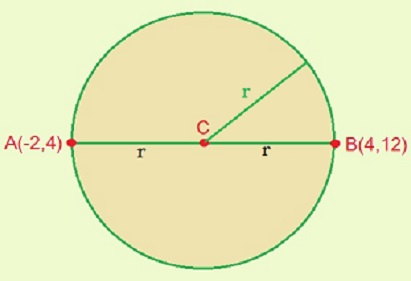
We can create a plane sweep algorithm like the Bentley-ottman algorithm for finding the intersection points to n circles, instead of line segments, in a similar time as the segments. This would require a few modifications, but we can keep the same idea.

One thing we should consider is the radius of the circle, we can find this by finding one end of the circle to the other. So, if we start sweeping through the plane and we see one end of the circle, then go through until we see the other end of the circle. Half of that distance will be the radius. We have left and right end points with Bentley-Ottman, well we have a left and right end points in a circle. So we can keep the same events, i.e. event on right, left and intersection.



Here is what I mean by two endpoints, it should be easy to find the radius too because we can keep the same y values, and just measure the difference between the x values.

We need to make sure that we have a way to find the radius of each circle since we will use these to compare to other circles to see if they intersect. We will also need the midpoint of these circles since that will also play a roll. So, let’s say that we have a circle 0 and circle 1 with r0 and r1 being their radii respectfully. And let’s say that d is the distance between their midpoints. There are several cases, if d > r0 + r1 then they don’t intersect. If d < |r0-r1| then one circle is contained in another, so there are no solutions, d = 0 and r0=r1, then the circles are the same and have infinite solutions. Then if d<r0+r1 and d>|r0-r1| then the circle will intersect.

So, now that we have a way to determine if there are intersection points, we need to know what circle to compare to each other.

We will need to take break all the circles down to half circles, each with a right and left endpoint. The semi circles would be of the upper and lower halves.

**Algorithm.**

Start by breaking down every circle into two semi-circles.

Then we have a priority queue with the leftmost and rightmost endpoints of each circle sorted by their x coordinate.

When we encounter the start event of a circle, we insert the top and bottom halves of that circle, into a balanced binary search tree. Every time we add a new circle to the tree, we test for intersections. When we reach the endpoint of a circle, we remove both half circles.

We test for the intersection in the same way I described before, with comparing the radii and the distance of the midpoints of the circles.

We can then find the intersections using the midpoints, and the distances between the two circles, so we do this by comparing the circle equations. This being (x-x0)^2+(y-y0)^2=r^2, where x0 and y0 being the midpoints of the circles. We need to run through a system of equations to find the intersection points, if they exist, of two circles. So, let’s take two equations for two circles respectively.

(x-x0)^2+(y-y0)^2=r0^2

(x-x1)^2+(y-y1)^2=r1^2

|

|

-2x(x0-x1)-2y(y0-y1) = (r0^2-r1^2)-(x0^2-x1^2)-(y0^2-y1^2)

|

|

Y = ((r0^2-r1^2)-(x0^2-x1^2)-(y0^2-y1^2)+ 2x(x0-x1))/(-2(y0-y1)

With that equation, we can find the y values where the intersections occur, and we can use those y values to find the x values associated with them. The expression for y should also be put back into one of the original circle equations so that we can get a quadratic equation. Then, we can find the answers using the quadratic formula.

Similar to the Bentley-Ottman line segment algorithm, whenever we add, remove, or swap a semicircle, we test its new neighbors for intersection.