0. CPP_HEAD

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

typedef long long ll;
typedef long double ld;
typedef pair<ll,ll> pll;

#define endl '\n'


/*----------Consts----------*/
const double eps=1e-6;
const double pi = acos(-1.0);
const long long INF=0x3fffffffffffffff;//15个
/*----------Consts----------*/

#define FORLL(i,l,r) for(ll i=l;i<=r;i++)
#define FORLL_rev(i,r,l) for(ll i=r;i>=l;i--)

#define ALL(A) (A).begin(),(A).end()
#define SORT(A) sort(ALL(A))
#define SORT_REV(A) sort((A).rbegin(),(A).rend())
#define Presentation(i,r) " \n"[i==r]

namespace MOLDULE
{
    const long MOD=1e9+7;
    #define Get_Mod(a) (((a)+MOD)%MOD)
    inline ll inv(ll x) {return CCLIB::qcpow(x,MOD-2);}
    inline ll add(ll x, ll y) {return Get_Mod(x + y);}
    inline ll addto(ll &x, ll y) {return x = add(x, y);}
    inline ll sub(ll x, ll y) {return Get_Mod(x - y);}
    inline ll subto(ll &x, ll y) {return x = sub(x, y);}
    inline ll mul(ll x, ll y) {return Get_Mod(1ll*x * y);}
    inline ll multo(ll &x, ll y) {return x = mul(x, y);}
    inline ll mdiv(ll x, ll y) {return Get_Mod(1ll*x*inv(y));}
    inline ll mdivto(ll &x, ll y) {return x = mdiv(x, y);}
}

#define FAST_IO
#define MUTIPLE_JUDGE

//using namespace MOLDULE;

/*----------Code Area----------*/
const ll N = 200005;
void solve()
{

}
/*----------Code Area----------*/
```

```
/*----------Code Area----------*/

unsigned main(){
    #ifdef FAST_IO
        ios::sync_with_stdio(false);
        cin.tie(nullptr); cout.tie(nullptr);
    #endif

    #ifdef MUTIPLE_JUDGE
        long T; cin >> T;
        while(T--) solve();
    #else
        solve();
    #endif

    return 0;
}
```

1. int128

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

struct int128{
    __int128_t value;

    int128():value(0){}
    int128(ll _val):value(_val){}
    int128(__int128_t _val):value(_val){}

    static int128 trans(string input) {
        bool isNegative = false;
        if (input[0] == '-') {isNegative = true;input = input.substr(1);}
        __int128_t result=0;
        for (char c : input) {result = result * 10 + (c - '0');}
        if (isNegative) {result = -result;}
        return int128(result);
    }

    void print() const {
        __int128_t x = value;
        if (x < 0) {putchar('-');x = -x;}
        if (x > 9) {int128(x / 10).print();}
        putchar(x % 10 + '0');
    }

    int128 operator + (const int128 &b)const{return value+b.value;}
    int128 operator - (const int128 &b)const{return value-b.value;}
    int128 operator * (const int128 &b)const{return value*b.value;}
    int128 operator / (const int128 &b)const{return value/b.value;}
};

istream& operator>>(istream& in,int128& x){
    string Input;
    in >> Input;
    x.trans(Input);
    return in;
}

ostream& operator<<(ostream& out,const int128& x){
    x.print();
    return out;
}

int main(){
    int128 i;
    i=0x3fffffffffffffff;
    ll t;cin >> t;
    ll x;
    while(t--){
        cin >> x;i=i/x;
        i.print();putchar('\n');
    }
    return 0;
}
```

2. 树状数组

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

#define N 100000
typedef long long ll;

#define lowbit(x) ((x) & (-(x))) // 取最后一个1所在位置的权值
struct BITree
{ // 树状数组，下标i从1开始
    vector<ll> Data;

    explicit BITree(ll n) : Data(n * 2 + 5, 0) {}

    void update(ll i, ll dif)
    { // 给予i增量dif,维护树状数组，O(logn)
        while (i < Data.size())
        {
            Data[i] += dif;
            i += lowbit(i);
        }
    }

    ll presum(ll i)
    { // 查询前缀和sum[i]，O(logn)
        ll sum = 0;
        while (i)
        {
            sum += Data[i];
            i -= lowbit(i);
        }
        return sum;
    }

    ll query(ll l, ll r)
    { // 查询区间和
        return presum(r) - presum(l - 1);
    }

    ll operator[](ll index)
    { // 下标调用元素（只读）
        return query(index, index);
    }
};

int main()
{
    ll n, t, tt = 0;
    cin >> n;
    BITree bt(n);
    for (ll i = 1; i <= n; i++)
    {
        cin >> t;
        bt.update(i, t); // 维护原数组，实现单点修改，区间查询
        /*
        bt.update(i,t-tt);tt=t;
        维护差分数组，实现区间修改，单点查询
        对区间[l,r]的修改变为update(l,dif);update(r+1,-dif);
        对元素a[i]的查询变为presum(i);
        */
    } // 建树O(nlogn)
    ll l, r;
    cin >> l >> r;
    cout << bt.query(l, r) << endl;
    ll i, x;
    cin >> i >> x;
    bt.update(i, x);
    cin >> l >> r;
    cout << bt.query(l, r) << endl;
    return 0;
}
```

3. 并查集

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

struct DSU
{
    vector<ll> parents, size;

    explicit DSU(ll n) : parents(n + 1), size(n + 1, 1) {
iota(parents.begin(), parents.end(), 0); }

    ll find(ll x) { return (parents[x] == x) ? x : (parents[x] =
find(parents[x])); }

    void merge(ll a, ll b)
    { // merge a into b
        a = find(a);
        b = find(b);
        if (a == b)
            return;
        if (size[a] > size[b])
            swap(a, b);
        parents[a] = b;
        size[a] += size[b];
    }
};
```

4. 字符串Hash

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

#define N 200005
struct strHash
{ // 字符串哈希
    typedef long long ll;
    typedef pair<ll, ll> pll;
    const ll P1 = 57751, mod1 = 1e9 + 7, P2 = 43331, mod2 = 1e9 + 9;
    size_t length, size;
    vector<ll> hz1, hf1, pz1, pf1, hz2, hf2, pz2, pf2;
    // h:Hash;p:Pow;
    // z:正向;f:反向;
    // 1/2:双Hash;下标从1开始
    strHash(string str)
    {
        length = size = str.length();
        str = ' ' + str;
        hz1.resize(size + 2);
        pz1.resize(size + 2);
        hf1.resize(size + 2);
        pf1.resize(size + 2);
        hz2.resize(size + 2);
        pz2.resize(size + 2);
        hf2.resize(size + 2);
        pf2.resize(size + 2);
        pz1[0] = 1;
        for (int i = 1; i <= size; i++)
        {
            hz1[i] = (hz1[i - 1] * P1 + str[i]) % mod1;
            pz1[i] = pz1[i - 1] * P1 % mod1;
        }
        pf1[size + 1] = 1;
        for (int i = size; i >= 1; i--)
        {
            hf1[i] = (hf1[i + 1] * P1 + str[i]) % mod1;
            pf1[i] = pf1[i + 1] * P1 % mod1;
        }
        pz2[0] = 1;
        for (int i = 1; i <= size; i++)
        {
            hz2[i] = (hz2[i - 1] * P2 + str[i]) % mod2;
            pz2[i] = pz2[i - 1] * P2 % mod2;
        }
        pf2[size + 1] = 1;
        for (int i = size; i >= 1; i--)
        {
            hf2[i] = (hf2[i + 1] * P2 + str[i]) % mod2;
            pf2[i] = pf2[i + 1] * P2 % mod2;
        }
    }
    pll findz(int l, int r)
    { // 返回[l,r]的正向双Hash
        return {((hz1[r] - hz1[l - 1] * pz1[r - l + 1]) % mod1 + mod1) % mod1, ((hz2[r] - hz2[l - 1] * pz2[r - l + 1]) % mod2 + mod2) % mod2};
    }
    pll findf(int l, int r)
    { // 返回[l,r]的反向双Hash
        return {((hf1[l] - hf1[r + 1] * pf1[length - r + l]) % mod1 + mod1) % mod1, ((hf2[l] - hf2[r + 1] * pf2[length - r + l]) % mod2 + mod2) % mod2};
    }
    bool isPalin(ll l, ll r)
    { // 判断[l,r]是否为回文串
        return findz(l, r) == findf(l, r);
    }
};
```

5. 线性筛+欧拉函数

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

const ll N = 1000;
bool check[N + 10]; // check=1表示合数，被筛除
ll phi[N + 10]; // 欧拉函数，phi[i]表示1~i中与i互质的数的个数
ll prime[N + 10]; // 素数表，下标从0开始
ll tot; // 素数的个数
void Phi_and_Prime_Table(ll N)
{
    memset(check, false, sizeof(check));
    phi[1] = 1;
    tot = 0;
    for (ll i = 2; i <= N; i++)
    {
        if (!check[i])
        {
            prime[tot++] = i;
            phi[i] = i - 1;
        }
        for (ll j = 0; j < tot; j++)
        {
            if (i * prime[j] > N)
                break;
            check[i * prime[j]] = true;
            if (i % prime[j] == 0)
            {
                phi[i * prime[j]] = phi[i] * prime[j];
                break;
            }
            else
            {
                phi[i * prime[j]] = phi[i] * (prime[j] - 1);
            }
        }
    }
}
```

6. 归并排序与逆序对计数

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

ll mergeAndCount(vector<ll> &arr, ll l, ll m, ll r)
{
    vector<ll> temp(r - l + 1);
    ll invCount = 0;
    ll i = l, j = m + 1, k = 0;
    while (i <= m && j <= r)
    {
        if (arr[i] <= arr[j])
            temp[k++] = arr[i++];
        else
        {
            temp[k++] = arr[j++];
            invCount += m - i + 1;
        }
    }
    while (i <= m)
        temp[k++] = arr[i++];
    while (j <= r)
        temp[k++] = arr[j++];
    for (ll p = 0; p < temp.size(); p++)
        arr[l + p] = temp[p];

    return invCount;
}

ll mergeSortAndCount(vector<ll> &arr, ll l, ll r)
{
    ll invCount = 0;
    if (l < r)
    {
        ll m = l + (r - l) / 2;
        invCount += mergeSortAndCount(arr, l, m);
        invCount += mergeSortAndCount(arr, m + 1, r);
        invCount += mergeAndCount(arr, l, m, r);
    }
    return invCount;
}
```

7. 大数因子快速随机

```
#include <bits/stdc++.h>
// Pollard_rho 大数因子分解快速随机算法
// Miller-Rabin 素数性测试算法
using namespace std;
typedef long long ll;
ll qcpow_p(ll a, ll b, ll p)
{
    ll ret = 1;
    for (; b; b >>= 1, a = (__int128)a * a % p)
        if (b & 1)
            ret = (__int128)ret * a % p;
    return ret;
}
bool Miller_Rabin(ll p)
{
    if (p < 2)
        return 0;
    if (p == 2 || p == 3)
        return 1;
    ll d = p - 1, r = 0;
    while (!(d & 1))
        ++r, d >>= 1;
    for (ll k = 0; k < 10; ++k)
    {
        ll a = rand() % (p - 2) + 2;
        ll x = qcpow_p(a, d, p);
        if (x == 1 || x == p - 1)
            continue;
        for (int i = 0; i < r - 1; ++i)
        {
            x = (__int128)x * x % p;
            if (x == p - 1)
                break;
        }
        if (x != p - 1)
            return 0;
    }
    return 1;
}
ll Pollard_Rho(ll x)//随机返回x的一个因子
{
    ll s = 0, t = 0;
    ll c = (ll)rand() % (x - 1) + 1;
    int step = 0, goal = 1;
    ll val = 1;
    for (goal = 1;; goal <<= 1, s = t, val = 1)
    {
        for (step = 1; step <= goal; step++)
        {
            t = ((__int128)t * t + c) % x;
            val = (__int128)val * abs(t - s) % x;
            if (step % 8 == 0)
            if (step % 127 == 0)
            {
                ll d = __gcd(val, x);
                if (d > 1)
                    return d;
            }
        }
        ll d = __gcd(val, x);
        if (d > 1)
            return d;
    }
}
int main()
{
    ll t, n;
    cin >> t;
    while (t--)
    {
        cin >> n;
        cout << Pollard_Rho(n) << endl;
    }
    return 0;
}
```

8. 二部图最大匹配

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

typedef long long ll;
struct augment_path
{
    vector<vector<ll>> g;
    vector<ll> pa; // 匹配
    vector<ll> pb;
    vector<ll> vis; // 访问
    ll n, m;         // 两个点集中的顶点数量
    ll dfn;          // 时间戳记
    ll res;          // 匹配数
    augment_path(ll _n, ll _m) : n(_n), m(_m)
    {
        assert(0 <= n && 0 <= m);
        pa = vector<ll>(n, -1);
        pb = vector<ll>(m, -1);
        vis = vector<ll>(n);
        g.resize(n);
        res = 0;
        dfn = 0;
    }
    void add(ll from, ll to)
    {
        assert(0 <= from && from < n && 0 <= to && to < m);
        g[from].push_back(to);
    }
    bool dfs(ll v)
    {
        vis[v] = dfn;
        for (ll u : g[v])
        {
            if (pb[u] == -1)
            {
                pb[u] = v;
                pa[v] = u;
                return true;
            }
        }
        for (ll u : g[v])
        {
            if (vis[pb[u]] != dfn && dfs(pb[u]))
            {
                pa[v] = u;
                pb[u] = v;
                return true;
            }
        }
        return false;
    }
    ll solve()
    ll solve()
    {
        while (true)
        {
            dfn++;
            ll cnt = 0;
            for (ll i = 0; i < n; i++)
            {
                if (pa[i] == -1 && dfs(i))
                {
                    cnt++;
                }
            }
            if (cnt == 0)
            {
                break;
            }
            res += cnt;
        }
        return res;
    } // 返回最大匹配数
};
int main()
{
    ll n, m;
    cin >> n >> m;
    augment_path G(n, n);
    ll u, v;
    for (ll i = 0; i < m; i++)
    {
        cin >> u >> v;
        G.add(u, v);
    }
    cout << G.solve() << endl;
    return 0;
}
```

9. 高精度加乘

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

//高精度正整数计算
namespace CCHA
{
    string HAintadd(const string& num1, const string& num2)
    {
        int len1 = num1.length();
        int len2 = num2.length();
        int diff = len1 - len2;
        int carry = 0;

        if (diff < 0)
            return HAintadd(num2, num1);

        string result(len1 + 1 , '0');

        for (int i = len1 - 1; i >= 0; i--)
        {
            int digitSum = (num1[i] - '0') + (i - diff >= 0 ? num2[i - diff] - '0' : 0) + carry;
            carry = digitSum / 10;
            result[i + 1] = (digitSum % 10) + '0';
        }

        if (carry)
            result[0] = carry + '0';
        else
            result.erase(result.begin());

        return result;
    }

    string HAintmul(const string& num1, const string& num2)
    {
        int len1 = num1.length();
        int len2 = num2.length();
        string result(len1 + len2, '0');

        for (int i = len1 - 1; i >= 0; i--)
        {
            int carry = 0;
            for (int j = len2 - 1; j >= 0; j--)
            {
                int digit = (num1[i] - '0') * (num2[j] - '0') + (result[i + j + 1] - '0') + carry;
                carry = digit / 10;
                result[i + j + 1] = (digit % 10) + '0';
            }
            result[i] = carry + '0';
        }
        }

        size_t pos = result.find_first_not_of('0');
        if (pos != string::npos) result.erase(0, pos);
        else result = "0";

        return result;
    }
}
```

10. 二维计算几何

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

typedef long long ll;
typedef pair<ll, ll> pll;

namespace DEFINITION
{
#define scanfll(a) scanf("%lld", &a)
#define lowbit(x) ((x) & (-(x)))
#define RESET(A) memset(A, 0, sizeof(A))
#define ALL(A) A.begin(), A.end()
#define SORT(A) sort(ALL(A))
#define Presentation(i, r) " \n"[i == r]
#define FORLL(i, l, r) for (ll i = l; i <= r; i++)
#define FORLL_rev(i, r, l) for (ll i = r; i >= l; i--)
#define Get_Mod(a) (((a) + MOD) % MOD)
#define NO "NO\n"
#define YES "YES\n"
}
using namespace DEFINITION;

/*----------Consts----------*/
const double eps = 1e-8;
const double inf = 1e20;
const double pi = acos(-1.0);
/*----------Consts----------*/
int sgn(double x)
{
    if (fabs(x) < eps)
        return 0;
    if (x < 0)
        return -1;
    return 1;
}
inline double sqr(double x) { return x * x; }

struct Point
{
    double x, y;
    Point() {} // Empty Point
    Point(double _x, double _y)
    {
        x = _x;
        y = _y;
    } // Point

    void input() { cin >> x >> y; }

    bool operator==(Point b) const { return sgn(x - b.x) == 0 && sgn(y -
b.y) == 0; }
```

```
    bool operator<(Point b) const { return sgn(x - b.x) == 0 ? sgn(y - b.y)
< 0 : x < b.x; }
    bool operator>(Point b) const { return sgn(x - b.x) == 0 ? sgn(y - b.y)
> 0 : x > b.x; }

    Point operator-(const Point &b) const { return Point(x - b.x, y - b.y);
} // 相减(转向量): A-B=BA
    Point operator+(const Point &b) const { return Point(x + b.x, y + b.y);
} // 向量和

    double operator*(const Point &b) const { return x * b.x + y * b.y; } //
点积
    double operator^(const Point &b) const { return x * b.y - y * b.x; } //
叉积

    double len() { return hypot(x, y); }    // 向量长度
    double len2() { return x * x + y * y; } // 向量长度平方

    double distance(Point p) { return hypot(x - p.x, y - p.y); } // 与另一点
的距离

    Point operator*(const double &k) const { return Point(x * k, y * k); }
    Point operator/(const double &k) const { return Point(x / k, y / k); }

    // 计算 pa 和 pb 的夹角, 就是求这个点看 a,b 所成的夹角
    double rad(Point a, Point b)
    {
        Point p = *this;
        return fabs(atan2(fabs((a - p) ^ (b - p)), (a - p) * (b - p)));
    }

    Point trunto(double r)
    {
        double l = len();
        if (!sgn(l))
            return *this;
        r /= l;
        return Point(x * r, y * r);
    } // 化为长度为 r 的向量

    Point rotleft() { return Point(-y, x); }  // 逆时针旋转 90 度
    Point rotright() { return Point(y, -x); } // 顺时针旋转 90 度

    // 绕着 p 点逆时针旋转angle(弧度制)
    Point rotate(Point p, double angle)
    {
        Point v = (*this) - p;
        double c = cos(angle), s = sin(angle);
        return Point(p.x + v.x * c - v.y * s, p.y + v.x * s + v.y * c);
    }
};
```

```
// 计算凸包
vector<Point> Convex_Hull(vector<Point> pvec)
{
    vector<Point> ch;
    ll n = pvec.size();
    SORT(pvec);
    vector<ll> stk(n + 1);
    ll top = 0;
    stk[++top] = 0;
    vector<bool> used(n + 1, false);
    FORLL(i, 1, n - 1)
    {
        while (top > 1 && (pvec[stk[top]] - pvec[stk[top -
1]]).operator^(pvec[i] - pvec[stk[top]]) <= 0)
            used[stk[top--]] = false;
        stk[++top] = i;
        used[i] = true;
    }
    ll tmp = top;
    FORLL_rev(i, n - 2, 0) if (!used[i])
    {
        while (top > tmp && (pvec[stk[top]] - pvec[stk[top -
1]]).operator^(pvec[i] - pvec[stk[top]]) <= 0)
            used[stk[top--]] = false;
        stk[++top] = i;
        used[i] = true;
    }
    FORLL(i, 1, top)
    ch.emplace_back(pvec[stk[i]]);
    return ch;
}
```