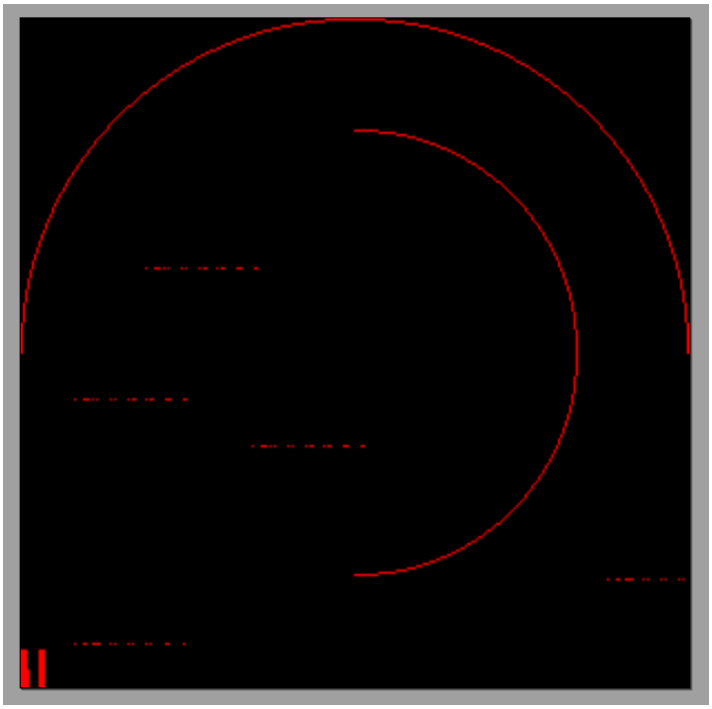
Hw1 required me to create two half circles one with the radius of 150 pixels and the other with a radius of 100 pixels. To create this, I used the midpoint circle algorithm. The algorithm takes a point, lights up the pixel then increments while choosing the best choice of pixel to the ideal circle and lighting up. The algorithm calls a function (called renderPixel in this case) to light up the pixels. In addition to lighting up the coordinate given, rederPixel also lights up any symmetrical point needed. Which in this assignment the needed points are (y,x), (-x,y), (-y,x), (y,-x,), and (x,-y).

However, multiple issues arise in the code given, one being how to get the other quarters of the circles, and the second being to get the circle to render in the correct location on the output file. Thankfully both of these issues can be correct at the same time. To fix the first issue we needed to find an alternative of negative x's and y's. So, to do this I simple subtracted the radius used by the value of x and y given by the rasterizeArc function. So instead of using image[-x][y] and image[-y][x] I used image[x][r - y] and image[y][x - r], similarly for the second circle with the smaller radius I used image[r-x][y] and image[r-y][x] instead of image[x][-y] and image[y][-x], which allowed me to get the desired output. Now for the second issue, all it took was basic addition to relocate the pixels to the proper location. For example, adding 150 (half the output file size) to the coordinates; also adding 50 to the coordinates of the second quadrant of the smaller circle. This allowed us to move all the quadrants from the bottom left of the output file to they're correct quadrants.



I believe the horizontal dotted lines along with the two columns in the bottom left is an error on my end of things. But the circles are rendered how intended with the larger circle in quadrant 1 and 2 being the circle with the radius of 150, and the smaller circle in quadrant 1 and 4 being the circle with the radius of 100.