```cpp
// TEAM NAME - Persistent_noobs
// INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

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

#define fastio ios_base::sync_with_stdio(false);cin.tie(NULL);cout.tie(NULL);

#define ll long long
#define ff first
#define ss second
#define pb push_back
#define pf push_front
#define mp make_pair
#define pu push
#define pp pop_back
#define in insert
#define ld long double
#define forn(low,high,i) for(i = low;i < high;i++)
#define forrev(high,low,i) for(i = high; i >= low;i--)
#define all(v) v.begin(),v.end()
#define sz(v) (int)v.size()
#define line cout << __LINE__;
#define prv(a) for(auto x : a) cout << x << ' ';cout << '\n';
#define prvp(a) for(auto x : a) cout << "{" << x.ff << ',' << x.ss << "}";cout <<
'\n';
#define decimal_digits cout << fixed << setprecision(15);
string to_string(string s) { return '"' + s + '"';}
string to_string(char s) { return string(1, s);}
string to_string(const char* s) { return to_string((string) s);}
string to_string(bool b) { return (b ? "true" : "false");}
template <typename A> string to_string(A);
template <typename A, typename B>string to_string(pair<A, B> p) {return "(" +
to_string(p.first) + ", " + to_string(p.second) + ")";}
template <typename A> string to_string(A v) {bool f = 1; string r = "{"; for
(const auto &x : v) {if (!f)r += ", "; f = 0; r += to_string(x);} return r + "}";}
void debug_out() { cerr << endl; }
template <typename Head, typename... Tail> void debug_out(Head H, Tail... T) {cerr
<< " " << to_string(H); debug_out(T...);}
#define pr(...) cerr << "[" << #__VA_ARGS__ << "]:", debug_out(__VA_ARGS__)

typedef unordered_map<int,int> umi;
typedef unordered_map<ll,ll> uml;
typedef vector<int> vi;
typedef vector<ll> vl;
typedef vector<vi> vvi;
typedef vector<vl> vvl;
typedef pair<int,int> pii;
typedef pair<ll,ll> pll;
typedef vector<pii> vpii;
typedef vector<pll> vpll;

mt19937_64 rng(chrono::steady_clock::now().time_since_epoch().count());

const int inf = 1e9;
const ll INF = 1e18;
const int mod = 1e9 + 7;
const int bit32 = log2(inf) + 3;
const int bit64 = log2(INF) + 3;

inline int add(int a, int b){a += b; if(a >= mod) a -= mod; return a;}
inline int sub(int a, int b){a -= b; if(a < 0) a += mod; return a;}
inline int mul(int a, int b){return (int)((long long) a * b % mod);}
inline int modexpo(int a, int b){int res = 1; while(b > 0){ if(b & 1) res = mul
(res, a); a = mul(a, a); b /= 2;} return res;}
inline int divide(int a, int b){ return mul(a, modexpo(b, mod - 2));}

clock_t time_p = clock();
```

```cpp
void ktj(){
  time_p = clock() - time_p;
  cerr << "Time elapsed : " << (float)(time_p)/CLOCKS_PER_SEC << "\n";
}

void pre(){ // Reset each global variable (esp. for graphs)

}

void solve(){

}

int main(){
  fastio

  pre();

  int t = 1;
  cin >> t;
  while(t--){
    solve();
  }

  ktj();
}

//  ---------------------------------------------------------------------------

/*
*   Z - Function
*   Computes the z-array that is for each 0 <= i <= n - 1,
*   z[i] is the length of longest substring starting at i which is also a prefix
of s[0,n - 1]
*   (Prefix may even overlap and contain the index i, so a prefix strictly before
i can be computed by the same algo on the reversed string)
*/

vector<int> z_function(string s){

    int n = s.size();
    vector<int> z(n,0);
    for(int i = 1,l = 0,r  = 0; i < n; i++){
        if(i <= r){
            z[i] = min(r - i + 1,z[i - l]);
        }
        while(i + z[i] < n && s[z[i]] == s[i + z[i]])
            z[i]++;
        if(i + z[i] - 1 > r){
            l = i;r = i + z[i] - 1;
        }
    }
    return z;
}

bool search_pattern(string pattern, string text){
    string s = pattern + '*' + text;
    vector<int> z = z_function(s);

    for(int i = 0;i < (int)text.size();i++){
        if(z[i + (int)pattern.size() + 1] == pattern.size()){
            return 1;
        }
    }
    return 0;
}

//  ---------------------------------------------------------------------------
// Segment Tree
// For large values of N DON'T USE the code inside struct. Make everything global

const int N = 1e5 + 5;
ll a[N];

struct Segtree{

  vector<ll> T, lazy;

  Segtree(int n = N){
    T.resize(4 * n, 0);
    lazy.resize(4 * n, 0);
  }

  void build(int v,int l,int r){
    if(l == r){
      T[v] = a[l];
      lazy[v] = 0;
      return;
    }
    int m = (l + r) >> 1;
    int n1 = 2*v + 1;
    int n2 = 2*v + 2;
    build(n1,l,m);
    build(n2,m + 1,r);

    T[v] = T[n1] + T[n2];
    return;
  }

  void propagate(int v,int start,int end){

    T[v] += (end - start + 1) * lazy[v];
    if(start != end){
      lazy[2*v + 1] += lazy[v];
      lazy[2*v + 2] += lazy[v];
    }
    lazy[v] = 0;
    return;
  }

  void update(int v,int l,int r,int start,int end,ll val){
    if(start > end || start > r || end < l){
      return;
    }
    if(lazy[v]){
      propagate(v,start,end);
    }
    if(l <= start and end <= r){
      T[v] += (end - start + 1) * val;
      if(start != end){
        lazy[2*v + 1] += val;
        lazy[2*v + 2] += val;
      }
      return;
    }
    int m = (start + end) >> 1;
    int n1 = 2*v + 1;
    int n2 = 2*v + 2;
    update(n1,l,r,start,m,val);
    update(n2,l,r,m + 1,end,val);
    T[v] = T[n1] + T[n2];
    return;
  }
  ll query(int v,int l,int r,int start,int end){
    if(start > end || start > r || end < l){
      return 0;
    }
    if(lazy[v]){
      propagate(v,start,end);
    }
    if(l <= start and end <= r){
      return T[v];
    }
    int m = (start + end) >> 1;
    int n1 = 2*v + 1;
    int n2 = 2*v + 2;
    ll p = query(n1,l,r,start,m);
    ll q = query(n2,l,r,m + 1,end);
    return p + q;
  }
};



//  ---------------------------------------------------------------------------

// Sparse Table


const int N = 1e5 + 5;
const int logN = log2(N) + 1;
int lgval[N], a[N];
int t[N][logN];

void precompute(){
  lgval[1] = 0;
  for(int i = 2; i < N; i++){
    lgval[i] = lgval[i/2] + 1;
  }
}

void build(int n){
  for(int i = 0; i < n; i++){
    t[i][0] = a[i];
  }
  for(int j = 1; j < logN; j++){
    for(int i = 0; i + (1 << j) <= n; i++){
      t[i][j] = min(t[i][j - 1], t[i + (1 << (j - 1))][j - 1]);
    }
  }
}

int query(int L, int R){
  int j = lgval[R - L + 1];
  return min(t[L][j], t[R - (1 << j) + 1][j]);
}


//  ---------------------------------------------------------------------------










// SCC
```

```cpp
const int N = 1e5 + 5;

class Kosaraju{
  vi g[N], g_prime[N];
  bool vis[N];

  public:
    void add_edge(int u, int v){
      g[u].pb(v);
    }

    void reset(){
      memset(vis,0,sizeof vis);
    }

    void separate(int v, stack<int> &s){
      vis[v] = 1;
      for(auto x : g[v]){
        if(!vis[x]){
          separate(x,s);
        }
      }
      s.push(v);
    }

    void transpose(int n){
      for(int i = 0; i < n; i++){
        for(auto x : g[i]){
          g_prime[x].pb(i);
        }
      }
    }

    void dfs(int v, vi &tmp){
      vis[v] = 1;
      tmp.pb(v);
      for(auto x : g_prime[v]){
        if(!vis[x]){
          dfs(x,tmp);
        }
      }
    }

    vvi generate_SCC(int n){
      vvi SCC;
      stack<int> s;
      reset();
      for(int i = 0; i < n; i++){
        if(!vis[i])
          separate(i,s);
      }

      transpose(n); reset();

      while(!s.empty()){
        int v = s.top();
        s.pop();
        if(!vis[v]){
          vi tmp; dfs(v,tmp); SCC.pb(tmp);
        }
      }
      return SCC;
    }

};
```

```cpp
// NTT

const int root = 646; // wrt mod 998244353
const int root_1 = modexpo(root, mod - 2);
const int root_pw = 1 << 20;
const int bits = 21;

int pow_w[bits], pow_w_1[bits];
const int R = 5e6 + 5;
int sizes[R];

void pre(){
  pow_w[0] = root;
  pow_w_1[0] = root_1;

  for(int i = 1; i < bits; i++){
    pow_w[i] = modexpo(pow_w[i - 1], 2);
    pow_w_1[i] = modexpo(pow_w_1[i - 1], 2);
  }

  for(int i = 1; i < R; i++){
    if(__builtin_popcount(i) == 1){
      sizes[i] = i;
    }else{
      int msb = 32 - __builtin_clz(i);
      sizes[i] = 1 << msb;
    }
  }
}

void fft(vl &a, bool invert){
  int n = sz(a);
  for(int i = 1, j = 0; i < n; i++) {
    int bit = n >> 1;
    for (; j >= bit; bit >>= 1)
      j -= bit;
    j += bit;
    if(i < j)
      swap(a[i], a[j]);
  }
  int itr = 1;
  for(int len = 2; len <= n; len *= 2){
    ll wn; itr++;
    if(invert){
      wn = pow_w_1[bits - itr];
    }else{
      wn = pow_w[bits - itr];
    }
    for(int i = 0; i < n; i += len){
      ll w = 1;
      for(int j = 0; j < len / 2; j++){
        ll u = a[i + j], v = mul(a[i + j + len / 2], w);
        a[i + j] = add(u, v);
        a[i + j + len / 2] = sub(u, v);
        w = mul(w, wn);
      }
    }
  }

  if(invert){
    ll n_ = modexpo(n, mod - 2);
    for(auto &c : a){
      c = mul(c, n_);
    }
  }
}

void multiply(vl &a, vl b){
  int s = sz(a) + sz(b);
  int r = sizes[s];
  a.resize(r);
  b.resize(r);
  fft(a, false);
  fft(b, false);
  for(int i = 0; i < r; i++){
    a[i] = mul(a[i], b[i]);
  }
  fft(a, true);
  while(!a.empty() and (a.back() == 0))
    a.pop_back();
}

vvl vectors;

void do_fft(int l, int r){
  if(r == l)
    return;
  if(r - l == 1){
    multiply(vectors[l], vectors[r]);
    return;
  }
  int m = (l + r) >> 1;
  do_fft(l, m);
  do_fft(m + 1, r);
  multiply(vectors[l], vectors[m + 1]);
}


//  -----------------------------------------------------------------------------




// BIT
struct FenwickTree{ // Range update, Point Query
  vector<int> bit;
  int n;
  FenwickTree(int n = N){
    this->n = n;
    bit.assign(n + 1, 0);
  }
  int sum(int idx){ // Uses 1 - based indexing
    int ans = 0;
    for(; idx > 0; idx -= (idx & (-idx))){
      ans += bit[idx];
    }
    return ans;
  }
  void add(int idx, int val){
    for(; idx <= n; idx += (idx & (-idx))){
      bit[idx] += val;
    }
  }
  void range_add(int l, int r, int val){
    add(l, val);
    add(r + 1, -val);
  }
};



//  -----------------------------------------------------------------------------
// Mo's Algorithm
// https://cp-algorithms.com/data_structures/sqrt_decomposition.html

void remove(idx);  // TODO: remove value at idx from data structure
void add(idx);     // TODO: add value at idx from data structure
int get_answer();  // TODO: extract the current answer of the data structure

int block_size;

struct Query {
    int l, r, idx;
    bool operator<(Query other) const
    {
        return make_pair(l / block_size, r) <
               make_pair(other.l / block_size, other.r);
    }
};

vector<int> mo_s_algorithm(vector<Query> queries) {
    vector<int> answers(queries.size());
    sort(queries.begin(), queries.end());

    // TODO: initialize data structure

    int cur_l = 0;
    int cur_r = -1;
    // invariant: data structure will always reflect the range [cur_l, cur_r]
    for (Query q : queries) {
        while (cur_l > q.l) {
            cur_l--;
            add(cur_l);
        }
        while (cur_r < q.r) {
            cur_r++;
            add(cur_r);
        }
        while (cur_l < q.l) {
            remove(cur_l);
            cur_l++;
        }
        while (cur_r > q.r) {
            remove(cur_r);
            cur_r--;
        }
        answers[q.idx] = get_answer();
    }
    return answers;
}




//  ---------------------------------------------------------------------------
// Matrix Expo
const ll MAX_N = 2 // Change as per the question

struct Matrix{  // Fibonacci (and related) Matrix

    ll a[MAX_N][MAX_N];
    ll mod = 1e9 + 7;

    void reset(){
        memset(arr,0,sizeof arr);
    }
    void Identity(){
        ll i,j;
        reset();
        for(i = 0;i < MAX_N;i++){
            a[i][i] = 1;
        }
    }

    Matrix operator + (const Matrix &o) const{
        Matrix res;
        ll i,j;
        for(i = 0;i < MAX_N;i++){
            for(j = 0;j < MAX_N;j++){
                res[i][j] = (a[i][j] % mod + o.a[i][j] % mod) % mod;
            }
        }
        return res;
    };

    Matrix operator * (const Matrix &o) const{
        Matrix res;
        ll i,j,k;
        for(i = 0; i < MAX_N; i++){
            for(j = 0; j < MAX_N; j++){
                res.a[i][j] = 0;
                for(k = 0;k < MAX_N;k++){
                    res.a[i][j] = (res.a[i][j] % mod +
                                (a[i][k] % mod * o.a[k][j] % mod) % mod) % mod;
                }
            }
        }
        return res;
    };

    Matrix print (Matrix a){
        ll i,j;
        for(i = 0;i < MAX_N;i++){
            for(j = 0;j < MAX_N;j++){
                cerr << a.a[i][j] << ' ';
            }cerr << '\n';
        }
    }

    Matrix power (Matrix a, ll n){

        Matrix res;
        res.Identity();
        while(n){
            if(n &  1){
                res = res * a;
            }
            a = a * a;
            n /= 2;
        }
        return res;
    }
};
```

```cpp
int main(){

    Matrix m;
    m.reset();

    ll n = 10 //let

    m.a[0][0] = m.a[0][1] = 1;
    m.a[1][0] = 1;

    m = m.power(m, n - 1); // Finds (F_n,F_n-1) in terms of F_0 and F_1 (n = 10
here)
}

//  ---------------------------------------------------------------------------

// LIS
// O(nlogn)

int lis(vector<int> const& a) {
    int n = a.size();
    const int INF = 1e9;
    vector<int> d(n+1, INF);
    d[0] = -INF;

    for (int i = 0; i < n; i++) {
        int j = upper_bound(d.begin(), d.end(), a[i]) - d.begin();
        if (d[j-1] < a[i] && a[i] < d[j])
            d[j] = a[i];
    }

    int ans = 0;
    for (int i = 0; i <= n; i++) {
        if (d[i] < INF)
            ans = i;
    }
    return ans;
}

// O(n^2) with restoration
vector<int> lis(vector<int> const& a) {
    int n = a.size();
    vector<int> d(n, 1), p(n, -1);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < i; j++) {
            if (a[j] < a[i] && d[i] < d[j] + 1) {
                d[i] = d[j] + 1;
                p[i] = j;
            }
        }
    }

    int ans = d[0], pos = 0;
    for (int i = 1; i < n; i++) {
        if (d[i] > ans) {
            ans = d[i];
            pos = i;
        }
    }

    vector<int> subseq;
    while (pos != -1) {
        subseq.push_back(a[pos]);
        pos = p[pos];
    }
    reverse(subseq.begin(), subseq.end());
    return subseq;
}
//  ---------------------------------------------------------------------------
// https://cp-algorithms.com/geometry/lines-intersection.html
// Line Intersection
struct pt {
  double x, y;
};

struct line {
  double a, b, c;
};

const double EPS = 1e-9;

double det(double a, double b, double c, double d) {
  return a*d - b*c;
}

bool intersect(line m, line n, pt & res) {
  double zn = det(m.a, m.b, n.a, n.b);
  if (abs(zn) < EPS)
      return false;
  res.x = -det(m.c, m.b, n.c, n.b) / zn;
  res.y = -det(m.a, m.c, n.a, n.c) / zn;
  return true;
}

bool parallel(line m, line n) {
  return abs(det(m.a, m.b, n.a, n.b)) < EPS;
}

bool equivalent(line m, line n) {
  return abs(det(m.a, m.b, n.a, n.b)) < EPS
      && abs(det(m.a, m.c, n.a, n.c)) < EPS
      && abs(det(m.b, m.c, n.b, n.c)) < EPS;
}

//  ----------------------------------------------------------------------------

// LCA Using Binary Lifting
int n, l;
vector<vector<int>> adj;

int timer;
vector<int> tin, tout;
vector<vector<int>> up;

void dfs(int v, int p)
{
    tin[v] = ++timer;
    up[v][0] = p;
    for (int i = 1; i <= l; ++i)
        up[v][i] = up[up[v][i-1]][i-1];

    for (int u : adj[v]) {
        if (u != p)
            dfs(u, v);
    }

    tout[v] = ++timer;
}

bool is_ancestor(int u, int v)
{
    return tin[u] <= tin[v] && tout[u] >= tout[v];
}

int lca(int u, int v)
{
    if (is_ancestor(u, v))
        return u;
    if (is_ancestor(v, u))
        return v;
    for (int i = l; i >= 0; --i) {
        if (!is_ancestor(up[u][i], v))
            u = up[u][i];
    }
    return up[u][0];
}

void preprocess(int root) {
    tin.resize(n);
    tout.resize(n);
    timer = 0;
    l = ceil(log2(n));
    up.assign(n, vector<int>(l + 1));
    dfs(root, root);
}

//  ----------------------------------------------------------------------------

// Detects cycle in graph
int n;
vector<vector<int>> adj;
vector<char> color;
vector<int> parent;
int cycle_start, cycle_end;

bool dfs(int v) {
    color[v] = 1;
    for (int u : adj[v]) {
        if (color[u] == 0) {
            parent[u] = v;
            if (dfs(u))
                return true;
        } else if (color[u] == 1) {
            cycle_end = v;
            cycle_start = u;
            return true;
        }
    }
    color[v] = 2;
    return false;
}

void find_cycle() {
    color.assign(n, 0);
    parent.assign(n, -1);
    cycle_start = -1;

    for (int v = 0; v < n; v++) {
        if (color[v] == 0 && dfs(v))
            break;
    }

    if (cycle_start == -1) {
        cout << "Acyclic" << endl;
    } else {
        vector<int> cycle;
        cycle.push_back(cycle_start);
        for (int v = cycle_end; v != cycle_start; v = parent[v])
            cycle.push_back(v);
        cycle.push_back(cycle_start);
        reverse(cycle.begin(), cycle.end());

        cout << "Cycle found: ";
        for (int v : cycle)
            cout << v << " ";
        cout << endl;
    }
}

//  ---------------------------------------------------------------------------

// Gauss Elimination
struct Gauss{
    // Remember to declare a[MAX_SIZE][MAX_SIZE], b[MAX_SIZE], Ans[MAX_SIZE] & mod
globally
    // Remember to declare the Modular_operations class before this, Goto ^
    // Goto ^
    // Solution of system of linear equations using Gauss - Jordan method under
modulo `mod`
    // This code finds the coefficients of a polynomial of degree k (set here to
be max 10)
    // This code can also be used to find the solutions to a given SOLE
    // Make sure to change MAX_SIZE as per the question --> Runs in O(MAX_SIZE^3),
so use only when MAX_SIZE ~ 100
    // Goto ^

    ll MAX_SIZE = 10;   // Set it as per the question

    // Use for interactive problems like https://codeforces.com/contest/1155/
problem/E

    ll interactor(ll x){

        ll f[MAX_SIZE + 1] = {1,0,1,0,0,0,0,0,0,0,0};

        ll curr = 1;

        ll ans = 0;

        for(ll i = 0;i <= MAX_SIZE;i++){
            ans = add(ans,mul(f[i],curr));
            curr = mul(curr,x);
        }
        return ans;
    }

    // ---- Routine Algorithm --- //

    void swap_row(ll i,ll j){

        for(ll k = 0;k <= MAX_SIZE;k++){
            swap(a[i][k],a[j][k]);
        }
        swap(b[i],b[j]);
    }

    // --- For Debugging purposes --- //

    void print_mat(){

        for(ll i = 0;i <= MAX_SIZE;i++){
            for(ll j = 0;j <= MAX_SIZE;j++){
                cout << fixed << setfill(' ') << setw(10);
                cout << a[i][j] << ' ';
            }
            cout << " : " << b[i];
            cout << '\n';
        }
    }

    // --- Stores the final solution to the SOLE --- //

    void getans(){
        for(ll i = 0;i <= MAX_SIZE;i++){
```

```cpp
            cout << Ans[i] << ' ';
        }
        cout << '\n';
    }

    // --- Reduce the given matrix to Echelon Form --- //
    // --- Remember, the below subroutine is for SOl of SOLE under a modulo `mod`
        // Don't use the below snippets for the actual answers (the answer
calculated here are modulo `mod`)

    void forward_elimination(){

        for(ll i = 0; i <= MAX_SIZE; i++){
            ll row,val;
            row = i;val = a[row][i];
            for(ll j = i + 1; j <= MAX_SIZE; j++){
                if(a[j][i] > val){
                    val = a[j][i];
                    row = j;
                }
            }
            // cerr << row << '\n';
            if(row != i)
                swap_row(row,i);

            for(ll j = i + 1;j <= MAX_SIZE;j++){
                ll d = divide(a[j][i],a[i][i]);
                for(ll k = i;k <= MAX_SIZE;k++){
                    a[j][k] = sub(a[j][k], mul(d,a[i][k]));
                }
                b[j] = sub(b[j],mul(d,b[i]));
            }
            // cout << "---" << i << "---\n";
            // print_mat();
        }
    }

    void backward_substitution(){

        for(ll i = MAX_SIZE;i >= 0;i--){

            Ans[i] = b[i];

            for(ll j = i + 1;j <= MAX_SIZE; j++){

                Ans[i] = sub(Ans[i],mul(a[i][j],Ans[j]));
            }

            Ans[i] = divide(Ans[i],a[i][i]);
        }
        // cout << "---Final---\n";
        // print_mat();
    }
};

//  ----------------------------------------------------------------------------

// Convex Hull
// https://cp-algorithms.com/geometry/point-in-convex-polygon.html

struct pt{
    long long x, y;
    pt(){}
    pt(long long _x, long long _y):x(_x), y(_y){}
    pt operator+(const pt & p) const { return pt(x + p.x, y + p.y); }
    pt operator-(const pt & p) const { return pt(x - p.x, y - p.y); }
    long long cross(const pt & p) const { return x * p.y - y * p.x; }
    long long dot(const pt & p) const { return x * p.x + y * p.y; }
    long long cross(const pt & a, const pt & b) const { return (a - *this).cross(b
- *this); }
    long long dot(const pt & a, const pt & b) const { return (a - *this).dot(b -
*this); }
    long long sqrLen() const { return this->dot(*this); }
};

bool lexComp(const pt & l, const pt & r){
    return l.x < r.x || (l.x == r.x && l.y < r.y);
}

int sgn(long long val){
    return val > 0 ? 1 : (val == 0 ? 0 : -1);
}

vector<pt> seq;
int n;

bool pointInTriangle(pt a, pt b, pt c, pt point){
    long long s1 = abs(a.cross(b, c));
    long long s2 = abs(point.cross(a, b)) + abs(point.cross(b, c)) + abs
(point.cross(c, a));
    return s1 == s2;
}

void prepare(vector<pt> & points){
    n = points.size();
    int pos = 0;
    for(int i = 1; i < n; i++){
        if(lexComp(points[i], points[pos]))
            pos = i;
    }
    rotate(points.begin(), points.begin() + pos, points.end());

    n--;
    seq.resize(n);
    for(int i = 0; i < n; i++)
        seq[i] = points[i + 1] - points[0];
}

bool pointInConvexPolygon(pt point){
    if(seq[0].cross(point) != 0 && sgn(seq[0].cross(point)) != sgn(seq[0].cross(seq
[n - 1])))
        return false;
    if(seq[n - 1].cross(point) != 0 && sgn(seq[n - 1].cross(point)) != sgn(seq[n -
1].cross(seq[0])))
        return false;

    if(seq[0].cross(point) == 0)
        return seq[0].sqrLen() >= point.sqrLen();

    int l = 0, r = n - 1;
    while(r - l > 1){
        int mid = (l + r)/2;
        int pos = mid;
        if(seq[pos].cross(point) >= 0)l = mid;
        else r = mid;
    }
    int pos = l;
    return pointInTriangle(seq[pos], seq[pos + 1], pt(0, 0), point);
}

//  -------------------------------------------------------------------------
```

```cpp
// Bridges in a graph

vector<vector<int>> adj;
vector<vector<int>> bridges;
vector<bool> visited;
vector<int> tin, low;
int timer;

void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            low[v] = min(low[v], tin[to]);
            if (low[to] > tin[v])
                bridges.push_back({v, to});
        } else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] > tin[v])
                bridges.push_back({v, to});
        }
    }
}

void find_bridges(int n) {
    timer = 0;
    visited.resize(n, false);
    tin.resize(n, -1);
    low.resize(n, -1);
    for (int i = 0; i < n; ++i) {
        if (!visited[i]){
            cout << i << '\n';
            dfs(i);
        }
    }
}

//  ---------------------------------------------------------------------------

// Articulation Points in a graph

vector<bool> visited;
vector<int> tin, low;
int timer;

void dfs(int v, int p = -1) {
    visited[v] = true;
    tin[v] = low[v] = timer++;
    int children=0;
    for (int to : adj[v]) {
        if (to == p) continue;
        if (visited[to]) {
            low[v] = min(low[v], tin[to]);
        } else {
            dfs(to, v);
            low[v] = min(low[v], low[to]);
            if (low[to] >= tin[v] && p!=-1)
                IS_CUTPOINT(v);
            ++children;
        }
    }
    if(p == -1 && children > 1)
        IS_CUTPOINT(v);
}

void find_cutpoints() {
    timer = 0;
    visited.assign(n, false);
    tin.assign(n, -1);
    low.assign(n, -1);
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs (i);
    }
}

//  ----------------------------------------------------------------------------

// Bipartite Check

int n;
vector<vector<int>> adj;

vector<int> side(n, -1);
bool is_bipartite = true;
queue<int> q;
for (int st = 0; st < n; ++st) {
    if (side[st] == -1) {
        q.push(st);
        side[st] = 0;
        while (!q.empty()) {
            int v = q.front();
            q.pop();
            for (int u : adj[v]) {
                if (side[u] == -1) {
                    side[u] = side[v] ^ 1
                    q.push(u);
                } else {
                    is_bipartite &= side[u] != side[v];
                }
            }
        }
    }
}

cout << (is_bipartite ? "YES" : "NO") << endl;

//  ----------------------------------------------------------------------------

// Topological Sort

int n; // number of vertices
vector<vector<int>> adj; // adjacency list of graph
vector<bool> visited;
vector<int> ans;

void dfs(int v) {
    visited[v] = true;
    for (int u : adj[v]) {
        if (!visited[u])
            dfs(u);
    }
    ans.push_back(v);
}

void topological_sort() {
    visited.assign(n, false);
    ans.clear();
    for (int i = 0; i < n; ++i) {
        if (!visited[i])
            dfs(i);
    }
    reverse(ans.begin(), ans.end());
}

//  ----------------------------------------------------------------------------
// FFT (Without Mod)

typedef complex<double> base;

const double PI = acos(-1.0l);
const int N = 8e5+5;
const int Maxb = 19;
const int Maxp = 450;

vector<int> rev;
vector<base> omega;

void calc_rev(int n, int log_n) //Call this before FFT
{
    omega.assign(n, 0);
    rev.assign(n, 0);
    for(int i=0;i<n;i++)
    {
        rev[i]=0;
        for(int j=0;j<log_n;j++)
        {
            if((i>>j)&1)
                rev[i] |= 1<<(log_n-j-1);
        }
    }
}

void fft(vector<base> &A, int n, bool invert)
{
    for(int i=0;i<n;i++)
    {
        if(i<rev[i])
            swap(A[i], A[rev[i]]);
    }
    for(int len=2;len<=n;len<<=1)
    {
        double ang=2*PI/len * (invert?-1:+1);
        int half=(len>>1);

        base curomega(cos(ang), sin(ang));
        omega[0]=base(1, 0);

        for(int i=1;i<half;i++)
            omega[i]=omega[i-1]*curomega;

        for(int i=0;i<n;i+=len)
        {
            base t;
            int pu = i,
                pv = i+half,
                pu_end = i+half,
                pw = 0;
            for(; pu!=pu_end; pu++, pv++, pw++)
            {
                t=A[pv] * omega[pw];
                A[pv] = A[pu] - t;
                A[pu] += t;
            }
        }
    }

    if(invert)
        for(int i=0;i<n;i++)
            A[i]/=n;
}

void multiply(int n, vector<base> &A, vector<base> &B, vector<int> &C)
{
    fft(A, n, false);
    fft(B, n, false);
    for(int i=0;i<n;i++)
        A[i] *= B[i];
    fft(A, n, true);
    for(int i=0;i<n;i++)
    {
        C[i] = (int)(A[i].real() + 0.5);
    }
}


void Solve(int n, vector<int> &coeffA, vector<int> &coeffB, vector<int> &result)
{
    vector<base> A(n), B(n);
    for(int i=0;i<n;i++)
    {
        A[i]=coeffA[i];
        B[i]=0;
    }
    for(int i=0;i<n;i++)
    {
        B[i]=coeffB[i];
    }
    vector<int> C(n);
    multiply(n, A, B, C);
    for(int i=0;i<n;i++)
    {
        int add=C[i];
        result[i]+=add;
    }
}

void do_FFT(vector<int> &A, vector<int> &B, vector<int> &result)
{
    int n=1, bits=0;
    while(n<2*A.size() || n<2*B.size())
        n<<=1, bits++;
    result.assign(n, 0);
    calc_rev(n, bits);
    vector<int> tempA(A.begin(), A.end());
    vector<int> tempB(B.begin(), B.end());
    tempA.resize(n);
    tempB.resize(n);
    Solve(n, tempA, tempB, result);
}

//  ----------------------------------------------------------------------------
```