补题建议：

1. T1、T2、T3是基础的语法题，建议先补。
2. T4、T5、T6略带一些思维，建议做完T1 T2 T3之后再补。
3. T7是一个经典的二分答案，建议做一道二分查找的模板后再来补。
4. T8是一个高级数据结构，可以用树状数组 + 离散化做也可以用主席树做。第一种方法要求掌握树状数组的模板题；第二种方法要求先学会线段树的模板题。可以拿`查询区间和`练手。

T1

语法题

根据题意判断x与30的大小关系

```
void solve()
{
    int x; cin >> x;
    if(x >= 30) cout << "Yes" << "\n";
    else cout << "No" << "\n";
}
```

T2

语法题

题目要求的是到原点距离≤d的点的个数，ans计数，输出答案

```
void solve()
{
    int n , d;
    cin >> n >> d;
    int ans = 0;
    while(n --){
        int x , y;
        cin >> x >> y;
        if(x * x + y * y <= d * d) ans++;
    }
    cout << ans << "\n";
}
```

T3

暴力枚举

注意到n最大是100，直接三重循环暴力枚举所有组合（第i个边、第j个边、第k个边），要满足三边长度各不相同且任意两边之和大于第三边。ans计数。

```
void solve()
{
```

```
    int n;
    cin >> n;
    vector<int> a(n + 1);
    for(int i = 1; i <= n; i++) cin >> a[i];
    int ans = 0;
    for(int i = 1; i <= n; i++){
        for(int j = i + 1; j <= n; j++){
            for(int k = j + 1; k <= n; k++){
                if(a[i] != a[j] && a[i] != a[k] && a[j] != a[k]){
                    if(a[i] + a[j] > a[k] && a[i] + a[k] > a[j] && a[k] + a[j] >
a[i]){
                        ans++;
                    }
                }
            }
        }
    }
    cout << ans << "\n";
}
```

T4

我们目标是找到第一个被k整除的项。也就是找到第一个x（n）% k == 0。设数列的第n项是xn，则推出同余式：**x(n) = (x(n-1) %k * (10%k)) %k+7%k**（等号是同余符号，两边式子mod k同余），从数列的第一项递推下去即可。

```
void solve()
{
    int k;
    cin >> k;
    int xi = 0;
    int ans = -1;
    for(int i = 1; i <= 1e6; i++){
        xi = (((xi % k) * (10 % k)) % k + 7 % k) % k;
        if(xi % k == 0){
            ans = i;
            break;
        }
    }
    cout << ans << '\n';
}
```

数学式子不容易观察到，手玩样例发现这是一个特殊的高精度。

```
1. input：101
2.
3. output：4
4.
5. 7%101=7
6. 7*10+7=77,77%101=77
7. 77*10+7=777,777%101=70
8. 70*10+7=707,707%101=0
```

```
#include<bits/stdc++.h>
using namespace std;
int main(){
    int n,k=7;scanf("%d",&n);
    for(int i=1;i<=1000000;i++)
    if(k%n==0){printf("%d\n",i);return 0;}
    else k=(k*10+7)%n;
    printf("-1\n");
    return 0;
}
```

T5

贪心

首先给出贪心策略：R的个数为num，那么答案就是1~num中W的数量。采用的操作就是把1到num的每一个W与num+1到n的每一个R交换。这样的策略永远是最优的，因为我们的目标是让序列最终状态为RR...（连续的R)...RW...（连续的W）...WW。那么1到num间的每一个W都需要被操作，1.改成R，2.与别的R的调换。我们发现如果采取2.与别的R调换，就可以实现我们想要的状态：前num个字符都是R，后n-num个字符都是W。

注：string是c++中STL容器，类似于字符数组char s[]，都是用来存字符串。s = “ " + s是为了字符串从下标1开始，举个例子，s = ”abc“，s = ” “ + s的作用就是s = “ ”（空格）+ “abc”，s变成“ abc”。

```
void solve()
{
    int n;
    cin >> n;
    string s;
    cin >> s;
    s = " " + s;
    int cnt = 0;
    for(int i = 1; i <= n; i++){
        if(s[i] == 'R') cnt++;
    }
    int ans = 0;
    for(int i = 1; i <= cnt; i++){
        if(s[i] == 'W'){
            ans ++;
        }
    }
    cout << ans << "\n";
}
```

T6

贪心 + 分类讨论

不妨设x为正数，考虑x点上有一个青蛙，它必须跳k次，希望最后一跳可以尽可能接近原点。那么它一定先向一直向左跳，跳到x % d的位置，这是正半轴上距离原点最近的位置；再向左跳一次，跳到负半轴上x % d - d，这个位置可能比x % d距离原点更近。答案就是min（x % d， |x % d - d|)。明确目标后剩下的任务就是分类讨论我们的跳跃次数k，这决定它最终跳到哪个位置。注意：不是最多跳k步，而是严格地跳k步。use就是跳到x%d花掉的次数，k-use就是到达x%d后还需要跳几次，如果k-use是偶数那么最终位置是x%d，否则是|x%d-d|

```
void solve()
{
    int x , k , d ;
    cin >> x >> k >> d;
    x = abs(x);
    int ans = 0;
    int use = x / d;
    if(use >= k){
        ans = x - d * k;
    }else if(use < k){
        int x1 = x % d;
        int x2 = x % d - d;
        x2 = abs(x2);
        k = k - use;
        if(k % 2) ans = x2;
        else ans = x1;
    }
    cout << ans << "\n";
}
```

T7

二分答案

题意：举个例子，对于样例n = 2 ，k = 3，a1 = 7， a2 = 9。我们可以选择切三刀，一刀切在第一根，两刀切在第二根；也可以选择切三刀，一刀切在第二根，两刀切在第一根；也可以选择切两刀，一刀切在第一根，一刀切在第二根；也可以选择切两刀，都切在第一根。总之这么多种方案中，我们发现最优方案：切三刀，一刀让a1裂成3,4；另外两刀让a2裂成4,4,1，切完后最长的木头是4，我们再也不找到其他的方案使最长的木头小于4.那么4就是我们要的答案。这也就是最短的最长长度的含义。

二分：两个性质：

1. 假设一开始的最长木头是max，我们发现切完后的最长长度一定在1到max之间。
2. 如果有一种方案的最长长度是l，那么最长长度为l+1也一定可以通过方案切出。所以答案序列一定是000...01....1111。0表示不可取，1表示可取。这意味着我们可以通过二分3查找第一个1出现的下标。

剩下就顺利成章了，解释一下check函数：mid作为最长长度，cnt记录了最多需要切几次。如果cnt > k，说明次数不够用，那么此时mid落在了000...000的区间，否则落在了111....111的区间。计算cnt的算法：举个例子：如果a[i]是10，最长长度mid=3，那么至少切3刀，因为切两刀总会有大于3的木头。

```
const int N = 2e5 + 10;
vector<int> a(N);
int n , k;
bool check(int mid)
{
    int cnt = 0;
    for(int i = 1; i <= n; i++){
        if(a[i] % mid){
            cnt += a[i] / mid;
        }else{
            cnt += a[i] / mid - 1;
        }
    }
    if(cnt > k) return false;
    return true;
```

```
}
void solve()
{
    cin >> n >> k;
    int mn = LLONG_MAX , mx = LLONG_MIN;
    for(int i = 1; i <= n; i++){
        cin >> a[i];
        mx = max(mx , a[i]);
        mn = min(mn , a[i]);
    }
    int l = 1 , r = mx;
    while(l < r){
        int mid = l + r >> 1;
        if(check(mid)){
            r = mid;
        }else{
            l = mid + 1;
        }
    }
    cout << l << "\n";
}
```

T8

树状数组+离线处理

这个题用树状数组比较容易写，代码也不多。

题意：查询区间l，r；返回l到r中不同元素的个数。

我们把询问离线出来，排序；让每种元素最后一次出现的下标做贡献。

举个例子：

id	1	2	3	4	5
元素	1	2	1	3	1

1. l = 1 , r = 3 。 1不做贡献，2做贡献，3做贡献。过程是：五个位置的贡献初始化都是0。下面更新贡献：先给idx=1的贡献设为1，第idx=2个贡献设为1，idx=3记录并把上一个1（idx=1）的贡献设为0。计算答案。
2. l = 1 , r = 5。 idx=4的贡献设为1，idx=5的贡献设为1同时上一次1的出现的位置：idx=3的贡献设为0。

对l，r从小到大排序，做一个离线处理，每次计算一对l，r，把答案记录下来，全部计算完统一输出。

大家如果想真正搞懂这道题，建议先去了解一下树状数组or线段树，写一下询问区间和的模板题，再来尝试用离线处理树状数组or主席树ac本题。（本题也可以用主席树写）

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

#define int long long
const int N = 1e6 + 10;
int a[N] , pos[N] , curpos = 1;
int tr[N] , ans[N];
```

```
int n , q;

struct Node{
    int l , r , id;
    bool operator <(const Node &p)const{
        return r < p.r;
    }
}lr[N];

int lowbit(int x)
{
    return x & (-x);
}

void add(int x , int v)
{
    for(int i = x; i <= n; i += lowbit(i)){
        tr[i] += v;
    }
}
int query(int r)
{
    int sum = 0;
    for(int i = r; i >= 1; i -= lowbit(i)){
        sum += tr[i];
    }
    return sum;
}
void f(int r)
{
    for(int i = curpos;i<=r;i++){
        if(pos[a[i]] == 0){
            pos[a[i]] = i;
            add(i,1);
        }else{
            add(pos[a[i]],-1);
            pos[a[i]] = i;
            add(i,1);
        }
    }
    curpos = r+1;
}
void solve()
{
    cin >> n >> q;
    for(int i = 1; i <= n; i++){
        cin >> a[i];
    }
    for(int i = 1; i <= q; i++){
        lr[i].id = i;
        cin >> lr[i].l >> lr[i].r;
    }
    sort(lr + 1,lr + q + 1);
    for(int i = 1; i <= q; i ++){
        f(lr[i].r);
        ans[lr[i].id] = query(lr[i].r) - query(lr[i].l-1);
    }
    for(int i = 1; i <= q; i++) cout << ans[i] << "\n";
```

```
}
signed main()
{
    ios::sync_with_stdio(false);
    cin.tie(0); cout.tie(0);
    solve();
    return 0;
}
```