Molinos

Autor del problema: Félix Moreno Peñarrubia (Miembro del comité organizador)

Notemos que lo que nos pide el enunciado es ordenar los puntos por el mínimo ángulo que hacen con la vertical y con otro de los puntos. Por tanto, una solución $O(n^2)$ es por cada punto, calcular el ángulo con todos los otros puntos y ordenar. Veremos cómo hacer una solución que calculará estos ángulos mínimos en $O(n \log n)$.

Ordenamos los puntos por coordenada x. Usaremos el siguiente lema: consideremos tres puntos A, B, C tales que A tenga menor coordenada x que B y B tenga menor coordenada x que C. Se puede comprobar que si B está por debajo de la recta AC, entonces todo punto con mayor coordenada x que C hace un ángulo menor con A o con C que con B.

En la solución vamos procesando los puntos de izquierda a derecha, y vamos manteniendo algunos de los puntos en una pila. Supongamos que actualmente estamos procesando el punto C, y sean B, A el punto en la cima de la pila y el que está justo por debajo respectivamente. Hacemos lo siguiente: mientras B esté por debajo de la recta AC, eliminamos B de la pila (y ahora A pasa a ser B y el nuevo A es el siguiente en la pila). Una vez tengamos que B está por encima de la recta AC o que sólo queda un punto en la pila, el ángulo mínimo de C será el que haga con B. Esto funciona porque por el lema anterior cada vez que eliminamos un B estamos descartando un punto que ya no es candidato a formar ángulo mínimo con los siguientes puntos y por cómo nos quedan los puntos en la pila vemos que el que formará ángulo mínimo será el que esté en la cima.

En la implementación, hay que tener un poco de cuidado para resolver empates como se indica en el enunciado. También conviene usar aritmética entera y no coma flotante para evitar errores de precisión, aunque los casos de prueba no estaban puestos para causar errores de precisión.

Solución en C++

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

using ll = long long;
using pt = pair<ll,ll>;
#define x first
#define y second

int sign(ll x) {
	if (x < 0) return -1;
	if (x > 0) return 1;
	return 0;
}

int orientation(pt a, pt b, pt c) {
	ll v = a.x*(b.y-c.y)+b.x*(c.y-a.y)+c.x*(a.y-b.y);
```

```
        return sign(v); // +1 -> sentido antihorario
}

bool cw(pt a, pt b, pt c, bool include_collinear) {
        int o = orientation(a, b, c);
        return o < 0 || (include_collinear && o == 0);
}
bool collinear(pt a, pt b, pt c) { return orientation(a, b, c) == 0; }

struct Fraction {
        ll num, den;
        Fraction(ll num_, ll den_) { num = num_; den = den_; }
};

ll sign(Fraction a) {
        return sign(a.num) * sign(a.den);
}

bool operator<(Fraction a, Fraction b) {
        if (sign(a) == sign(b)) {
        if (sign(a) == 1)
                return abs(a.num * b.den) < abs(b.num * a.den);
        else if (sign(a) == -1)
                return abs(a.num * b.den) > abs(b.num * a.den);
        else
                return false;
        }
        return sign(a) < sign(b);
}

int main() {
        ios::sync_with_stdio(0);
        cin.tie(0);

        int n;
        cin >> n;
        vector<pt> pts(n);
        for (int i = 0; i < n; ++i) {
        cin >> pts[i].x >> pts[i].y;
        }

        sort(pts.begin(), pts.end(), [](pt a, pt b) {
        if (a.x != b.x) return a.x < b.x;
        return a.y < b.y;
        });

        vector<pt> buffer;
        for (int i = 0; i < n; ++i) {
        if (i+1 < n and pts[i].x == pts[i+1].x)
                cout << pts[i].x << ' ' << pts[i].y << '\n';
        else
                buffer.push_back(pts[i]);
        }
```

```
        pts = buffer;
        n = pts.size();

        vector<tuple<int,Fraction,pt>> order;
        vector<pt> st;
        for (pt p : pts) {
        while (st.size() >= 2 and !cw(st[st.size()-2], st.back(), p, true)) {
                st.pop_back();
        }
        if (not st.empty()) {
                ll dy = p.y - st.back().y;
                ll dx = p.x - st.back().x;
                if (dy < 0)
                order.push_back({-1, Fraction(dx, -dy), p});
                else if (dy > 0)
                order.push_back({1, Fraction(-dx, dy), p});
                else
                order.push_back({0, Fraction(0, 1), p});
        }
        st.push_back(p);
        }

        sort(order.begin(), order.end());

        for (auto t : order) {

        cout << get<2>(t).x << ' ' << get<2>(t).y << '\n';
        }
}
```

Autor de la solución: Izan Beltran Ferreiro (Miembro del comité organizador)