MOST BASED TEMPLATE EVER

By KawakiMeido

```cpp
/*
    | Centroid Decomposition |
    Desc: Technique for path queries. Takes O(n*log(n)) to build.
    Source: KawakiMeido
    State: Untested lmao
*/

int sz[N];
int vis[N];

int findSize(int u, int p=0){
    sz[u] = 1;
    for (auto v:adj[u]){
        if (vis[v] || v==p) continue;
        sz[u] += findSize(v,u);
    }
    return sz[u];
}

int findCentroid(int u, int n, int p=0){
    for (auto v:adj[u]){
        if (vis[v] || v==p) continue;
        if (sz[v]>n/2) return findCentroid(v,n,u);
    }
    return u;
}

void dfsCentroid(int u, int p, int depth=1){
    for (auto v:adj[u]){
        if (vis[v] || v==p) continue;
        dfsCentroid(v,u,depth+1);
    }
}

void buildCentroid(int s){
    findSize(s);
    int u = findCentroid(s,sz[s]);
    vis[u] = true;

    for (auto v:adj[u]){
        if (vis[v]) continue;
        dfsCentroid(v,u);
    }

    for (auto v:adj[u]){
        if (vis[v]) continue;
        buildCentroid(v);
    }
}
```

```cpp
/*
    | Convex Hull Trick |
    Desc: DP opt for problems involving linear functions.
    Source: KawakiMeido
    State: Untested lmao
*/

struct Line{
    int m,n;

    Line(int _m=0, int _n=0): m(_m), n(_n){};

    int operator()(const int& x) const { return m*x+n;};

    friend ld intersect (Line a, Line b) {
        return (ld)(b.n-a.n)/(ld)(a.m-b.m);
    };
};

struct LineContainer{
    deque<Line> dq;

    void add(Line line){
        while ((int)dq.size()>1 && intersect(dq[dq.size()-1],dq[dq.size()-2]) > intersect(dq[dq.size()-1],line)){
            dq.pop_back();
        }
        dq.push_back(line);
    }

    int getLine(int x){
        int ans = 0, l=1, r=dq.size()-1;
        while (l<=r){
            int mid = (l+r)/2;
            if (intersect(dq[mid],dq[mid-1])<=x){
                ans = mid;
                l = mid+1;
            }
            else r = mid-1;
        }
        return ans;
    }

    int getVal(int x){
        int idx = getLine(x);
        return dq[idx](x);
    }
} CHT;

```

```cpp
/*
    | Combinatorics |
    Desc: Library for BinPow, InvMod, and Binomial Coefficient
    Source: KawakiMeido
    State: Untested lmao
*/

namespace Comb {
    using ll = long long;

    const int MD = 1e9+7;
    const int N = 2e5;
    const int LG = 30;

    int fac[N+1];

    int binPow(int a, int b){
        ll res = 1;
        for (int lg = LG-1; lg >= 0; lg--){
            res = res*res%MD;
            if ((1LL<<lg)&b) res = res*a%MD;
        }
        return res;
    }

    int invMod(int x, int MD){
        return binPow(x,MD-2);
    }

    int nCk(int n, int k){
        return 1LL*fac[n]*invMod(fac[k],MD)%MD*invMod(fac[n-k],MD)%MD;
    }

    struct Init {
        Init() {
            fac[0] = 1;
            for (int i = 1; i <= N; i++){
                fac[i] = (int)(1LL*fac[i-1]*i%MD);
            }
        }
    } _init;
}

```

```cpp
/*
    | Dijkstra |
    Desc: Single-source shortest path in O(n*log(n))
    Source: KawakiMeido
    State: Untested lmao
*/

void dijkstra(int s, int dist[]){
    for (int i=1; i<=n; i++){
        dist[i] = INF;
    }
    priority_queue<pii,vector<pii>,greater<pii>> pq;
    dist[s] = 0;
    pq.push({0,s});
    while (!pq.empty()){
        int u = pq.top().se;
        int d = pq.top().fi;
        pq.pop();

        if (d > dist[u]) continue;

        for (auto in:a[u]){
            int v = in.fi;
            int delta = in.se;
            if (dist[v] > d+delta){
                dist[v] = d+delta;
                pq.push({d+delta,v});
            }
        }
    }
}
```

```cpp
/*
    | Dinitz's Max Flow algorithm |
    Desc: Calculating Max flow in O(V^2*E).
    Source: KawakiMeido
    State: Untested lmao (But that one problem in ECNA practice works sooooo
*/

struct Node{
    int u,v,flow,cap;
    Node(int _u, int _v, int _cap): u(_u), v(_v), cap(_cap){
        flow = 0;
    }
};

int s,t,edgecnt;
vector<Node> edge;
vector<int> adj[N];
int level[N],ptr[N];

void AddEdge(int u, int v, int cap){
    edge.emplace_back(u,v,cap);
    edge.emplace_back(v,u,0);
    adj[u].push_back(edgecnt);
    adj[v].push_back(edgecnt+1);
    edgecnt+=2;
}

bool BFS(){
    queue<int> q;
    memset(level,-1,sizeof(level));
    level[s] = 0;
    q.push(0);

    while (!q.empty()){
        int u = q.front();
        q.pop();

        for (auto id:adj[u]){
            int v = edge[id].v;
            if (level[v] == -1 && edge[id].flow≠edge[id].cap){
                level[v] = level[u]+1;
                q.push(v);
            }
        }
    }

    return (level[t]≠-1);
}

```

```
int DFS(int u, int pushed){
    if (pushed == 0) return 0;
    if (u==t) return pushed;

    int res = 0;

    for (int &pos = ptr[u]; pos<(int)adj[u].size(); pos++){
        int id = adj[u][pos];
        int v = edge[id].v;
        if (level[v] == level[u]+1 && edge[id].flow≠edge[id].cap){
            if ((res = DFS(v,min(pushed,edge[id].cap-edge[id].flow)))){
                edge[id].flow+=res;
                edge[id^1].flow-=res;
                return res;
            }
        }
    }

    return 0;
}

int Dinitz(){
    int max_flow = 0;
    while (BFS()){
        memset(ptr,0,sizeof(ptr));
        int flow;
        while ((flow = DFS(s,INF))){
            max_flow+=flow;
        }
    }
    return max_flow;
}
```

```cpp
/*
    | Divide & Conquer DP Optimization |
    Desc: Optimizing DP transitions in the form of

        dp[i][j] = min(dp[i-1][k-1] + C(k,j)) for (0 ≤ k ≤ j)

    where C(k,j) is a cost function.

    lets define opt(i,j) be the value of k that optimize dp[i][j].
    DnC ONLY APPLIES IF:

        opt(i,j) ≤ opt(i,j+1)

    One case of where this condition holds is when cost function C(k,j) satisfy the Quadrangle Inequality:

        C(a,c) + C(b,d) < C(a,d) + C(b,c) for a ≤ b ≤ c ≤ d. (Note that "<" indicates more optimal)

    Runs in O(n*log(n)*C) with C is time to compute Cost function C(k,j)

    Source: KawakiMeido
    State: Untested lmao
*/

void DnC (int k){
    deque<pair<int,pair<pii,pii>>> dq;
    int lvl = 0;
    dq.push_back({1,{{k,m},{k,m}}});
    while (!dq.empty()){
        auto in = dq.front();
        int curlvl = in.fi;
        int l = in.se.fi.fi;
        int r = in.se.fi.se;
        int optl = in.se.se.fi;
        int optr = in.se.se.se;
        dq.pop_front();

        if (curlvl≠lvl){
            lvl = curlvl;
            BIT.Init(m);
        }

        int mid = (l+r)/2;

        pii best = {0,-INF};

        for (int i = optl; i≤min(mid,optr); i++){
            int sum = dpPrev[i-1] - precalc[mid][i-1] + precalc[mid][mid];
```

```
48
49              if (sum > best.se){
50                  best.fi = i;
51                  best.se = sum;
52              }
53          }
54          dp[mid] = best.second;
55          int opt = best.first;
56
57          if (l⩽mid-1) dq.push_back({lvl+1,{{l,mid-1},{optl,opt}}});
58          if (mid+1⩽r) dq.push_back({lvl+1,{{mid+1,r},{opt,optr}}});
59      }
60  }
```

```cpp
/*
    | DP Digit |
    Desc: A general framework for DP Digit
    States would be DP[pos][k][over][under][start] where:
        - pos: The digit position
        - k: An arbitrary state related to the problem
        - over: Whether the current num is guaranteed to be over lower bound
        - under: Whether the current num is guaranteed to be under lower bound
        - started: Whether the number has started (not full 0s)
    Source: KawakiMeido
    State: Untested lmao
*/

int Call_DP(int pos, bool started, bool over, bool under){
    if (pos == sz){
        //Do something
    }

    if (dp[pos][started][over][under] != -1) return dp[pos][started][over][under];


    for (int i=0; i<10; i++){
        if (!over && i<digl) continue;
        if (!under && i>digr) break;
        //Do something
    }
    return dp[pos][started][over][under] = res;
}
```

```cpp
/*
    | Disjoint Set Union |
    Desc: Maintaining disjoint set in O(n*α(n)).
    Source: KawakiMeido
    State: Untested lmao
*/

//DSU
//

struct DSU{
    int n;
    vector<int> parent;

    void Init(int _n){
        n = _n;
        parent.resize(n,0);
        for (int i=1; i⩽n; i++){
            parent[i] = i;
        }
    }

    int Find(int x){
        return (x == parent[x])? x : parent[x] = Find(parent[x]);
    }

    bool IsSame(int u, int v){
        return (Find(u) == Find(v));
    }

    void Union(int u, int v){
        int x = Find(u);
        int y = Find(v);
        if (x≠y){
            parent[y] = x;
        }
    }
};
```

```cpp
/*
    | Extended Euclidean Algorithm |
    Desc: Find a way to represent GCD in terms of a and b for which a*x + b*y = gcd(a,b)
    Source: KawakiMeido
    State: Untested lmao
*/

pll extend_euclid(ll a, ll b) {
        if (b == 0) { return {1, 0}; }
        pll p = extend_euclid(b, a % b);
        return {p.se, p.fi - a / b * p.se};
}
```

```cpp
/*
    | Fenwick Tree |
    Desc: Point update range query / Range update point query in O(n*log(n))
    Source: KawakiMeido
    State: Untested lmao
*/

struct Fenwick{
    vector<int> BIT;

    Fenwick(int _n=0): n(_n){
        BIT.resize(n+10);
    }

    void Init (int _n, int val=0){
        BIT.resize(n+10,0);
    }

    void update(int idx, int val){
        while (idx<=n){
            BIT[idx]+=val;
            idx+=(idx&(-idx));
        }
    }

    int getPoint(int idx){
        int res = 0;
        while (idx>0){
            res+=BIT[idx];
            idx-=(idx&(-idx));
        }
        return res;
    }

    int getVal(int l, int r){
        return (getPoint(r)-getPoint(l-1));
    }
}
```

```cpp
/*
    | Hashing |
    Desc: Hashing with base B and mod M.
    Source: USACO Guide
    State: Its USACO Guide
*/

class HashedString {
  private:
        // change M and B if you want
        static const long long M = 1e9 + 9; //882517247 //905798389 //854099959
        static const long long B = 9973;

        // pow[i] contains B^i % M
        static vector<long long> pow;

        // p_hash[i] is the hash of the first i characters of the given string
        vector<long long> p_hash;

  public:
        HashedString(const string &s) : p_hash(s.size() + 1) {
                while (pow.size() <= s.size()) { pow.push_back((pow.back() * B) % M); }

                p_hash[0] = 0;
                for (int i = 0; i < s.size(); i++) {
                        p_hash[i + 1] = ((p_hash[i] * B) % M + s[i]) % M;
                }
        }

        long long get_hash(int start, int end) {
                long long raw_val = (p_hash[end + 1] - (p_hash[start] * pow[end - start + 1]));
                return (raw_val % M + M) % M;
        }
};
vector<long long> HashedString::pow = {1};
```

```cpp
/*
    | Heavy Light Decomposition |
    Desc: BEST TECHNIQUE EVER. Path queries in O(n*log^2(n)) with Segment Tree.
    Source: KawakiMeido
    State: Untested lmao
*/

int curpos = 0;
int parent[N], sz[N], depth[N];
int root[N], pos[N];

void dfsHLD(int u, int p=0){
    sz[u] = 1;
    for (auto v:adj[u]){
        if (v==p) continue;
        parent[v] = u;
        depth[v] = depth[u]+1;
        dfsHLD(v,u);
        sz[u] += sz[v];
    }
}

void buildHLD(int u, int r){
    pos[u] = ++curpos;
    root[u] = r;

    int nxt = 0;

    for (auto v:adj[u]){
        if (v==parent[u]) continue;
        if (!nxt || sz[v]>sz[nxt]) nxt = v;
    }

    if (nxt){
        buildHLD(nxt,r);
    }

    for (auto v:adj[u]){
        if (v==parent[u] || v==nxt) continue;
        buildHLD(v,v);
    }
}

void updateHLD(int x){
    int u = x;
    while (root[u]!=1){
        int v = parent[root[u]];

        u = v;
```

```
50      }
51  }
```

```cpp
/*
    | Hopcroft-Karp algorithm |
    Desc: Maximum Bipartite in O(E*sqrt(V))
    Source: KawakiMeido
    State: Untested lmao
*/

int pairX[N],pairY[N],dist[N];
bool visX[N],visY[N];

bool BFS(){
    memset(visX,0,sizeof(visX));
    memset(visY,0,sizeof(visY));
    queue<int> q;
    for (int i=1; i<=n; i++){
        if (pairX[i] == 0){
            dist[i] = 0;
            q.push(i);
        }
        else dist[i] = INF;
    }
    dist[0] = INF;
    while (!q.empty()){
        int x = q.front();
        q.pop();

        visX[x] = true;
        for (auto y:adj[x]){
            int v = pairY[y];
            visY[y] = true;
            if (dist[v]==INF){
                dist[v] = dist[x]+1;
                q.push(v);
            }
        }
    }
    return (dist[0]!=INF);
}

bool DFS(int u){
    if (u == 0) return true;
    for (auto y:adj[u]){
        int v = pairY[y];
        if (dist[v] == dist[u]+1 && DFS(v)){
            pairX[u] = y;
            pairY[y] = u;
            return true;
        }
    }
```

```
    dist[u] = INF;
    return 0;
}

int Hopcroft_Karp(){
    int matching = 0;
    memset(pairX,0,sizeof(pairX));
    memset(pairY,0,sizeof(pairY));
    while (BFS()){
//        cout << dist[0] << endl;
        for (int i=1; i<=n; i++){
            if (pairX[i] == 0 && DFS(i)){
                ++matching;
            }
        }
    }
    return matching;
}
```

```
/*
    | Knuth-Morris-Pratt |
    Desc: Single string matching in O(n)
    Source: KawakiMeido
    State: Untested lmao
*/

pi[0] = 0;
for (int i=1; i<(int)s.size(); i++){
    int cur = i;
    while (cur≠0){
        if (s[i]==s[pi[cur-1]]){
            pi[i] = pi[cur-1]+1;
            break;
        }
        cur = pi[cur-1];
    }
    if (cur == 0){
        if (s[i] == s[0]) pi[i] = 1;
        else pi[i] = 0;
    }
}

```

```cpp
/*
    | Segment Tree - Lazy Propagation |
    Desc: Range Update Range Queries in O(n*log(n)).
    Source: KawakiMeido
    State: Untested lmao
*/

struct LazySegmentTree{
    struct Node{
        int val;
        Node(){
            val = INF;
        }
    };

    int n;
    vector<Node> IT;
    vector<int> lazy;

    void add(int idx, int x){

        //Update logic

    }

    void propagate(int idx){
        add(idx*2,lazy[idx]);
        add(idx*2+1,lazy[idx]);
        lazy[idx] = 0;
    }

    Node comb(Node l, Node r){
        if (l.val == INF) return r;
        if (r.val == INF) return l;

        Node i;

        //Update logic

        return i;
    }

    void build(int idx, int l, int r){
        if (l==r){

            //Update logic

            return;
        }
```

```
50
51          int mid = (l+r)/2;
52          build(idx*2,l,mid);
53          build(idx*2+1,mid+1,r);
54          IT[idx] = comb(IT[idx*2],IT[idx*2+1]);
55      }
56
57      void update(int idx, int l, int r, int x, int y, int val){
58          if (y < l || r < x) return;
59          if (x <= l && r <= y){
60              add(idx,val);
61              return;
62          }
63
64          propagate(idx);
65
66          int mid = (l+r)/2;
67          update(idx*2,l,mid,x,y,val);
68          update(idx*2+1,mid+1,r,x,y,val);
69          IT[idx] = comb(IT[idx*2],IT[idx*2+1]);
70      }
71
72      Node getNode(int idx, int l, int r, int x, int y){
73          if (y < l || r < x) return Node();
74          if (x <= l && r <= y){
75              return IT[idx];
76          }
77
78          propagate(idx);
79
80          int mid = (l+r)/2;
81          return comb(getNode(idx*2,l,mid,x,y),getNode(idx*2+1,mid+1,r,x,y));
82
83      }
84
85      void init(int _n){
86          n = _n;
87          IT.resize(n*4+10, Node());
88          lazy.resize(n*4+10, 0);
89          build(1,1,n);
90      }
91  };
92
```

```cpp
/*
    | Lowest Common Ancestor - Binary Lifting |
    Desc: Finding LCA in O(n*log(n)). Can support additional path computations.
    Source: KawakiMeido
    State: Untested lmao
*/

const int LG_LCA = 18

int up[LG_LCA][N];
int depth[N];

void dfsLCA(int u, int p=0){
    depth[u] = depth[p]+1;
    up[0][u] = p;
    for (int lg=1; lg<LG_LCA; lg++){
        int v = up[lg-1][u];
        up[lg][u] = up[lg-1][v];
    }
    for (auto v:adj[u]){
        if (v==p) continue;
        dfsLCA(v,u);
    }
}

int binLift(int u, int x){
    for (int lg=0; lg<LG_LCA; lg++){
        if ((1<<lg)&x) u = up[lg][u];
    }
    return u;
}

int getDist(int u, int v){
    if (depth[u]>depth[v]) swap(u,v);
    v = binLift(v,depth[v]-depth[u]);
    if (u==v) return u;
    for (int lg=LG_LCA-1; lg>=0; lg--){
        if (up[lg][u]!=up[lg][v]){
            u = up[lg][u];
            v = up[lg][v];
        }
    }
    return up[0][u];
}
```

```cpp
/*
    | Matrix Template |
    Desc: Template for Matrix operations.
    Source: KawakiMeido
    State: Untested but new code
*/

const int LG = 30;

struct Matrix{
    int n,m;
    vector<vector<int>> val;

    Matrix(int _n=0, int _m=0){
        n = _n;
        m = _m;
        val.resize(n,vector<int>(m,0));
    }

    friend Matrix mul (Matrix a, Matrix b){
        if (a.m≠b.n) return Matrix();
        Matrix res(a.n,b.m);
        for (int i=0; i<a.n; i++){
            for (int j=0; j<b.m; j++){
                for (int k=0; k<a.m; k++){
                    res.val[i][j] = (res.val[i][j] + a.val[i][k]*b.val[k][j]%MD)%MD;
                }
            }
        }
        return res;
    }
};

Matrix matrixExp(Matrix a, int b){
    Matrix res(a.n,a.n);
    for (int i=0; i<a.n; i++){
        res.val[i][i] = 1;
    }
    for (int lg=LG-1; lg≥0; lg--){
        res = mul(res,res);
        if ((1LL<<lg)&b) res = mul(res,a);
    }
    return res;
}
```

```cpp
/*
    | Monotone Chain |
    Finding Convex Hull in O(n*log(n))
    Source: USACO Guide
    State: Idk its from USACO Guide
*/

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

using pii = pair<int, int>;

vector<pii> points;
vector<pii> hull;

// cross product, the signed area of these there points
int area(pii O, pii P, pii Q) {
        return (P.first - O.first) * (Q.second - O.second) -
               (P.second - O.second) * (Q.first - O.first);
}

void monotone_chain() {
        // sort with respect to the x and y coordinates
        sort(points.begin(), points.end());
        // distinct the points
        points.erase(unique(points.begin(), points.end()), points.end());
        int n = points.size();

        // 1 or 2 points are always in the convex hull
        if (n < 3) {
                hull = points;
                return;
        }

        // lower hull
        for (int i = 0; i < n; i++) {
                // if with the new point points[i], a right turn will be formed,
                // then we remove the last point in the hull and test further
                while (hull.size() > 1 &&
                       area(hull[hull.size() - 2], hull.back(), points[i]) <= 0)

                        hull.pop_back();
                // otherwise, add the point to the hull
                hull.push_back(points[i]);
        }

        // upper hull, following the same logic as the lower hull
        auto lower_hull_length = hull.size();
        for (int i = n - 2; i >= 0; i--) {
```

```cpp
                // we can only remove a point if there are still points left in the
                // upper hull
                while (hull.size() > lower_hull_length &&
                       area(hull[hull.size() - 2], hull.back(), points[i]) <= 0)
                        hull.pop_back();
                hull.push_back(points[i]);
        }
        // delete point[0] that has been added twice
        hull.pop_back();
}

int main() {
        cin.tie(0)->sync_with_stdio(false);

        int n;
        cin >> n;
        while (n != 0) {
                points.assign(n, {});
                hull = {};
                for (auto &p : points) cin >> p.first >> p.second;
                monotone_chain();

                cout << hull.size() << "\n";
                for (auto &p : hull) cout << p.first << " " << p.second << "\n";

                cin >> n;
        }

        return 0;
}
```

```cpp
/*
    | Point Class |
    Desc: A generic point class with some helper funcs
    Source: KawakiMeido
    State: Untested and VERY buggy lmao
*/

#define X real()
#define Y imag()

template <typename T>
class Point {
public:
    static constexpr double EPS = 1e-6;

    std::complex<T> p;

    // Contructors
    Point(T x = 0, T y = 0) : p(x, y) {}
    explicit Point(const std::complex<T>& val) : p(val) {}

    // Accessors
//    T real() { return p.real(); }
    T real() const { return p.real(); }
//    T imag() { return p.imag(); }
    T imag() const { return p.imag(); }

    void setX(int x) {
        p.real(x);
    }

    void setY(int y) {
        p.imag(y);
    }

    // Comparisons
    bool operator==(const Point& other) const {
        if constexpr (std::is_floating_point_v<T>) {
            return (std::abs(p.real() - other.p.real()) < EPS) &&
                   (std::abs(p.imag() - other.p.imag()) < EPS);
        } else {
            return p == other.p;
        }
    }
    bool operator≠(const Point& other) const { return !(*this == other); }

    // Arithmetics
    Point& operator+=(const Point& other) { p += other.p; return *this; }
    Point& operator-=(const Point& other) { p -= other.p; return *this; }
```

```
50      friend Point operator+(Point a, const Point& b) { a += b; return a; }
51      friend Point operator-(Point a, const Point& b) { a -= b; return a; }
52
53      // Helper Functions
54      friend T dot(const Point& a, const Point& b)   { return (std::conj(a.p) *
    b.p).real(); }
55      friend T cross(const Point& a, const Point& b) { return (std::conj(a.p) *
    b.p).imag(); }
56      friend T sqdist(const Point& a, const Point& b){ return std::norm(a.p -
    b.p); }
57      friend T dist(const Point& a, const Point& b)  { return std::abs(a.p - b.p);
    }
58      friend long double angle(const Point& a, const Point& b) { return
    std::arg(b.p - a.p); }
59      friend long double slope(const Point& a, const Point& b) { return
    std::tan(std::arg(a.p - b.p)); }
60  };
```

```cpp
/*
    | Segment Tree |
    Desc: Classic segment tree. Point Update Range Queries in O(n*log(n)).
    Source: KawakiMeido
    State: Untested lmao
*/

struct SegmentTree{
    struct Node{
        int val;
        Node(){
            val = INF;
        }
    };

    int n;
    vector<Node> IT;

    Node comb(Node l, Node r){
        if (l.val == INF) return r;
        if (r.val == INF) return l;

        Node i;
        //Update logic

        return i;
    }

    void build(int idx, int l, int r){
        if (l==r){
            //Update logic
            return;
        }

        int mid = (l+r)/2;
        build(idx*2,l,mid);
        build(idx*2+1,mid+1,r);
        IT[idx] = comb(IT[idx*2],IT[idx*2+1]);
    }

    void update(int idx, int l, int r, int x, Node val){
        if (r < x || x < l) return;
        if (l==r){
            //Update logic
            return;
        }
        int mid = (l+r)/2;
        update(idx*2,l,mid,x,val);
        update(idx*2+1,mid+1,r,x,val);
```

```
50          IT[idx] = comb(IT[idx*2],IT[idx*2+1]);
51      }
52
53      Node getNode(int idx, int l, int r, int x, int y){
54          if (y < l || r < x) return Node();
55          if (x <= l && r <= y){
56              return IT[idx];
57          }
58
59          int mid = (l+r)/2;
60          return comb(getNode(idx*2,l,mid,x,y),getNode(idx*2+1,mid+1,r,x,y));
61
62      }
63
64      void init(int _n){
65          n = _n;
66          IT.resize(n*4+10, Node());
67          build(1,1,n);
68      }
69  };
```

```cpp
/*
    | Sieve of Eratosthenes |
    Desc: Get all primes from 1 to MXP in O(n*log(log(n)))
    Source: KawakiMeido
    State: Untested lmao
*/
const int MXP = 1e6;

vector<int> primes;

void Sieve(){
    bitset<MXP+1> bs;
    bs.set();
    bs[0] = bs[1] = 0;
    for (int i=2; i*i<=MXP; i++){
        if (!bs[i]) continue;
        for (int j=i*i; j<=MXP; j+=i){
            bs[j]=0;
        }
    }

    for (int i=1; i<=MXP; i++){
        if (bs[i]) primes.push_back(i);
    }
}
```

```
/*
    | SOS DP |
    Desc: Compute sum of all subset of a mask in O(n*2^n).
    Source: KawakiMeido
    State: Tested
*/

for (int i=0; i<LG; i++){
    for (int mask=0; mask<(1<<LG); mask++){
        if ((mask&(1<<i))){
            SOS[mask] = SOS[mask]+SOS[mask^(1<<i)];
        }
    }
}
```

```cpp
/*
    | Tarjan |
    Desc: Algorithm for finding Strongly Connected Components
    Source: CP2
    State: Probably works but idk
*/

vi dfs_num, dfs_low, S, visited;

void tarjanSCC(int u) {
    dfs_low[u] = dfs_num[u] = dfsNumberCounter++; // dfs_low[u] <= dfs_num[u]
    S.push_back(u); // stores u in a vector based on order of visitation
    visited[u] = 1;
    for (int j = 0; j < (int)AdjList[u].size(); j++) {
        ii v = AdjList[u][j];
        if (dfs_num[v.first] == DFS_WHITE)
            tarjanSCC(v.first);
        if (visited[v.first]) // condition for update
            dfs_low[u] = min(dfs_low[u], dfs_low[v.first]);
    }
    if (dfs_low[u] == dfs_num[u]) { // if this is a root (start) of an SCC
        printf("SCC %d:", ++numSCC); // this part is done after recursion
        while (1) {
            int v = S.back(); S.pop_back(); visited[v] = 0;
            printf(" %d", v);
            if (u == v) break;
        }
        printf("\n");
    }
}
```

```cpp
/*
    | Trie |
    Desc: Multiple string matching in O(max(s.size())) for all operations
    Source: KawakiMeido
    State: VERY Untested and old code lmao
*/

struct Trie{

    struct Node{
        Node* child[2];
        int cnt;
        int l,r;
        Node(){
            child[0] = child[1] = NULL;
            cnt = 0;
        }
    };

    Node* Root;
    int cnt;
    Trie(){
        Root = new Node();
        cnt = 0;
    }

    void Init(){
        clr(Root);
    }
    void clr(Node* cur){
        if (cur->child[0]!=NULL){
            clr(cur->child[0]);
            cur->child[0]=NULL;
        }
        if (cur->child[1]!=NULL){
            clr(cur->child[1]);
            cur->child[1]=NULL;
        }
        if (cur!=Root) delete cur;
    }

    void Add(int x, int pos){
        Node* cur = Root;
        for (int i=29; i>=0; i--){
            int idx = ((x>>i)&1);
            if (cur->child[idx] == NULL) cur->child[idx] = new Node();
            cur = cur->child[idx];
            cur->cnt++;
        }
```

```
    }
    int Get(int x, Node* cur, int lg){
        int res = 0;
        if (cur->cnt <= 0) return INF;
        for (int i=lg; i >= 0; i--){
            int idx = ((x>>i)&1);
            if (cur->child[idx] == NULL){
                res = res+(1<<i);
                cur = cur->child[(idx+1)%2];
            }
            else{
                cur = cur->child[idx];
            }
        }
        return res;
    }
};

Trie TR;
```

```cpp
/*She smiles, but nothing behind it feels real. The neon glow wraps around her like armor vibrant, untouchable, cold. Once, maybe, there was warmth in her gestures… but now it’s rehearsed. Perfectly practiced detachment. Her wave is polite, her wink playful, yet there’s an eerie hollowness like a ghost who forgot what it meant to feel. She doesn’t break down. She doesn’t react. She simply exists flawless, empty, and free. Because having zero feelings means never being hurt again.*/
#include <bits/stdc++.h>

#define TEXT ""

using namespace std;

#define pb push_back
#define endl "\n"
#define all(x) (x).begin(),(x).end()
#define lb lower_bound
#define ub upper_bound
#define fi first
#define se second

typedef int int2;
#define int long long

typedef long long ll;
typedef long double ld;
typedef pair<int, int> pii;
typedef pair<ll,ll> pll;
typedef pair<double,double> pdd;

mt19937_64 rd(chrono::high_resolution_clock::now().time_since_epoch().count());

const int N = 2e5+10;
const int INF = 1e9+7;
const int MD = 1e9+7; //998244353;
const long long LLINF = 1e18+3;

//Starts here

int n;

void solve(){

}

/*Driver Code*/
signed main(){
    cin.tie(0) -> sync_with_stdio(0);
    if (fopen(TEXT".inp","r")){
```

```cpp
        freopen(TEXT".inp","r",stdin);
        freopen(TEXT".out","w",stdout);
    }

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

    return 0;
}
```

```cpp
/*Author: KawakiMeido*/
#include <bits/stdc++.h>
#define pb push_back
#define endl "\n"
#define ll long long
#define all(x) (x).begin(),(x).end()
#define pii pair<int,int>
#define fi first
#define se second

#define NAME ""

using namespace std;

/*Constants*/
const int N = 2e5+10;
const int INF = 1e9+7;
const long long LLINF = 1e18+3;

/*Global Variables*/
int n;
mt19937_64 mt(chrono::high_resolution_clock::now().time_since_epoch().count());

ll rd(ll l, ll r){
    return uniform_int_distribution<ll> (l,r) (mt);
}

void Gen(){
    ofstream cout(NAME".inp");

    cout.close();
}

/*Solution*/
void Solve(){
    Gen();

    system(NAME ".exe");
    system(NAME "_BRUTE.exe");
    if (system("fc " NAME ".out " NAME ".ans")){
        cerr << "WA" << endl;
        exit(0);
    }
}

/*Driver Code*/
signed main(){
    ios_base::sync_with_stdio(0);
    cin.tie(0);
```

```
50
51      srand(time(NULL));
52      int TEST=10;
53      for (int testid = 1; testid⩽TEST; testid++){
54          Solve();
55      }
56      Solve();
57
58      return 0;
59  }
```