概述

	A	B	C	D	E	F
题目参考难度	1	1	3	3	2	2
涉及知识点	暴力枚举	冒泡排序原理	分类讨论，数学	前缀和	字符串	简单模拟

题解报告

> tips:如果有不懂的地方可以在评论区留言.

A. 互质数

暴力枚举即可，事实上跟一个数互质的数远多于跟他有公因子的数。

```
void solve()
{
    int n;
    cin >> n;
    int t = 2;
    while(gcd(t + n, t) > 1) t++;
    cout << t << '\n';
}
```

B. Orange的冒泡排序

题目本质上就是考察冒泡排序，根据冒泡排序的定义模拟排序过程即可，但是需要注意的是，冒泡排序是稳定的排序算法，因此在排序过程中需要判断两个相邻元素是否相等，如果相等则不交换。

```
signed main()
{
    IOS;
    int n;
    cin >> n;
    vector<int> ar(n + 1);
    for(int i = 1; i <= n; i ++)
    {
        cin >> ar[i];
    }
    int ans = 0;
    for(int i = n - 1; i >= 0 ; i --)
    {
        for(int j = 1; j <= i; j ++)
        {
            if(ar[j] > ar[j + 1])
            {
                swap(ar[j], ar[j + 1]);
                ans ++;
            }
        }
    }
    cout << ans << '\n';
    return 0;
}
```

C. 走夜路

题目给出三个坐标，分别为终点 P ，两个圆心的 A, B 的坐标。 现在，我们的起点被设置为在 $(0, 0)$ 的地方，我们要去往终点 P，但是我们只能在两个圆的范围内进行行走，问，两个圆的半径取值 r 最小为多少?

我们钦定原点 $(0, 0)$ 为 O。

事实上，我们可以发现，所有的情况可以被归类为以下4种，2大类:

第一类：两点都在一个圆内

1. 圆 B 太远了，圆 A 自身的范围就可以包裹住起点和终点 P。这时候取值为点 A 距离起点和终点哪个比较远的取值:

$$r = \max(distance(A, O), distance(A, P))$$

2. 圆 A 太远了，圆 B 自身的范围就可以包裹住起点和终点 P。这时候取值为点 B 距离起点和终点哪个比较远的取值:

$$r = \max(distance(B, O), distance(B, P))$$

第二类：两点分别被不同的圆包住

3. 两个圆刚好相切或者相交，同时包裹住了起点和终点，取值刚好相切的半径即可。
4. 两个圆相切或者相交，但是都没有包裹住终点或者起点，这时候就选择两个点当中距离较远的那个点，扩张半径直到触碰到为止。

注意一下，对于第二类情况，应该两两合并讨论。

因此总的表达式为：

$$r = \min(\max(distance(A, O), distance(B, P), \frac{distance(A, B)}{2}), \max(distance(B, O), distance(A, P), \frac{distance(A, B)}{2}))$$

最后，只需要对上述所有情况取最小值即可。

```
using point = pair<double, double>;
#define x first
#define y second
point A, B, P;
double distance(point p1, point p2)
{
    return sqrt(pow(p1.x - p2.x, 2) + pow(p1.y - p2.y, 2));
}
int main()
{
    int _;
    scanf("%d", &_);
    while (_--)
    {
        point O = {0, 0};
        scanf("%lf %lf", &P.x, &P.y);
        scanf("%lf %lf", &A.x, &A.y);
        scanf("%lf %lf", &B.x, &B.y);
        double r1 = max(distance(A, O), distance(A, P));
        double r2 = max(distance(B, O), distance(B, P));
        double r3 = max({distance(A, O), distance(B, P), distance(A, B) / 2});
        double r4 = max({distance(B, O), distance(A, P), distance(A, B) / 2});
        printf("%.10lf\n", min({r1, r2, r3, r4}));
    }
    return 0;
}
```

D. Orange的序列检验

题目给我们两个长度相同的序列，并给出了多次询问，查询给定两个序列相同的对应子区间是否每个位置都不一样，也就是： $\forall i \in [l, r] a_i \neq b_i$。

我们通过高中的逻辑知识已知，要去证明：$\forall i \in [l, r]$ 有 $a_i \neq b_i$，实际上可以等价于去证明 $\exists i \in [l, r]$ 有 $a_i = b_i$ 不成立。

而这就很好做了，我们钦定 $c_i = [a_i = b_i]$ ，那么我们判断对应子区间是否所有位置都不相同，只需要判断 c 序列中 $\sum_{i=l}^{r} c_i = 0$ 即可，显然，我们可以对 c_i 作前缀和从而快速完成这一步操作。

```
int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int n;
    cin >> n;
    vector<int> a(n), b(n);
    for (auto &i : a) cin >> i;
    for (auto &i : b) cin >> i;
    vector<int> c(n + 1);
    for (int i = 0; i < n; i++) c[i + 1] = c[i] + (a[i] == b[i]);
    int q;
    cin >> q;
    while (q--)
    {
        int l, r;
        cin >> l >> r;
        l--;
        cout << (c[r] - c[l] ? "NO" : "YES") << "\n";
    }
    return 0;
}
```

E.Orange的双重回文串

本题主要考察字符串的操作，实际上，首先需要判断给定的 s 是否为回文串，我们只需要把他反转过来变成 s'，在判断是否与原来的 s 相等即可。然后，我们要判断他是否为一个双重回文串，需要先把字符串劈开变成左右相等的两份，然后分别判断是否为回文串即可，这里有一个小推论：如果劈开的 $s1 = s2$，那么他们一定都是回文串，可以自行证明。

```
void solve()
{
    string s;
    cin >> s;
    auto t = s;
    reverse(all(t));
    if(s != t)
    {
        cout << "No\n";
        return;
    }
    auto l = s.substr(0, (s.size() + 1) / 2);
    auto r = s.substr(s.size() / 2);
    if(l != r)  cout << "No\n";
    else cout << "Yes\n";
}
```

F. Orange的幸运数

按照题意模拟即可，注意数据范围，使用long long类型变量。

```
#define int long long
#define endl '\n'

signed main()
{
    int n;
    cin >> n;
    auto work = [&](int u)
    {
        int res = 0;
        while (u)
        {
            res += u % 10;
            u /= 10;
        }
        return res;
    };
    auto check = [&](int t)
    {
        int sum = 0;
        int p = t;
        vector<int> w(1);
        while (p)
        {
            w.push_back(p % 10);
            p /= 10;
        }
        vector<int> ans = w;
        for (int i = 1; i < ans.size(); i++)
        {
            if (i % 2 == 1)
            {
                int t = w[i] * 7;
                if (t < 10)
                {
                    ans[i] = t;
                    continue;
                }
                else
                {
                    while (t > 9)
                        t = work(t);
                    ans[i] = t;
                }
            }
        }
        int ss = 0;
        for (int i = 1; i < ans.size(); i++)
            ss += ans[i];
        if (ss % 8 == 0)
            cout << "Yes" << endl;
        else
            cout << "No" << endl;
    };
    while (n--)
    {
        int t;
        cin >> t;
        check(t);
    }
}
```