**//Geometry**

**Area of sector of a circle:**

|  |  |
| --- | --- |
| http://www.regentsprep.org/regents/math/geometry/gp16/RefAre5.gif where *n* is the number of degrees in the central angle of the sector. | http://www.regentsprep.org/regents/math/geometry/gp16/RefAre10.gif where CS is the arc length of the sector. |

**Area of segment of a circle:**

http://www.mathopenref.com/images/segmentarea/segmentarea.gif, where C is the central angle in DEGREES

**Triangle Area**

returns double of area of triangle

int triArea2(const point &a, const point &b, const point &c) {

    return (a.x\*(b.y-c.y) + b.x\*(c.y-a.y) + c.x\*(a.y-b.y));

}

**Points in convex polygon**

C[] array of points of convex polygon in ccw order,

nc number of points in C, p target points.

returns true if p is inside C (including edge) or false otherwise.

complexity O(lg n)

inline bool inConvexPoly(point \*C, int nc, const point &p) {

    int st = 1, en = nc - 1, mid;

    while(en - st > 1) {

        mid = (st + en)>>1;

        if(triArea2(C[0], C[mid], p) < 0) en = mid;

        else st = mid;

    }

    if(triArea2(C[0], C[st], p) < 0) return false;

    if(triArea2(C[st], C[en], p) < 0) return false;

    if(triArea2(C[en], C[0], p) < 0) return false;

    return true;

}

**Circle intersection area**

This code assumes the circle center and radius to be integer.

Change this when necessary.

\*/

inline double commonArea(const Circle &a, const Circle &b) {

    int dsq = sqDist(a.c, b.c);

    double d = sqrt((double)dsq);

    if(sq(a.r + b.r) <= dsq) return 0;

    if(a.r >= b.r && sq(a.r-b.r) >= dsq) return pi \* b.r \* b.r;

    if(a.r <= b.r && sq(b.r-a.r) >= dsq) return pi \* a.r \* a.r;

    double angleA = 2.0 \* acos((a.r \* a.r + dsq - b.r \* b.r) / (2.0 \* a.r \* d));

    double angleB = 2.0 \* acos((b.r \* b.r + dsq - a.r \* a.r) / (2.0 \* b.r \* d));

    return 0.5 \* (a.r \* a.r \* (angleA - sin(angleA)) + b.r \* b.r \* (angleB - sin(angleB)));

}

**Closest pair**

closestPair(Point \*X, Point \*Y, int n);

X contains the points sorted by x co-ordinate,

Y contains the points sorted by y co-ordinate,

One additional item in Point structure is needed, the original index.

typedef long long i64;

typedef struct { int x, y, i; } Point;

int flag[MAX];

inline i64 closestPair(Point \*X, Point \*Y, int n) {

    if(n == 1) return INF;

    if(n == 2) return sqdist(X[0], X[1]);

    int i, j, k, n1, n2, ns, m = n >> 1;

    Point Xm = X[m-1], \*XL, \*XR, \*YL, \*YR, \*YS;

    i64 lt, rt, dd, tmp;

    XL = new Point[m], YL = new Point[m];

    XR = new Point[m+1], YR = new Point[m+1];

    YS = new Point[n];

    for(i = 0; i < m; i++) XL[i] = X[i], flag[X[i].i] = 0;

    for(; i < n; i++) XR[i - m] = X[i], flag[X[i].i] = 1;

    for(i = n2 = n1 = 0; i < n; i++) {

        if(!flag[Y[i].i]) YL[n1++] = Y[i];

        else YR[n2++] = Y[i];

    }

    lt = closestPair(XL, YL, n1);

    rt = closestPair(XR, YR, n2);

    dd = min(lt, rt);

    for(i = ns = 0; i < n; i++)

        if(sq(Y[i].x - Xm.x) < dd)

            YS[ns++] = Y[i];

    for(j = 0; j < ns; j++)

        for(k = j + 1; k < ns && sq(YS[k].y - YS[j].y) < dd; k++)

            dd = min(dd, sqdist(YS[j], YS[k]));

    delete[] XL; delete[] XR;

    delete[] YL; delete[] YR;

    delete[] YS;

    return dd;

}

**Polygon Area (2D)**

P[] holds the points, must be either in cw or ccw,

function returns double of the area.

int dArea(int np) {

    int area = 0;

    for(int i = 0; i < np; i++)

        area += p[i].x\*p[i+1].y - p[i].y\*p[i+1].x;

    return abs(area);

}

**Segment Intersection (finite)**

struct Point

{

int x, y;

};

bool onSegment(Point p, Point q, Point r)

{

if (q.x <= max(p.x, r.x) && q.x >= min(p.x, r.x) &&

q.y <= max(p.y, r.y) && q.y >= min(p.y, r.y))

return true;

return false;

}

int orientation(Point p, Point q, Point r)

{

int val = (q.y - p.y) \* (r.x - q.x) - (q.x - p.x) \* (r.y - q.y);

if (val == 0) return 0; // colinear

return (val > 0)? 1: 2; // clock or counterclock wise

}

bool doIntersect(Point p1, Point q1, Point p2, Point q2)

{

int o1 = orientation(p1, q1, p2);

int o2 = orientation(p1, q1, q2);

int o3 = orientation(p2, q2, p1);

int o4 = orientation(p2, q2, q1);

if (o1 != o2 && o3 != o4) return true;

if (o1 == 0 && onSegment(p1, p2, q1)) return true;

if (o2 == 0 && onSegment(p1, q2, q1)) return true;

if (o3 == 0 && onSegment(p2, p1, q2)) return true;

if (o4 == 0 && onSegment(p2, q1, q2)) return true;

return false;

}

**lower and upper bound**

lower bound = first element in the range which does not less than val.

Upper bound = first element in the range which greater than val

Let, v={10 10 10 20 20 20 30 30} after sort

vector<int>::iterator low,up;

low= lower\_bound (v.begin(), v.end(), 20); ^

up= upper\_bound (v.begin(), v.end(), 20); ^

cout << (low- v.begin()) ; //output 3

cout << (up - v.begin()) ; //output 6

**Arithmatic Progression:**

nth term = a + (n−1)d //a=first term, d=difference

sum of first n terms = n/2 {2a+(n-1)d}

**Geometric Progression:**

**nth term = ar^(n-1)**

Sum**sum of first n terms: (finite & r>1)**

Example**finite sum =**