A. Rectangle in Loop Curve

通过手画可以发现对于任意一个闭合曲线，只要曲线围成的面积不为 0，就能够找到四个点都在曲线上的矩形。

则这题只需要判断输入的 n 个点是否都在一条直线上即可。若都在一条直线上输出 `No`，否则输出 `Yes`。

B. Sequence

做法1：正常LCS做法

以最长上升子序列为例，使用一个数组 h_i 记录当最长上升子序列长度为 i 时，序列末尾最小可能的值，初始$h_0 = 0, h_i = +\infty(i > 0)$。

按顺序枚举 a_i。加入 a_i 时从 h 中找到编号最大的小于等于 a_i 的数，记作h_j。则以 a_i 结尾的最长的上升子序列长度为$j+1$，同时更新 $h_{j+1} = \min(h_{j+1}, a_i)$。

由于此题只需要判断是否存在 $n+1$ 的上升/下降子序列，于是 h 大小只需要 n 。使用二分去寻找 h 中找到编号最大的小于等于 a_i 的数，于是整题的时间复杂度为 $O(n^2 \log n)$

做法2：

```
printf("YE5");
```

通过鸽巢原理可以证明必然存在长度为 $n+1$ 的上升或下降子序列。

这就导致此题的条件非常宽松，各种乱搞均有可能通过此题。

C. Even Component

题目大意为需要从树上删去任意条边，使剩下的偶数大小的连通块尽量的多。

如果连通块大小大于2，则可以继续分割，答案不会更劣。最后我们可以得到若干大小为 1 或 2 的连通块。

则原题就是需要计算 2 连通块的最大数量，就是一个树上最大匹配问题，可以通过树形dp解决。

```
#include<bits/stdc++.h>
using namespace std;
const int maxn = 1e5+10;
vector<int> g[maxn];
int dp[2][maxn];
//dp[0][u] 表示u被匹配时u子树最大匹配数量。
//dp[1][u] 表示u未被匹配时u子树最大匹配数量。
void dfs(int u, int fa){
    for (int v : g[u]) if (v != fa){
        dfs(v, u);
        dp[0][u] = max(dp[0][u] + dp[0][v], dp[1][u] + dp[1][v] + 1);
        dp[1][u] += dp[0][v];
    }
```

```
}
int main(){
    int n;
    cin >> n;
    for (int i = 1; i < n; i++) {
        int u, v;
        scanf("%d%d", &u, &v);
        g[u].push_back(v);
        g[v].push_back(u);
    }
    dfs(1, 0);
    cout << dp[0][1] << endl;
}
```

D. Minimum Sum

最小的 $(a_i + b_i)\%n$ 就是从 b_i 中找到最小的大于等于 $n - a_i$ 的数。如果找不到取 b_i 的最小值。

则将所有 b_i 扔进`multiset`，对 a_i 从前往后一个一个找最优的 b_i，找到一个 b_i 就把它从`multiset`中删除。

```
#include <bits/stdc++.h>
using namespace std;
const int maxn = 200020;
int a[maxn];
multiset<int> s;
int main() {
    int n;
    ios::sync_with_stdio(false);
    cin >> n;
    for (int i = 1; i <= n; i++) cin >> a[i];
    for (int i = 1; i <= n; i++){
        int x;
        cin >> x;
        s.insert(x);
    }
    for (int i = 1; i <= n; ++i){
        int x = n - a[i];
        auto it = s.lower_bound(x);
        if (it == s.end())
            it = s.begin();
        printf("%d ", (a[i] + *it) % n);
        s.erase(it);
    }
    return 0;
}
```

E. Master of Lowbit

当二进制最低位 1 是至少两个连续的 1 时，$+lowbit(x)$ 一定不会更劣。

```
#include<bits/stdc++.h>
using namespace std;
int main(){
    unsigned long long x;
    cin >> x;
    int cnt = 0;
    while (x) {
        unsigned long long b = x & (-x);
        if (x & (b << 1)) {
            x += b;
        }else{
            x -= b;
        }
        cnt++;
    }
    cout << cnt << endl;
}
```

当然，使用记忆化搜索也可以通过。

F. Cutting

答案为

$$\sum_{i=0}^{k} C_n^i$$

证明方法：

设 $f(k,n)$ 为将 k 维空间使用 n 个 $k-1$ 维超平面分割形成的最大区域数量。

则 $f(k,n)$ 它等于用 $n-1$ 个超平面分割的数量+新加入一个超平面后新增的最大区域数量。

新增的区域数量它等于原来的平面将新平面分割的部分个数，它等于 $f(k-1,n-1)$

则可以得到下面的递推式：

$$f(k,n)=f(k,n-1)+f(k-1,n-1)$$

求通项可得：

$$f(k,n)=\sum_{i=0}^{k} C_n^i$$

引用文章：https://blog.csdn.net/frostime/article/details/104980496

G. Lower Bound

对输入的每列士兵排序，询问的时候每列都 `lower_bound` 一下就行了。

```
#include<bits/stdc++.h>
using namespace std;
int w[41][10010];
int main(){
```

```
    int n, K, Q;
    cin >> n >> K >> Q;
    for (int i = 1; i <= K; i++) {
        for (int j = 1; j <= n; j++) cin >> w[i][j];
        sort(w[i] + 1, w[i] + 1 + n);
    }
    long long sum = 0;
    int lastans = 0;
    for (int i = 1; i <= Q; i++) {
        int x;
        cin >> x;
        x ^= lastans;
        int ans = 0;
        for (int j = 1; j <= K; j++) {
            int p = lower_bound(w[j] + 1, w[j] + 1 + n, x) - w[j];
            int val = (p <= n) ? w[j][p] : 0;
            ans ^= val;
        }
        sum += ans;
        lastans = ans;
    }
    cout << sum ;
}
```