```
{
 "cmd":["bash", "-c", "g++ -std=c++14 '${file}' -o compile && ./compile <input.txt >output.txt"],
 "working_dir": "${file_path}"
}


#include<bits/stdc++.h>
#define PI acos(-1.0)
using namespace std;
struct PT
    {
        double x, y;
        PT() {}
        PT(double x, double y) : x(x), y(y) {}
        PT(const PT &p) : x(p.x), y(p.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);
        }
        PT operator * (double c)     const
        {
            return PT(x*c,   y*c  );
        }
        PT operator / (double c)     const
        {
            return PT(x/c,   y/c  );
        }
        bool operator < (const PT A) const
        {
            if (x == A.x) return y < A.y;
            return x < A.x;
        }
    };
    double dot(PT p, PT q)
    {
        return p.x*q.x+p.y*q.y;
```

```
}
double dist2(PT p, PT q)
{
    return dot(p-q,p-q);
}
double distPoint(PT p, PT q)
{
    return sqrt(dot(p-q,p-q));
}
double cross(PT p, PT q)
{
    return p.x*q.y-p.y*q.x;
}
double cross(const PT &O, const PT &A, const PT &B)
{
    return (A.x - O.x) * (B.y - O.y) - (A.y - O.y) * (B.x - O.x);
}
/// Returns a list of points on the convex hull in counter-clockwise order.
/// Note: the last point in the returned list is the same as the first one.
vector<PT> convex_hull(vector<PT> P)
{
    size_t n = P.size(), k = 0;
    if (n <= 3) return P;
    vector<PT> H(2*n);

    // Sort points lexicographically
    sort(P.begin(), P.end());

    // Build lower hull
    for (size_t i = 0; i < n; ++i) {
        while (k >= 2 && cross(H[k-2], H[k-1], P[i]) <= 0) k--;
        H[k++] = P[i];
    }

    // Build upper hull
    for (size_t i = n-1, t = k+1; i > 0; --i) {
        while (k >= t && cross(H[k-2], H[k-1], P[i-1]) <= 0) k--;
        H[k++] = P[i-1];
    }

    H.resize(k-1);
    return H;
}
double Angle(PT a, PT b, PT c)
{
    double dot_product = dot(b - a, c - a);
    double d1 = distPoint(a, b);
    double d2 = distPoint(a, c);
    double cur = dot_product / (d1 * d2);
    double degree = (acos(cur) * 180.0) / PI;
    return degree;
}
int main()
{
    int t, cas = 1;
    scanf("%d",&t);
    while(t--)
    {
        int n;
        scanf("%d",&n);
        vector<PT> points;
        for(int i = 1; i<=n; i++)
        {
            PT p;
            scanf("%lf %lf",&p.x, &p.y);
            points.push_back(p);
        }
        if(n < 3)
        {
            printf("Case %d: 0\n",cas++);
            continue;
        }
        double angle = 180.0;
        vector<PT> vt = convex_hull(points);
        if(vt.size() < 3)
        {
            printf("Case %d: 0\n",cas++);
            continue;
        }
        angle = min(angle, Angle(vt[0], vt[vt.size() - 1], vt[1]));
        angle = min(angle, Angle(vt[vt.size() - 1], vt[vt.size() - 2], vt[0]));
        for(int i = 1; i < vt.size() - 1; i++)
        {
            angle = min(angle, Angle(vt[i], vt[i - 1], vt[i + 1]));
        }
        if(angle == 180.0) angle = 0.0;
        printf("Case %d: %0.6lf\n",cas++, angle);
    }
    return 0;
}
```

RECTANGLE UNION

```
  #define INF 500000000
#define maxN 30010
typedef pair<int, int> ii;
typedef vector<ii> vii;
typedef vector<int> vi;

int read(int &n){return scanf(" %d ",&n);}
int read(int64 &n){return scanf(" %lld ",&n);}
int read(uint64 &n){return scanf(" %llu ",&n);}
int read(double &n){return scanf(" %lf ",&n);}
int read(char *c){return scanf(" %s ",c);}

struct Edge {
    bool open;
    int x, yMin, yMax;
    Edge(int x, int y1, int y2, bool op) {
        this->x = x;
        yMin = y1, yMax = y2;
        open = op;
    }
    bool operator < (const Edge &e) const {
        return (x < e.x);
    }
};
int n, m, h[maxN << 1];
int sum[maxN << 5], counter[maxN << 5];
vector<Edge> edges;

void update(int p, int l, int r, int yMin, int yMax, bool open) {
    if (h[r] < yMin || yMax < h[l]) return;
    int c = p << 1, mid = (l + r) >> 1;
    if (yMin <= h[l] && h[r] <= yMax) {
        counter[p] += open ? 1 : -1;
        if (counter[p]) sum[p] = h[r] - h[l];
        else sum[p] = sum[c] + sum[c + 1];
        return;
    }
    if (l + 1 >= r) return;
    update(c, l, mid, yMin, yMax, open);
    update(c + 1, mid, r, yMin, yMax, open);
    if (counter[p]) sum[p] = h[r] - h[l];
    else sum[p] = sum[c] + sum[c + 1];
}

int64 solve() {
    // process height
    sort(h + 1, h + m + 1);
    int k = 1;
    FOR (i, 2, m) if (h[i] != h[k])
        h[++k] = h[i];
    m = k;
    // init tree
    for (int i = 0, lm = maxN << 4; i < lm; i++)
        sum[i] = 0, counter[i] = 0;
    // solve
    int64 area = 0LL;
    sort(all(edges));
    update(1, 1, m, edges[0].yMin, edges[0].yMax, edges[0].open);
    for (int i = 1; i < edges.size(); i++) {
        area += sum[1] * (int64)(edges[i].x - edges[i - 1].x);
        update(1, 1, m, edges[i].yMin, edges[i].yMax, edges[i].open);
    }
    return area;
}

int main() {
    #ifndef ONLINE_JUDGE
        inpFile("test.inp"); //outFile("test.out");
    #endif
    int caseNo, cases = 0, x1, y1, x2, y2;
    read(caseNo);
    while (caseNo--) {
        read(n);
        edges.clear();
        m = 0;
        FOR (i, 1, n) {
            scanf("%d %d %d %d", &x1, &y1, &x2, &y2);
            edges.pb(Edge(x1, y1, y2, true));
            edges.pb(Edge(x2, y1, y2, false));
            h[++m] = y1;
            h[++m] = y2;
        }
        printf("Case %d: %lld\n", ++cases, solve());
    }
    return 0;
}
```