P THINK FAST S

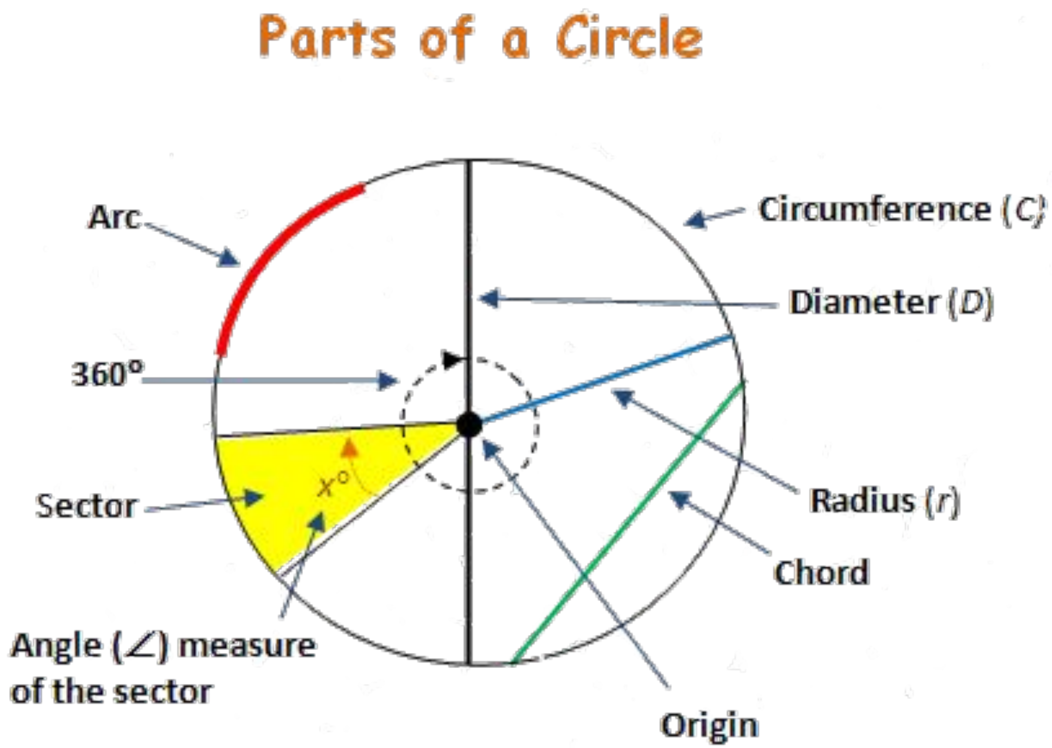
Competitive Programming

From Problem 2 Solution in O(1)

Computational Geometry
Circles

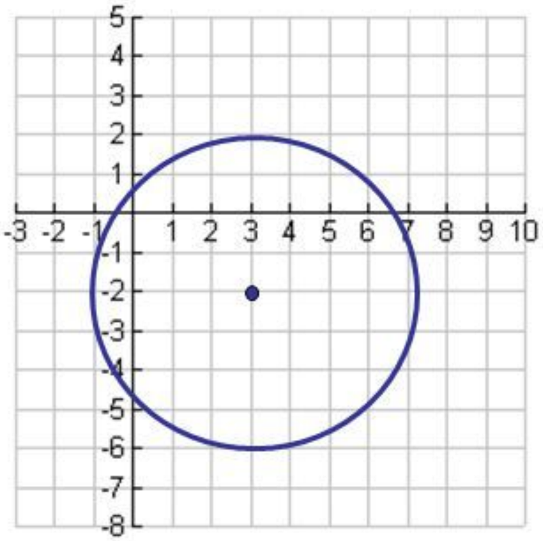
Mostafa Saad Ibrahim
PhD Student @ Simon Fraser University

Circles



Src: http://ssepkowitz.pbworks.com/f/1241790691/SAT_Geometry_Circles1.png

Circles



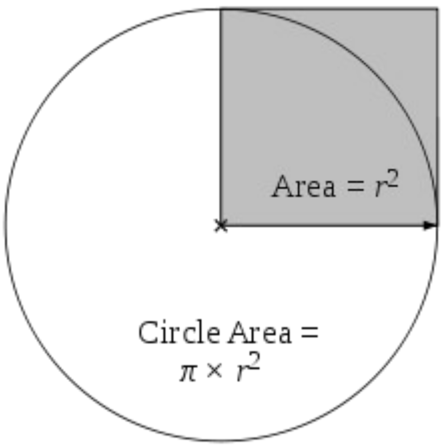
$$(x-h)^2+(y-k)^2=r^2$$

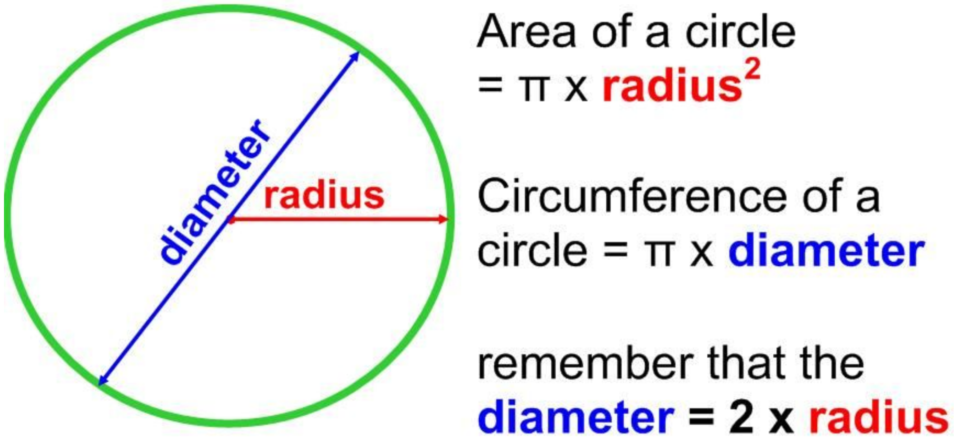
$$(x-3)^2 + (y-(-2))^2 = 4^2$$

$$(x-3)^2 + (y+2)^2 = 16$$

Src: http://images.slideplayer.com/18/6070989/slides/slide_4.jpg

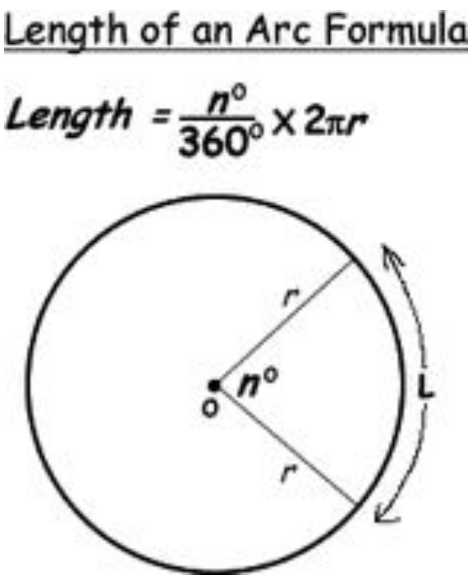
Circles

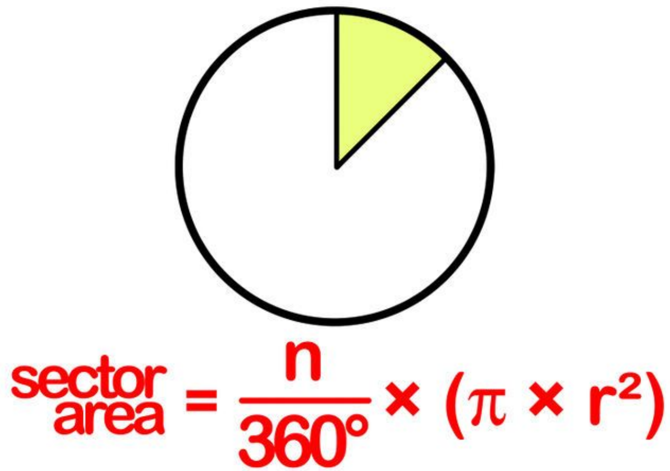


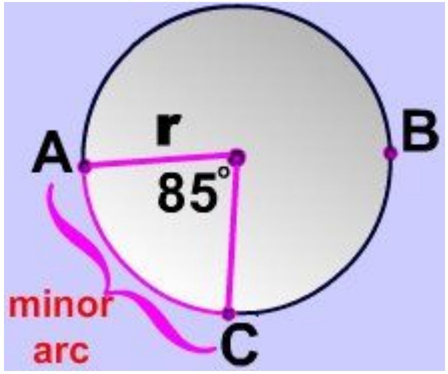


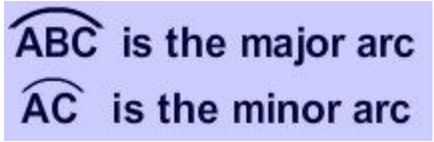
Src: http://img1.wikia.nocookie.net/__cb20140723160517/central/images/4/4d/Circle-formuasl-.jpg https://upload.wikimedia.org/wikipedia/commons/thumb/c/ce/Circle_Area.svg/220px-Circle_Area.svg.png

Circles



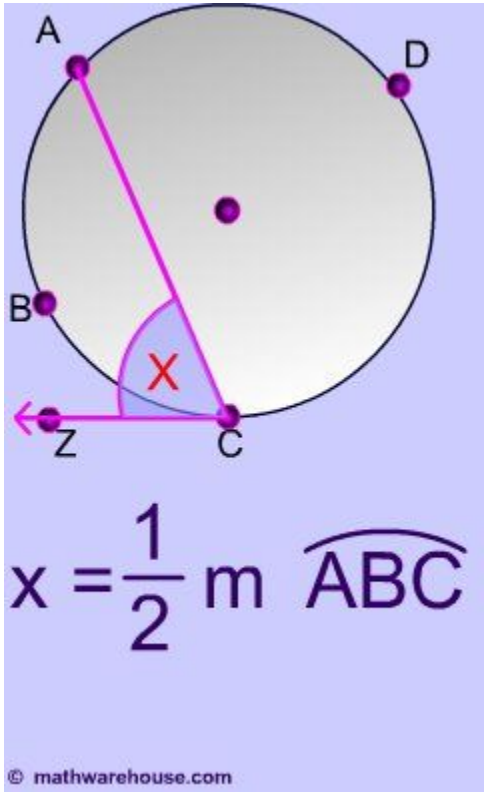


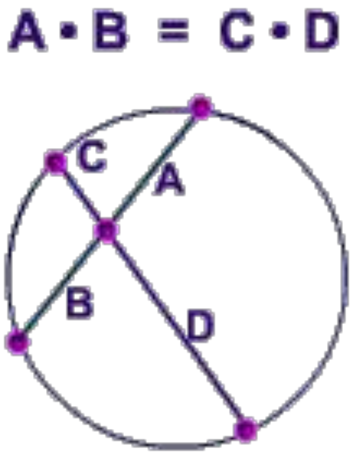


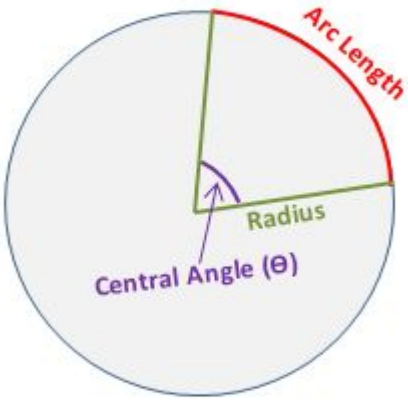


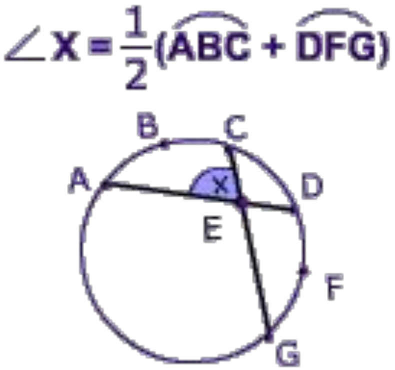
Src: http://thearclengthofacircle.weebly.com/uploads/1/2/3/3/12338358/3055696.gif?160

Circles

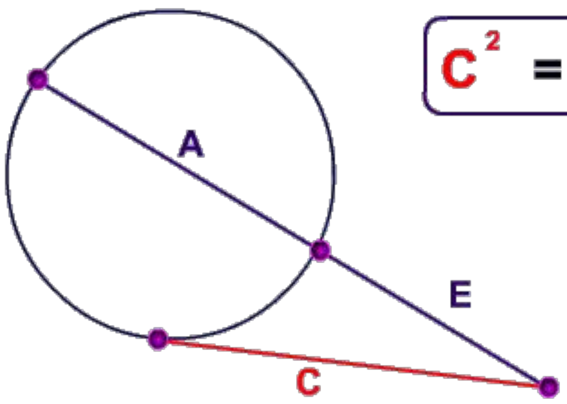

A
D
B
X
Z
C
$x = \frac{1}{2} m \widehat{ABC}$
© mathwarehouse.com


A • B = C • D
C
A
B
D


Arc Length
Radius
Central Angle (Θ)

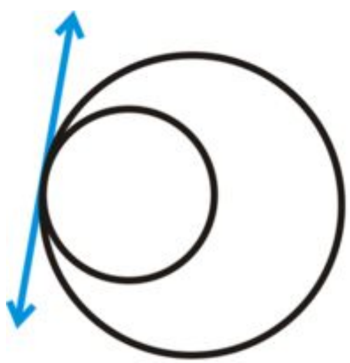

$\angle X = \frac{1}{2}(\widehat{ABC} + \widehat{DFG})$
B
C
A
X
D
E
F
G

Circles

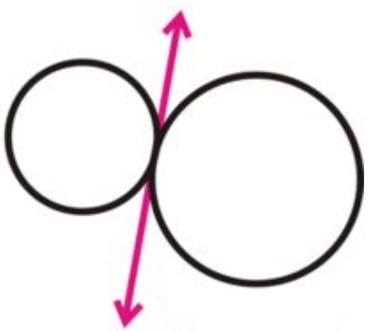


$$C^2 = E \cdot (A + E)$$

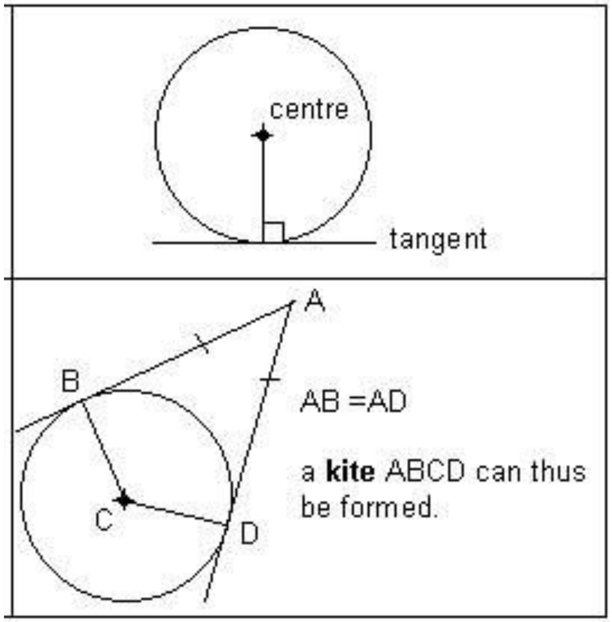
www.mathwarehouse.com



internally tangent



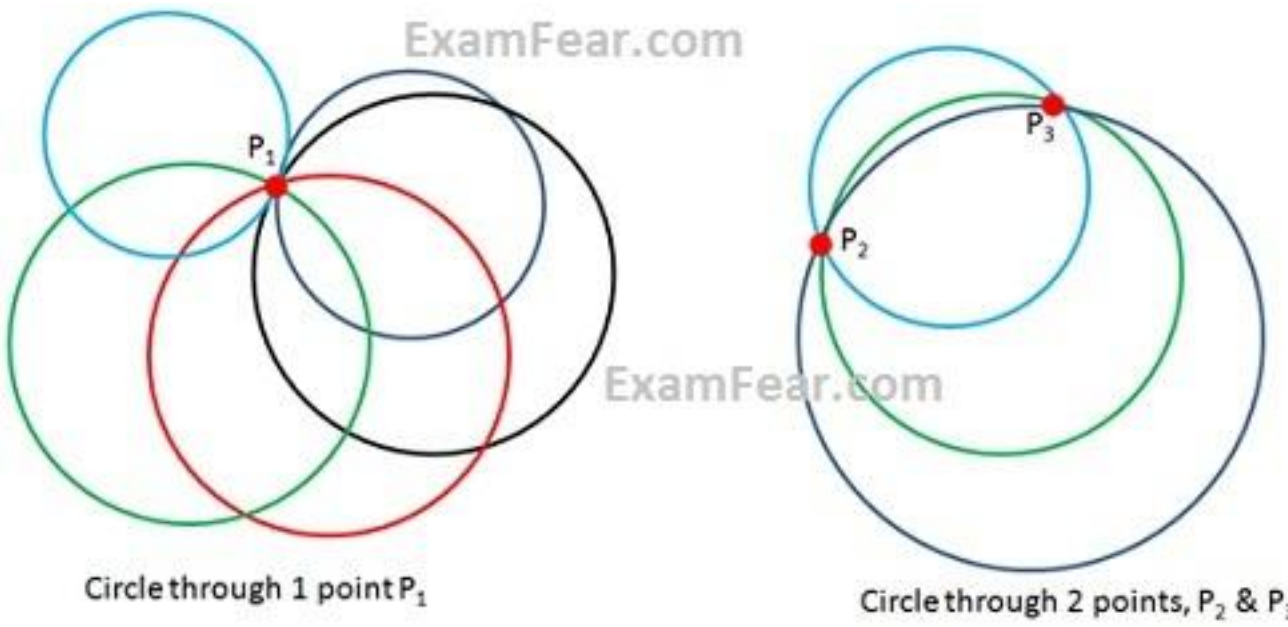
externally tangent



Src: http://www.funmaths.com/math_tutorials/images/tutorial_geometry6_clip_image002.jpg http://www.mathwarehouse.com/geometry/circle/images/secant-tangent-sides/secant-tangent-side-picture.gif

Circle from 1 or 2 points

- Infinite # of circles pass with 1 or 2 points



Src: http://www.examfear.com/notes/Class-9/Maths/Circles/101/Circle-through-3-points.htm

Circle from 3 points

- If they are collinear => no solution
- Otherwise 1 solution exists

Circle through 3 non collinear points P_4, P_5, P_6 Circle through 3 collinear points P_7, P_8, P_9

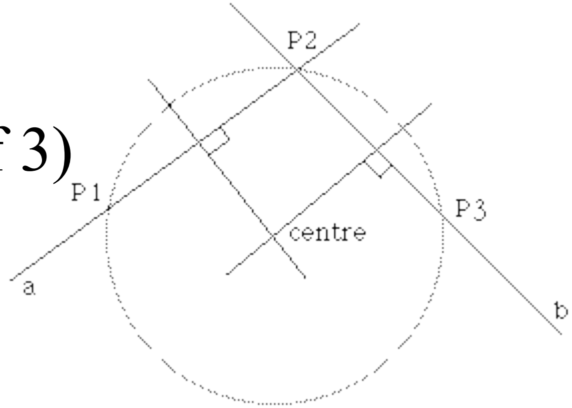
Src: http://www.examfear.com/notes/Class-9/Maths/Circles/101/Circle-through-3-points.htm

Circle from 3 points

- Such as many basic geometry things, there can be several approaches to it (7 here - read)
- This should always trigger in your mind
  - Think from different angles => different solutions
  - So never stuck with one direction
  - Different methods => Different implementations
  - An easy idea can be very tricky to implement
  - Remember books explain approaches for you to use paper and pencil.
  - When converting to code think as programmer. Utilize your utilities.

Using bisectors method

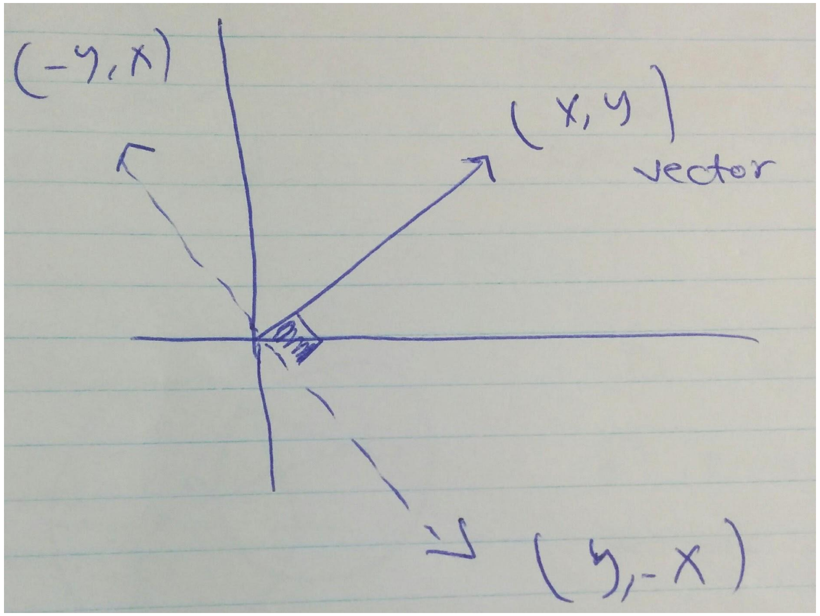
- *The first idea comes to mind is substitution*
- However, a Fact: The perpendicular bisectors of two chords meet at the centre
  - Create 2 lines from the 3 points (they are chords)
  - Get the lines median points
  - Compute 2 perpendicular lines
  - Intersect them = Circle Center
  - Radius = Dist(Center, any point of 3)



Src: http://homer.com.au/webdoc/geometry/circlefrom3_files/circlefrom31.gif

Recall: Vector Perpendicular

- Vector v1 = (x, y) has 2 perpendiculars
  - v2 = (y, -x) or v2 = (-y, x)
  - Hence slope(v1) * slope(v2) = y/x * -x/y = -1

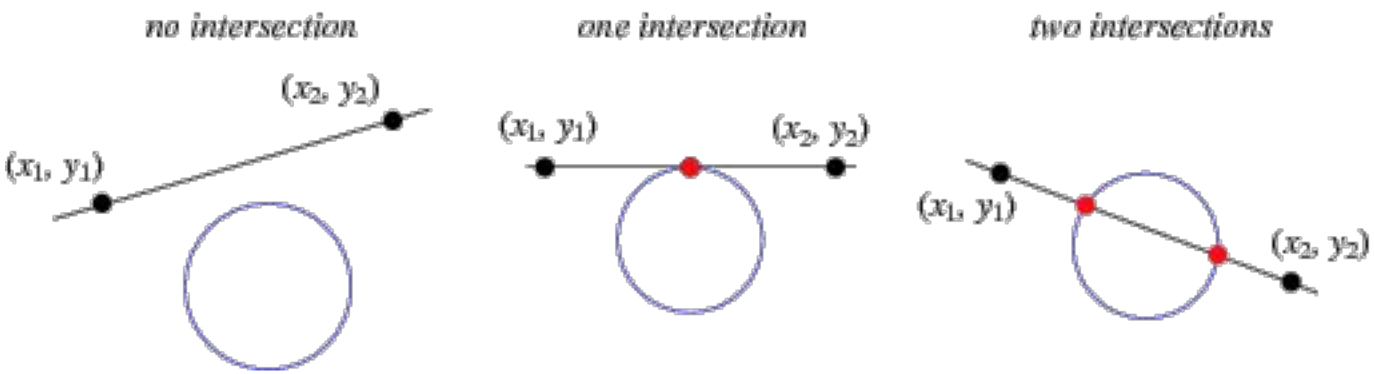


Circle from 3 points

```
pair<double, point> findCircle(point a, point b, point c) {
    //create median, vector, its prependicular
    point m1 = (b+a)*0.5, v1 = b-a, pv1 = point(v1.Y, -v1.X);
    point m2 = (b+c)*0.5, v2 = b-c, pv2 = point(v2.Y, -v2.X);
    point end1 = m1+pv1, end2 = m2+pv2, center;
    intersectSegments(m1, end1, m2, end2, center);
    return make_pair( length(a-center), center );
}
```

Circle-Line Intersection

- Tangent (1 point), Secant (2 points)



Src: http://mathworld.wolfram.com/Circle-LineIntersection.html

Using substitution method

- Let's substitute in parametric format
- Center C, Line (P0, P1), Intersection Point P
  - p = p0 + t(p1-p0) => point location on L
  - (p-c)(p-c) = r^2 => distance of p to c = r
- Substitute p in circle equation and rearrange
  - $(p1\text{-}p0)^2 t^2 + 2(p1\text{-}p0)(p0\text{-}C)t + (p0\text{-}C)^2 = r^2$
  - Quadratic Equation = $ax^2 + bx + c = 0$
  - E.g. $a = (p1\text{-}p0)^2$
  - Sqrt(f): < 0, = 0 , > 0

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

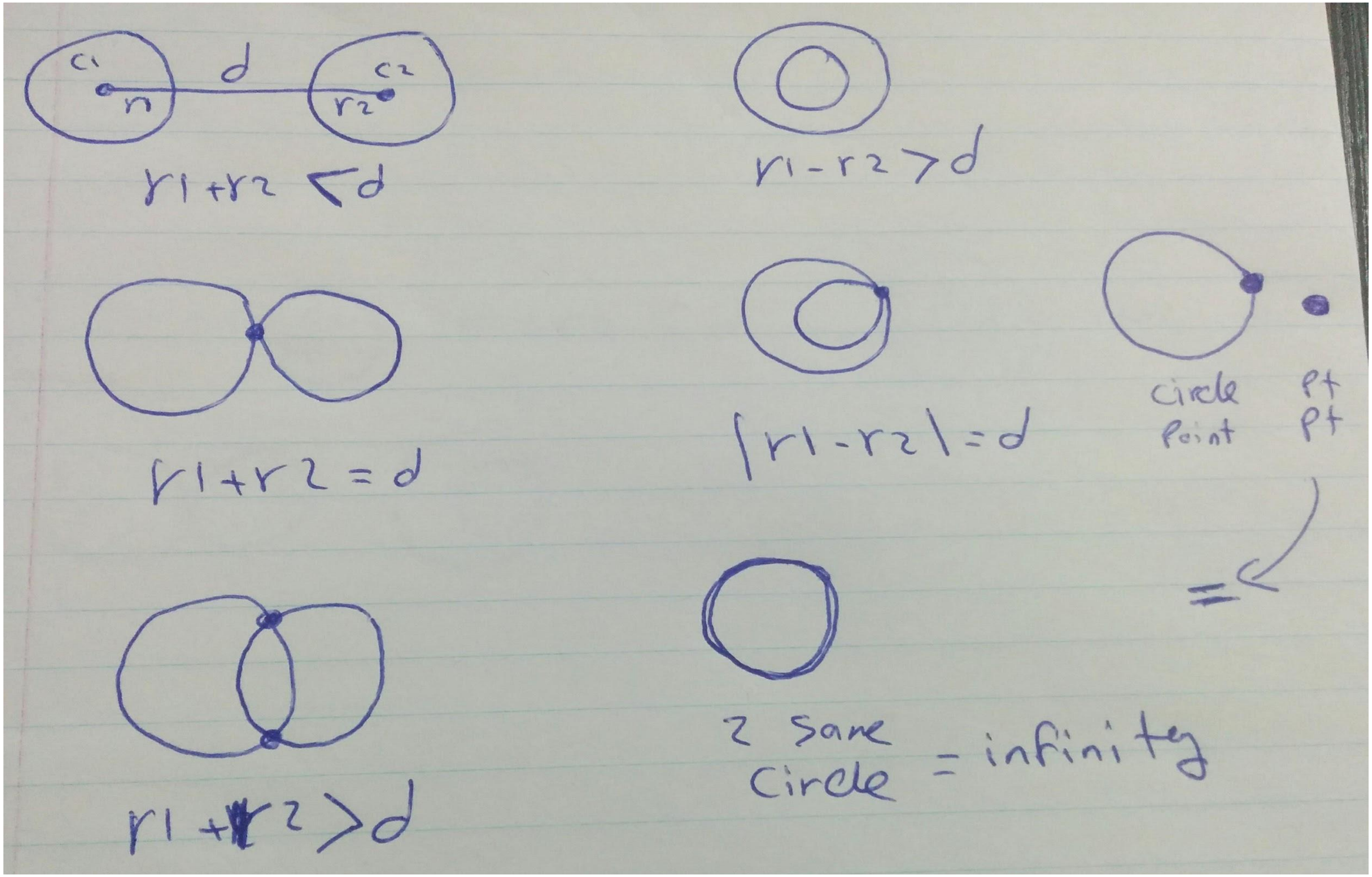
Circle-Line Intersection

```
// return 0, 1 or 2 points (using parametric parameters / substitution method)
vector<point> intersectLineCircle(point p0, point p1, point C, double r) {
    double a = dp(p1-p0, p1-p0), b = 2*dp(p1-p0, p0-C), c = dp(p0-C, p0-C) - r*r;
    double f = b*b - 4*a*c;

    vector<point> v;
    if( dcmp(f, 0) >= 0) {
        if( dcmp(f, 0) == 0)    f = 0;
        double t1 =(-b + sqrt(f))/(2*a);
        double t2 =(-b - sqrt(f))/(2*a);
        v.push_back( p0 + t1*(p1-p0) );
        if( dcmp(f, 0) != 0)
            v.push_back( p0 + t2*(p1-p0) );
    }
    return v;
}
```

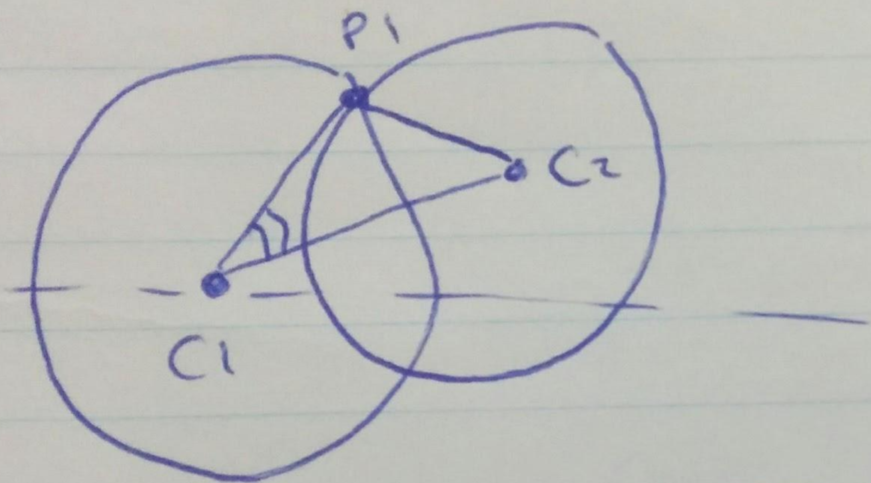
Circle-Circle Intersection

- When intersecting 2 circles
  - 0, 1, 2 intersection points
  - infinity intersection points
- Thinking about amount of variability results in many cases
  - Nested circles, Duplicate circles
  - Circle can be just a point
- Again, we can use substitution method
  - Things might get complicated
  - Let’s analyze little more

All Cases


c1
d
c2
r1
r2
r1+r2 < d
r1-r2 > d
r1+r2 = d
|r1-r2| = d
circle point
pt pt
=
r1 + r2 > d
2 same circle = infinity

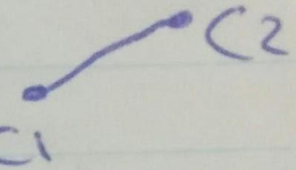
Handling the general case



P1 has lenght = k

P1 angle = 2 angles

angle1 =



angle2 = $\widehat{P_1 C_1 C_2}$

we have $\boxed{\text{3 sides}}$

Implementation Cases Problems

- The safest approach (inside contest), just go and handle case by case
  - Longer code...much code = much bugs
  - This is close to math books solutions style
- Another approach is grouping cases
  - E.g. group cases that can be handled same way
- Let's go here for the extreme
  - Let's handle infinity case alone (logically hard to group)
  - I will handle all others with the general case!
- Grouping is risky..think much...test all

Implementation Cases Problems

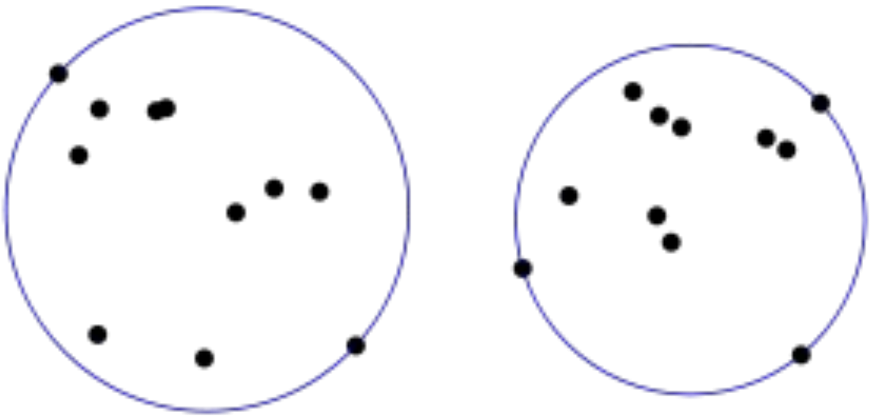
- Just code general way to compute the 2 intersection points
  - If point p1 does not lie on the 2 circles = no intersection
  - If point p1 == p2, then actually it is 1 intersection
  - Otherwise we have 2 intersections
- Concerns
  - When circles are just points distances/radius = 0
  - So make sure this doesn't affect overall
  - And so on

Circle-Circle Intersection

```
vector<point> intersectCircleCircle1(point c1, double r1, point c2, double r2) {
  // Handle infinity case first: same center/radius and r > 0
  if (same(c1, c2) && dcmp(r1, r2) == 0 && dcmp(r1, 0) > 0)
    return vector<point>(3, c1);    // infinity 2 same circles (not points)

  // Compute 2 intersection case and handle 0, 1, 2 cases
  double ang1 = angle(c2 - c1), ang2 = getAngle_A_abc(r2, r1, length(c2 - c1));

  if(::isnan(ang2)) // if r1 or d = 0 => nan in getAngle_A_abc (/0)
    ang2 = 0; // fix corruption

  vector<point> v(1, polar(r1, ang1 + ang2) + c1);

  // if point NOT on the 2 circles = no intersection
  if(dcmp(dp(v[0]-c1, v[0]-c1), r1*r1) != 0 ||
      dcmp(dp(v[0]-c2, v[0]-c2), r2*r2) != 0 )
    return vector<point>();

  v.push_back(polar(r1, ang1 - ang2) + c1);
  if(same(v[0], v[1]))  // if same, then 1 intersection only
    v.pop_back();
  return v;
}
```

Minimal Enclosing Circle

- Given set of points, find a circle of smallest radius that contains them



Src: http://mathworld.wolfram.com/Circle-LineIntersection.html

Minimal Enclosing Circle

- Circle needs maximum 3 boundary points
  - 3 nested loops can compute answer
  - Can we do it linearly?
- Emo Welzl invented a recursive randomized O(N) algorithm
  - To find circle of N points, find it for first N-1 points
  - If pth point inside the computed circle => ok
  - If not, find a bigger circle with pth point on its **boundary**

Minimal Enclosing Circle

- Think in 2 lists P (initially all points) and R (initially empty max size is 3)
  - R is the max 3 boundary points
- If pth point is not in the computed Circle
  - Add pth point to R $\Rightarrow$ Find new bigger circle
  - Recurse for the first N-1 points. Then we build circle for lower points considering some boundary points
- This algorithm looks exponential
  - Welzl pick the point randomly (or shuffle initially)
  - We can prove such randomized behaviour is O(N)

Minimal Enclosing Circle

```
function sed(P,R)
{
    if (P is empty or |R| = 3) then
         D := calcDiskDirectly(R)
    else
        choose a p from P randomly;
        D := sed(P - {p}, R);
        if (p lies NOT inside D) then
            D := sed(P - {p}, R u {p});
    return D;
}
```

Src: http://www.sunshine2k.de/coding/java/Welzl/Welzl.html

Minimal Enclosing Circle

```
const int MAX = 100000+9;
point pnts[MAX], r[3], cen;
double rad;
int ps, rs; // ps = n, rs = 0, initially

// Before running shuffle daya
// random_shuffle(pnts, pnts+ps);
void MEC() {
  if(ps == 0 && rs == 2)
    cen = (r[0]+r[1])/2.0, rad = length(r[0]-cen);
  else if(rs == 3) {
    pair<double, point> p = findCircle(r[0], r[1], r[2]);
    cen = p.second, rad = p.first;
  }
  else if(ps == 0)
    cen = r[0], rad = 0;
  else {
    ps--;
    MEC();

    if(length(pnts[ps]-cen) > rad) {
      r[rs++] = pnts[ps];
      MEC();
      rs--;
    }
    ps++;
  }
}
```

Your Turn: Compute

- For fun and practising, think about some of following:
- Circle-Rectangle Intersection
- Circle-Triangle Intersection
- 3 Circles Intersection
- The area of intersecting circles
- Tangents of 2 circles
- Is line tangent to circle?

تم بحمد الله

علمكم الله ما ينفعكم

ونفعكم بما تعلمتم

وزادكم علماً