cin>>n;
    for(int i=1;i<=n;i++){
        cin>>a[i];
        fa[i] = id[i] = i;
        sz[i] = 1;
        ans += a[i];
    }

    sort(id+1, id+1+n, cmp);
    for(int i=1;i<=n;i++){
        int u = id[i];
        vis[u] = 1;
        if(u>1 && vis[u-1]) merge(u, u-1);
        if(u<n && vis[u+1]) merge(u, u+1);
    }
    cout<<ans<<'\n';
    return 0;
}
```

算法4

双向链表

将所有数串成一条链（双向链表），从小->大删除每个数，删除一个数时在链表中获知前驱后继信息

```
void del(int i){
    R[L[i]] = R[i];
    L[R[i]] = L[i];
}

int main(){
    cin>>n;
    for(int i=1;i<=n;i++){
        cin>>a[i].v;
        a[i].i = i;
        L[i] = i-1; R[i] = i+1;
    }

    sort(a+1, a+1+n);
    for(int i=1;i<=n;i++){
        int id = a[i].i;
        pre[id] = L[id];
        suf[id] = R[id];
        del(id);
    }
    return 0;
}
```

P6186 [NOI Online #1 提高组] 冒泡排序

直接把之前的结论搬出来。每一轮排序，每一个元素前面比它大的元素减少一个 （如果有）。 设 $rev[x]$ 表示 x 前面有多少个元素比 x 大。

- 操作 1 就是修改两个位置的 $rev[x]$
- 操作 2 是询问 $\sum max\{0, rev[x]-k\} = \sum_{x>k}(rev[x]-k)$

使用两个值域上的数据结构维护 $\sum 1, \sum x$，询问就可以通过在两个数据结构 上询问后缀和求得。

P2127 序列排序

首先dfs找环，对于每个环（大小= n ）贪心地用环中的最小点 $w1$ 进行 $n-1$ 次swap：

$$\begin{aligned} ans1 &= (n-1)\times w1 + \sum_{w\neq w1} w \\ &= (n-2)\times w1 + \sum w \end{aligned}$$

dfs时维护环上最小点权 $w1$ 和总点权 $\sum w$ 即可。

需要注意的是上面贪心有一个例外，也可以用2次swap将全局最小点权 $w0$ 当做环上最小点权 $w1$ 的“替身”：

$$\begin{aligned} ans2 &= (n-1)\times w0 + \sum_{w\neq w1} w + 2(w0+w1) \\ &= (n+1)\times w0 + \sum w{+}w1 \end{aligned}$$

每个环的最终答案 = $min(ans1, ans2)$

P2125 图书馆书架上的书

原数组 a 以及前缀和 s_a，考虑均分纸牌：

- $Ans = \sum |s_a[i] - i \times ave|$

构造数组 $b[i] = a[i] - ave$，前缀和为 s_b，有：

- $Ans = \sum |s_b[i]|$
- $s_b[N] = 0$

均分纸牌的目的就是将 b 数组变为全 0。

对于环形均分纸牌，可以考虑枚举分界点 k （k 不向 $k+1$ 传牌），断环成链，复杂度 $O(N^2)$。

进一步，对于分界点 k ，断环后 b 数组和 s_b 数组变为：

```
b[k+1]          s_b[k+1]-s_b[k]
b[k+2]          s_b[k+2]-s_b[k]
.
.
b[N]            s_b[N]-s_b[k]
b[1]            s_b[1]+(s_b[N]-s_b[k])
.
.
b[k]            s_b[k]+(s_b[N]-s_b[k])
```

注意到有 $s_b[N] = 0$，那么有：

$Ans = \sum |s_b[i]| = \sum |s_b[i] - s_b[k]|$

那么问题变为经典的山区建小学问题，将 s_b 数组排序后取中位数作为 $s_b[k]$ 即可取到最小 Ans 。

输出方案，递推即可：

```
x[k] = 0
x[i] = x[i-1] + a[i] - ave
```

总复杂度1个log。

CF1136D Nastya Is Buying Lunch

交换关系 (u, v) 看成单向边存在`adj[u]`里。

维护一个集合 P 代表【所有Nastya不能跨越的人】，这部分人必须排在Nastya前面。

从队尾往前依次考虑，对于第 i 个人，他能排在Nastya后面当且仅当：他可以和Nastya交换并且也能和 P 集合里的所有人交换，否则只能把 i 加到集合 P 里。

最后输出能排在Nastya后面的人数即可，集合 P 可以用一个数组维护，总复杂度 = 遍历邻接复杂度 $O(n+m)$ 。

暴力实现即可，复杂度是对的。

CF1187D Subarray Sorting

转化：将"子序列 $[l, r]$ 排序" -> 用若干 $len = 2$ 的操作排序 -> 通过相邻 `swap` 消除掉 $[l, r]$ 中所有逆序对

观察到操作"不会产生新的逆序对"，即合法的**必要条件**是"没有新的逆序对"。

设 $pos_a[b[i]] = b[i]$ 在 a 序列中的位置。

没有新的逆序对的意思是：对于 b 序列中的所有元素 $b[i]$，不存在 j 满足 $j < i$ 且 $b[j] > b[i]$ 且 $pos_a[b[j]] > pos_a[b[i]]$

那么：$max\{pos_a[b[j]] | j < i, b[j] > b[i]\} < pos_a[b[i]]$，转化成二维偏序上的最值问题，单点修改后缀求max，可以使用线段树维护。

这个条件的**充分性**可以考虑模拟冒泡排序，构造性证明。

P4378 [USACO18OPEN]Out of Sorts S

询问冒泡排序的趟数。

不妨先将序列离散化为1个排列 p，$p[i]$ = 排序结束时 $a[i]$ 的最终位置

- $rev[p[i]]$ = i 之前 $> p[i]$ 的数字数量
- $id[i]$ = 排序结束时 i 位置上的数的初始下标，排序的过程就是每个数从 $id[i]$ 移动到 i 的过程

```
bool cmp(int i, int j){
    if(a[i]==a[j]) return i < j;//如果不判这个，需要使用stable_sort()
    else return a[i] < a[j];
}

sort(id+1, id+1+N, cmp);
for(int i=1;i<=N;i++) p[id[i]] = i;
```

对于每个 $p[i]$，如果 i 之前有 $> p[i]$ 的数（$rev[p[i]] > 0$），每趟冒泡肯定会有1个跨越到 $p[i]$ 的后面，则说明了可行性。

需要注意的是 $x = p[i]$ 这个数在一趟冒泡排序之后，下标位置也会变化。

答案就是 $max\{rev[p[i]]\}$。

另一种思路是：发现每趟冒泡可以冒出一个当前最大值，但其他数最多向左移动1格，那么若 $i < id[i]$（即结束位置在开始位置之前）则肯定需要 $id[i] - i$ 这么多趟，根据最优性 $ans \geq max(id[i] - i)$，可行性不太好说明。

P4375 [USACO18OPEN]Out of Sorts G

询问鸡尾酒排序的趟数。

注意到对于每个前缀 i，每趟 $0 \to n-2$冒泡都可以让1个且仅有1个 $j > i$ 并且 $id[j] < i$ 的数“逃出”这个前缀跨越到 i 后面，而 $n-2 \to 0$ 冒泡都可以让一个 $id[j] > i$ 并且 $j \leq i$ 的数“回到”这个前缀，即每趟鸡尾酒排序可以从前缀 i 里用 $> i$ 的数换回一个 $\leq i$ 的数。

记 $g[i] = i$ 之前 $id[j] > id[i]$ 的数量；由最优性&可行性可知，$ans = max(g[i])$。

另一种思路，可以发现对于一个值域为 $\{0,1\}$ 的序列进行一轮鸡尾酒排序的作用相当于找到最后一个 0 和第一个 1，如果 1 在 0 前面则交换。于是对于一个 $0/1$ 序列我们很容易计算出经过几轮鸡尾酒排序之后有序：

- 假设序列里有 x 个 0，那么需要的轮数就是前 x 个元素中 1 的个数。

考虑一个阈值 V，设值域为 $\{0,1\}$ 的序列 $b_{V,i}=[a[i]\geq V]$。则 a 有序当且仅当对所有 V，b_V 有序。

所以按 V 递增考虑每一个 b_V 需要几轮鸡尾酒排序之后有序，答案和其取max。需要支持单点 +1 和区间求和，使用树状数组维护。

P4372 [USACO18OPEN]Out of Sorts P

考虑点 i 的贡献，就是 i 点被分裂成单点的轮数。

设 $t[i]$ 为 $a[1...i]$ 这个分割点的出现轮数，则 i 点的贡献就是 $max(t[i-1],t[i])$。

由于每个数每轮只能往左走一格，对于前缀 $a[1...i]$，$t[i]$ 就是 $1...i$ 这些数回到这个前缀的最晚时间，即:

- $t[i]=max\{id[1\sim i]\}-i$
- 注意 $t[i]$ 还要和 1 取max

另一种思路，对于 V，当 b_V 有序时，b_V 的 01 交界处出现一个分界点。01 序列上的冒泡排序也有很简单的形式：

- 每一轮冒泡排序相当于把第一个 1 删掉，并在序列最后插入一个 1。
- 所以将一个 0/1 序列排成有序的次数就是总共 1 的个数减去极长全 1 后缀的长度。

所以我们可以知道每一个位置什么时候出现分界点。一个元素对总代价没有贡献当且仅当两边全部出现分界点，于是可以计算出每个元素贡献几轮，进而计算总代价。

U76006 环上游戏

【题意】

有一个长度为 n 的环，上面的每个位置不重复地放置有 $0...n-1$ 中的一个整数。每秒我们可以同时让任意多的数沿顺时针或逆时针移动一个位置，这个过程中一个位置可以没有数或有多个数。 请问最少需要多少秒，使得每个位置仍恰好只有一个数，且沿顺时针或逆时针依次为 $0...n-1$?

【题解】

首先分顺时针、逆时针2种情况分别作，答案取min。

以顺时针为例，环上从 0 计数。

旗法：考虑每个数 $a[i]=x$，在 $(a[i]-x)\%n$ 处插一个旗子（旗子位置代表其所对应答案中 0 的位置），旗子随着 x 一起转动。那么把 $0...n-1$ 排成1个环相当于把 n 个旗子转到同一个点。

问题就变成把 n 个点移到同一个点，求最小步数。发现在环上，有旗子的点=1，没旗子=0，那么最小步数与最长连续0的长度相关（或者说包含所有1的最小弧长），那么用dp递推求出最长连续0的长度即可。

```
#include<bits/stdc++.h>
#define MAXN 5000005
using namespace std;
```

```
typedef long long ll;

int N;
int a[MAXN];

int cnt[MAXN], dp[MAXN];

int work(){
    for(int i=1;i<=N;i++) cnt[i+N] = cnt[i];
    int ans = 0;
    for(int i=1;i<=2*N;i++){
        if(cnt[i]) dp[i] = 0;
        else dp[i] = dp[i-1] + 1;
        ans = max(ans, dp[i]);
    }
    return ans;
}

int solve(int f){
    memset(cnt, 0, sizeof(cnt));
    for(int i=1;i<=N;i++){
        if(f == 0) cnt[(i-a[i]+N)%N+1]++;
        if(f == 1) cnt[(i+a[i])%N+1]++;
    }
    return (N - work())/2;
}

int main(){
    scanf("%d", &N);
    for(int i=1;i<=N;i++){
        scanf("%d", &a[i]);
    }

    printf("%d\n", min(solve(0), solve(1)));
    return 0;
}
```

P1053 篝火晚会

首先不要被题目迷惑，经过分析，最多仅需要1次命令（1个置换）就能完成。

满足条件的位置关系 = 1个环，并且这个环循环移位 $1 \sim N-1$ 次的形成的新环都可以满足条件。 那么题意就等价为：

- 初始环->目标环的代价 = 不在正确位置上的人数 = N - 在正确位置上的人数
- 给 1 个初始环和 N 个目标环，且 N 个目标环互为循环移位，求最小代价。

看懂题意之后，就可以有一个 $O(N^2)$ 暴力，将初始环和 N 个目标环依次匹配即可。需要注意的是需要顺时针、逆时针各做一遍取min。

为了方便起见，将初始环和目标环调换，认为初始环 $b[1...N]$ 满足位置关系，换去 $1 \sim N$ 的某个循环移位 $c[1...N]$，这样并不影响答案。

为了快速统计不同目标环的“在正确位置上的人数”，采用插旗法：

- 对于初始环上的每个数 $b[i]$ ，在 $(i-b[i]) \pmod N$ 处插旗 `cnt[(i-b[i])%N]++`

- 这里 $cnt[j]$ 代表“若目标环的 1 在 j 处（$c[j]=1$），有多少个数在正确的位置上”

这样插完 N 个旗子之后，找到 $cnt[j]$ 最大的点即可。

复杂度 $O(N)$

P3644 [APIO2015]八邻旁之桥

https://www.luogu.com.cn/article/q1ggwhn0

【算法1】

对于 $k=1$ 的数据，做转化：

- n 个人过桥 -> $2n$ 个人去桥处上小学

在中位数处建桥即可，22分。

【算法2】

对于某个人 (s,t)，不妨设 $s \le t$，它把数轴分成了3个区域，从左->右标记为：

- 区域I：$(-inf,s)$
- 区域II：$[s,t]$
- 区域III：$(t,+inf)$

简单情况是若区域II中有桥，那么肯定走区域II中的桥。

假设2座桥的位置为 $(p1,p2)$，且 $p1$ 在区域I、$p2$ 在区域II，那么此时决策为：

$$dis1=(s-p1)+(t-p1)=(s+t)-2p1$$
$$dis2=(p2-s)+(p2-t)=2p2-(s+t)$$

根据相邻交换的套路，此时令 $dis1<dis2$，能导出条件：

- $s+t<p1+p2$

也就是说，当 $p1,p2$ 确定时，$s+t$ 值小的走 $p1$ 桥，$s+t$ 值大的走 $p2$ 桥。那么：

- 所有人按 $s+t$ 排序，走 $p1$ 桥的人是一个前缀，走 $p2$ 桥的人是一个后缀

那么做法就是按 $s+t$ 排序后枚举分界点，之后在前缀/后缀的中位数上建桥。这要求我们动态维护前/后缀的中位数以及对应的答案，因为要支持插入/删除，开2棵权值线段树可以完成目标。

也可以正反各做一遍，分别计算前缀/后缀的中位数，这样就不用支持删除，用2个堆的经典做法维护即可。

总复杂度 $O(nlogn)$。

2个堆

```
//P3644 [APIO2015]八邻旁之桥
priority_queue<int> q0;
priority_queue<int,vector<int>,greater<int> > q1;

ll s0,s1;
void work(int x){
    if(q0.empty() || x<=q0.top()){
        q0.push(x);
        s0 += x;
```

```
    }
    else{
        q1.push(x);
        s1 += x;
    }

    while(q0.size()>q1.size()){
        q1.push(q0.top()); s1 += q0.top();
        s0 -= q0.top();
        q0.pop();
    }
    while(q1.size()>q0.size()){
        q0.push(q1.top()); s0 += q1.top();
        s1 -= q1.top();
        q1.pop();
    }
}

void solve2(){
    sort(nList+1, nList+1+N);
    for(int k=1;k<=N;k++){
        work(nList[k].s);
        work(nList[k].t);

        ll x = q0.top();
        ans0[k] = (q0.size()*x - s0) + (s1 - q1.size()*x);
    }

    while(!q0.empty()) q0.pop();
    while(!q1.empty()) q1.pop();

    s0 = s1 = 0;
    for(int k=N;k>=1;k--){
        work(nList[k].s);
        work(nList[k].t);

        ll x = q0.top();
        ans1[k] = (q0.size()*x - s0) + (s1 - q1.size()*x);
    }

    ll m0 = INF;
    for(int k=0;k<=N;k++){
        m0 = min(m0, ans0[k] + ans1[k+1]);
    }
    cout<<m0 + ans + N;
    return;
}
```