**Author:**  *Nguyen, Abram*

**Assignment:** *Lab 1 Report*

**Course:** *CS 2302 - Data Structures*

**Instructor:**  *Fuentes, Olac*

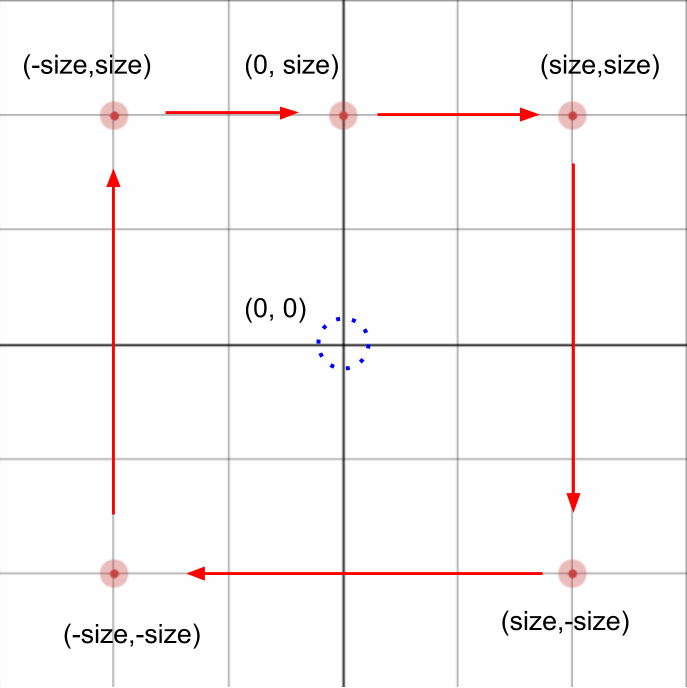
**T.A.:**  *Nath, Anindita*

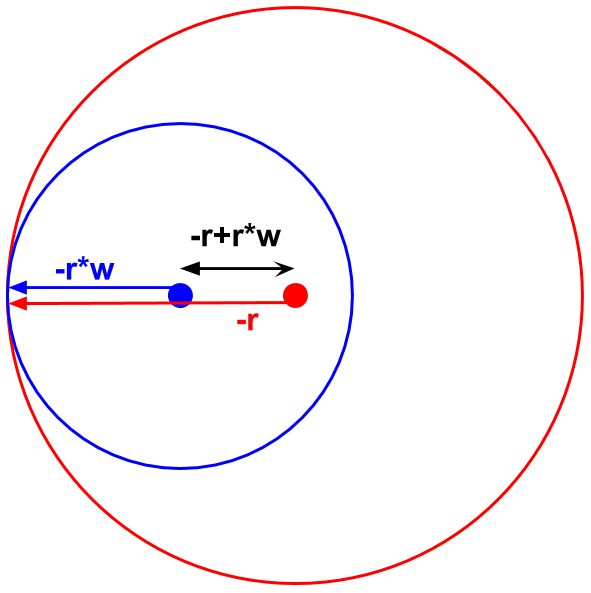
Introduction:

I’m using recursive loops to draw fractals from initial shapes. This helps to illustrate the use of recursion in the real world. For example, the geometry of a snowflake can be explained by the concept of fractals. I can draw a snowflake using recursion, starting with an initial shape. Another example would be drawing a binary tree, which can be drawn and traversed using recursion.

Proposed solution design and implementation:

I separated the program into four parts. Parts 1, 2, 3, and 4 were called

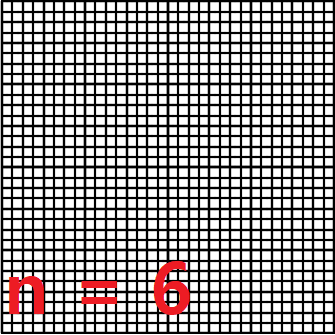
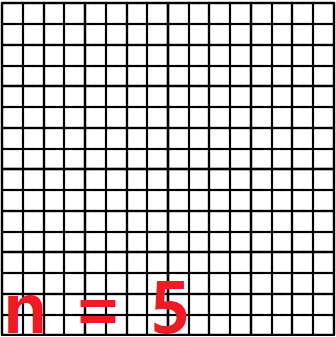
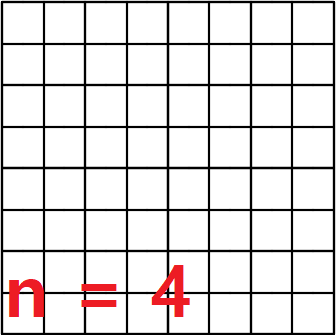
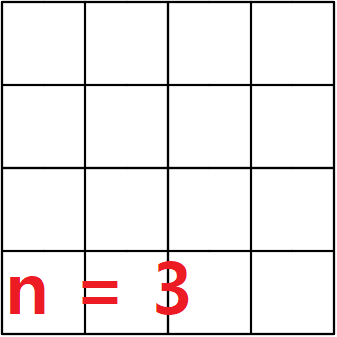
‘squares on corners’, ‘cascading circles’, ‘binary tree representation’, and ‘six circles into one’, respectively. Each section of the program contains method(s) and 3 calls to the method to demonstrate the results of the program. 12 windows will show up when the program is executed, each is a drawing of a figure.

1. For the first figure, I decided to use 4 recursive calls in my method after drawing one square. The first square would be drawn around an origin point. I drew the square using a set of points in relation to my origin point. The points were determined by variable ‘size’, half the length of a square. I plotted the points shown in the image to the right, starting from (0,size) and ending at the same point. Then, I used 4 recursive calls to draw the squares on the corners of the initial square. The new origin point of the four new squares are (size,size), (size,-size), (-size,-size), and (-size,size). This will recurse ‘n’ number of times.
2. For the second figure, I decided to take the approach of drawing one circle at a time around a center point. The radius of this circle will be ‘r’. The radius of a smaller circle within the first is r\*w, ‘w’ being a weighted average, a multiplier that is used to make each consecutive circle smaller by only a certain amount. So, as shown by the image to the right, I moved the center of the new circle to the left(-x coordinate) by r+r\*w.
   1. The calculation: -r - (-r\*w) ⇒ -r+r\*w
3. For the third pattern, I began by drawing an initial shape. This shape is a binary tree with a depth of 1, it’s a head node with 2 leaf nodes. I used a constant variable called ‘size’ to keep the height of every generation of branches consistent. Starting from the head node, I would move my “pen” down by ‘size’ amount and both to the right and left by 2n to draw an initial tree. The variable ‘size’ is a constant variable used to keep the height of each branch or generation consistent. After my initial shape is completed, I would continue the rest of my tree by using 2 recursive calls to draw 2 more shapes, 1 call/shape per previous leaf node. This results in 2n leaf nodes of a tree of depth ‘n’.
4. For the fourth pattern, I started by creating and drawing one circle of radius ‘r’ around center point ‘center’, which is a 2-element array containing a set of x and y coordinates. Then, I created and drew 5 more circles at certain points within the initial circle. The center of all subsequent circles depend on the center of their respective initial circle. In relation to the initial circle, the center of the new center circle would be the same. The centers of the other four would be moved up, down, left, and right by radius\*2 amount. As the method recurses, the radius of circles being drawn will be one-third the size of the previous. This is because of the way 5 circles fit into 1. There are 3 that fit into the circle side-by-side perfectly, so each must be one-third the size.

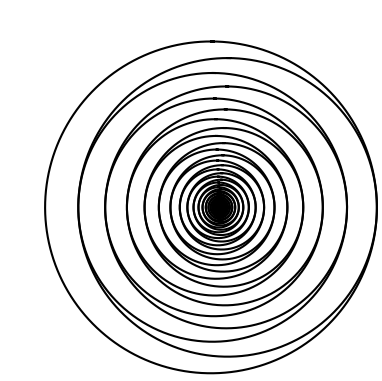
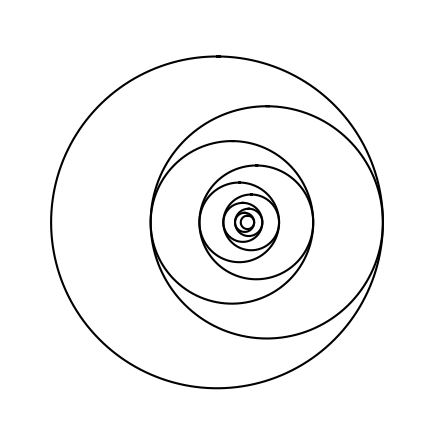
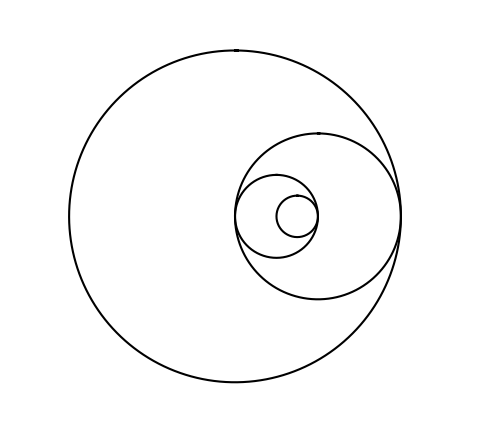
Experimental results:

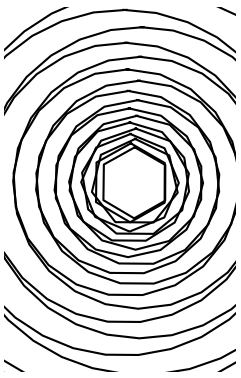
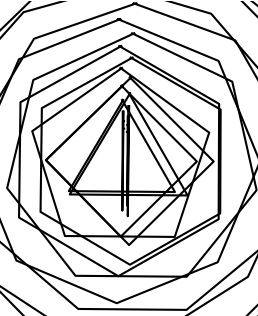
I’ve organized my experiments into 4 parts, each part corresponding to 1 part of the lab.

1. For the first figure, I decided to experiment with the size of the squares and their positioning.
   1. I kept everything in the method the same, except for my ‘x’ and ‘y’ modification in my recursive calls. I added/subtracted the variable ‘size’ to/from my coordinate values originally, but this time halved the variable ‘size’. Instead of drawing with the new squares’ centers on the corner of the initial, the squares’ corners were aligned. They were aligned in such a way that it made the figure look like a checkerboard. The 4 figures below are the results of my experiment. Note that the height/width of these boards are. This experiment took significantly more time to run as ‘n’ increased. My Spyder software ended up crashing when I tried changing ‘n’ to 8 or higher. This was most likely due to the fact that my method contained 4 recursive calls.

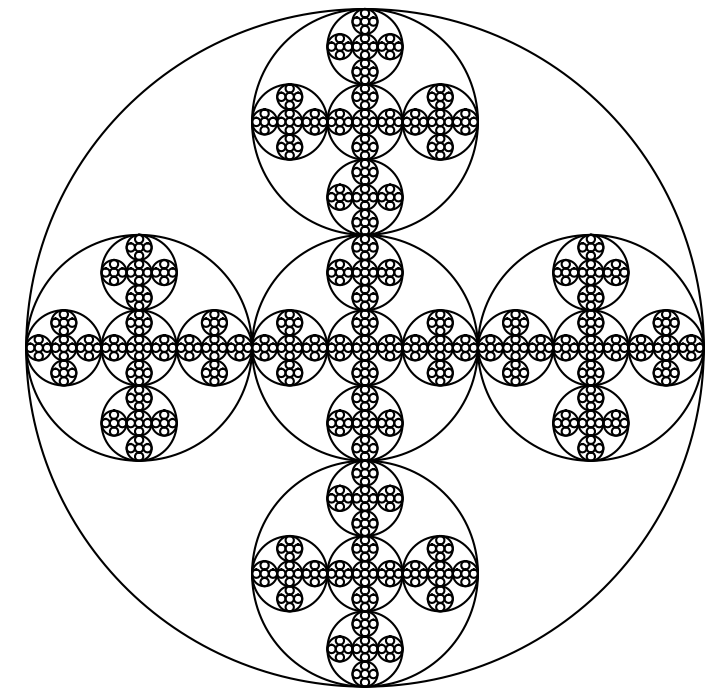
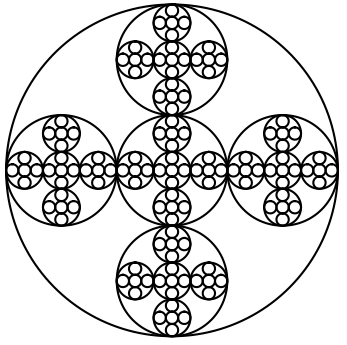
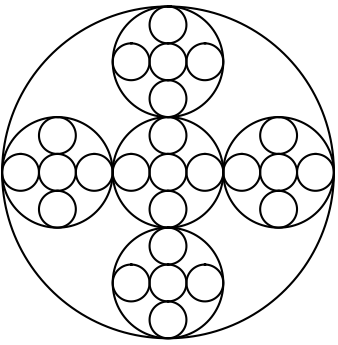


1. For the second figure, I decided to experiment with the orientation of the circles, where they would each be placed in relation to the circle drawn before it.
   1. The first part of this experiment, I tried messing around a little with if-else statements to change up the orientation of the circles every other recursive iteration. In one case, I had the program draw a circle toward the right of the inside of a circle. In the other case, I had the program draw a circle toward the left. This is how it turned out given different values of ‘n’ and ‘w’ (weighted average):

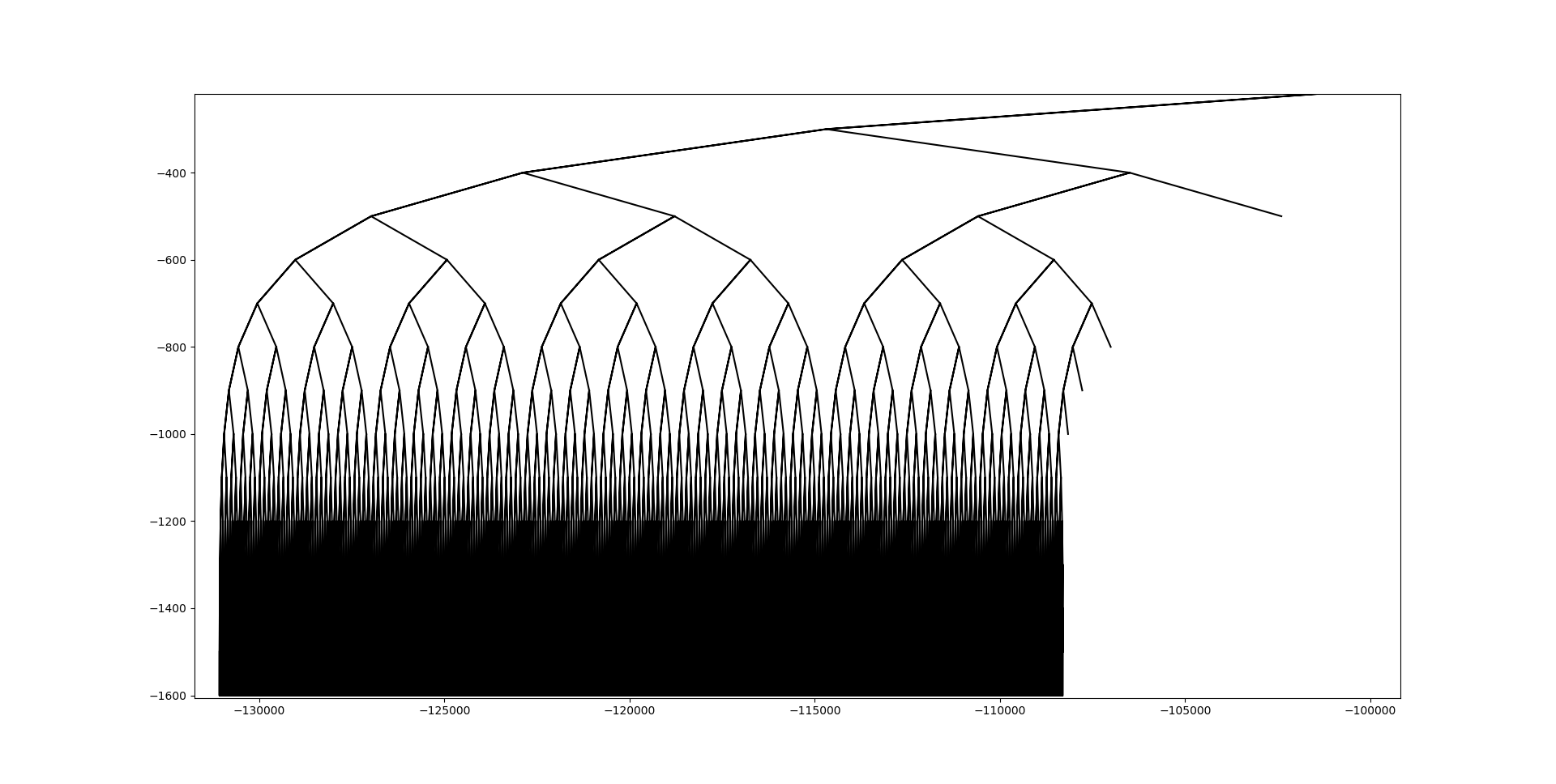
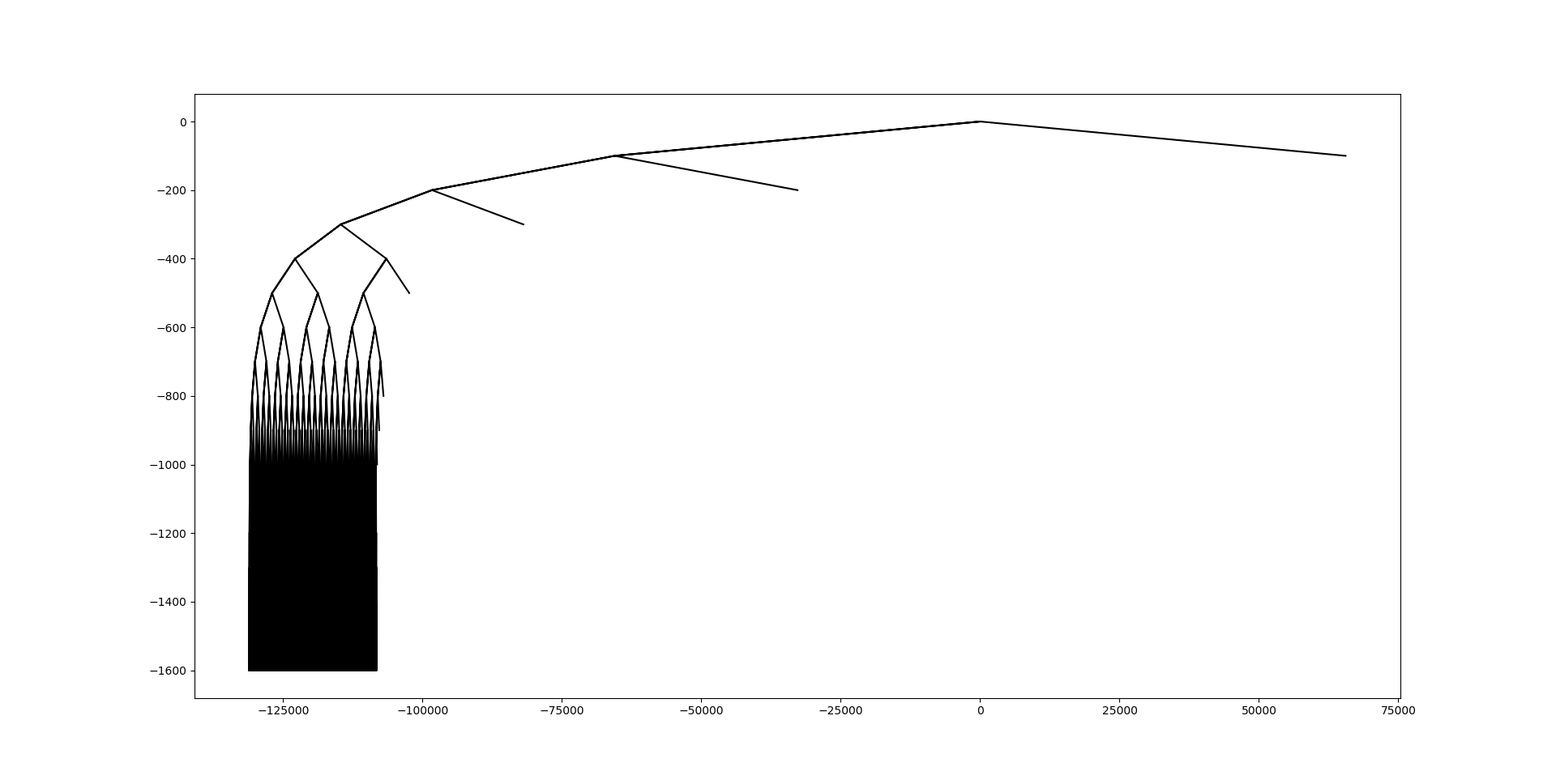


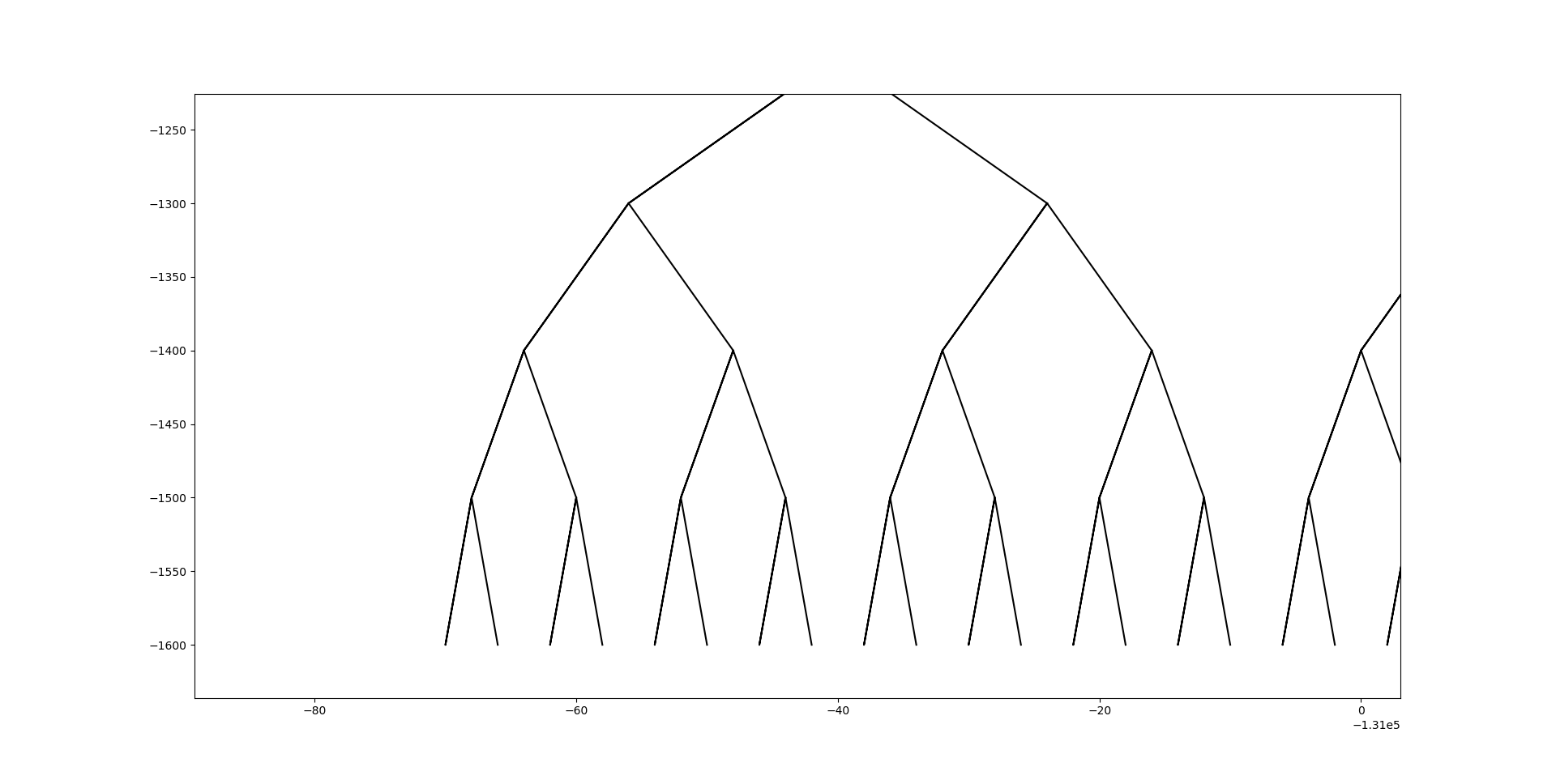
It’s worth noting that when I zoom into the third figure, as the circles become smaller and smaller, I find polygons instead of circles. The polygons at the center seem to decrease in number of sides as ‘n’ increases: ⇒ 

* 1. In the second part of my experimentation with the second part of the lab, I recreated the figure in part 4 of the lab. Understanding the concept of the figures in part 4 made it easy to recreate them with the technique I used to create the figures of part 2. Even so, nothing much has changed, the main difference is that I used the weighted average ‘w’ to divide the radius of the smaller circles by 3. Also, I calculated the center of the new circles in a slightly different way. This time, I used ‘w’ to modify the radius to help me find my new center points. Like the squares in part 1, this experiment took significantly more time to run as ‘n’ increased, even by 1.

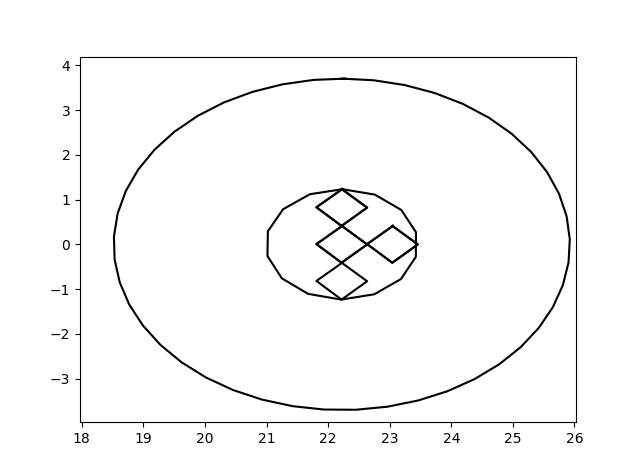
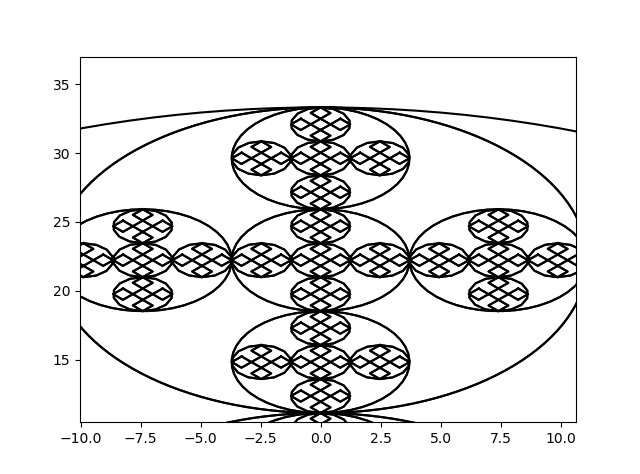
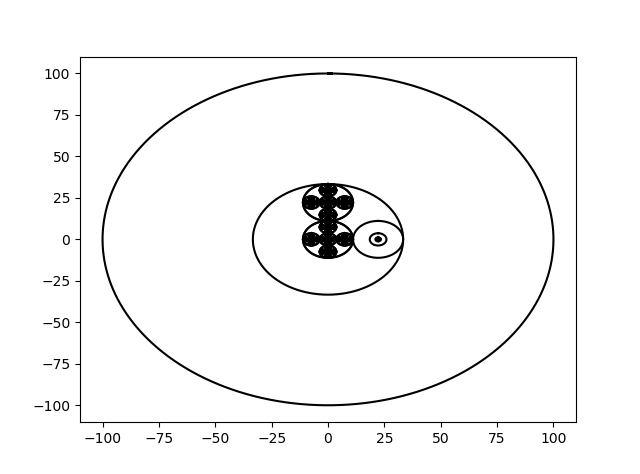


1. For the third figure, I decided to experiment with pushing the limits of the ‘n’ value
   1. In this experiment, I made ‘n’ equal to 16. So, 216 leaf nodes would be created. Because of the limitations of my computer and most likely also Spyder, the graph isn’t fully drawn because I had to force the program to stop early before it crashed or timed out. It becomes clear how Spyder plots the graph using the instructions I gave it. I zoomed in to show that some of the branches were finished:

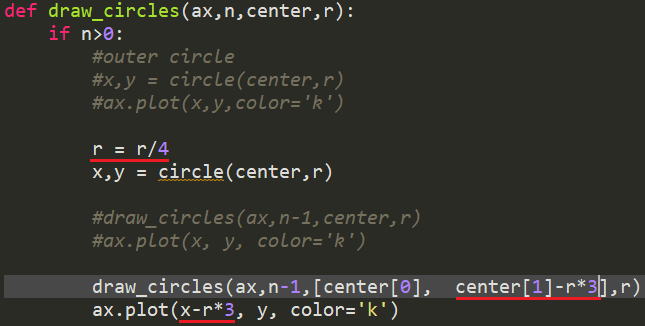
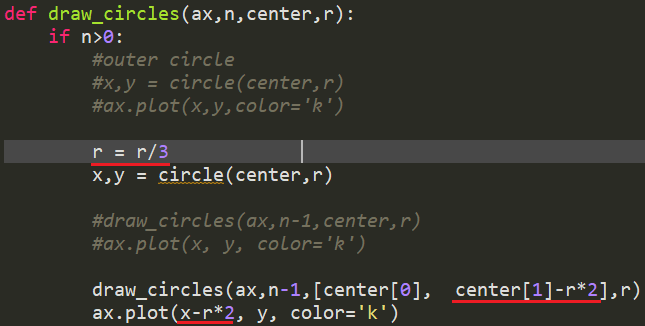
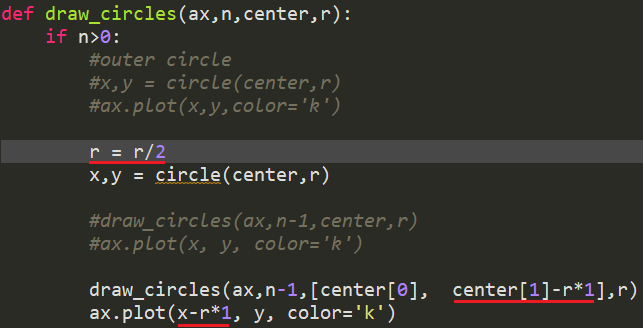


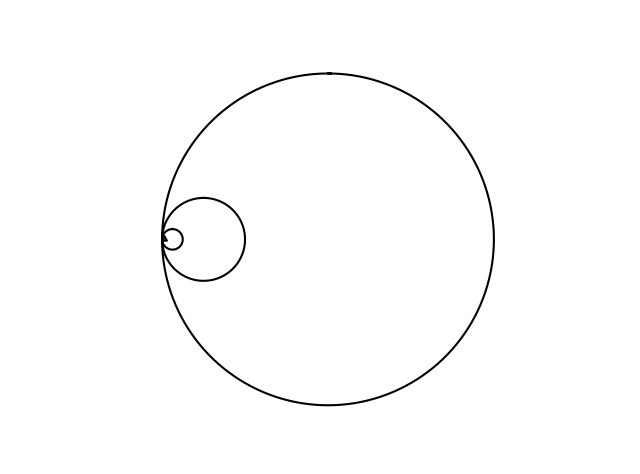
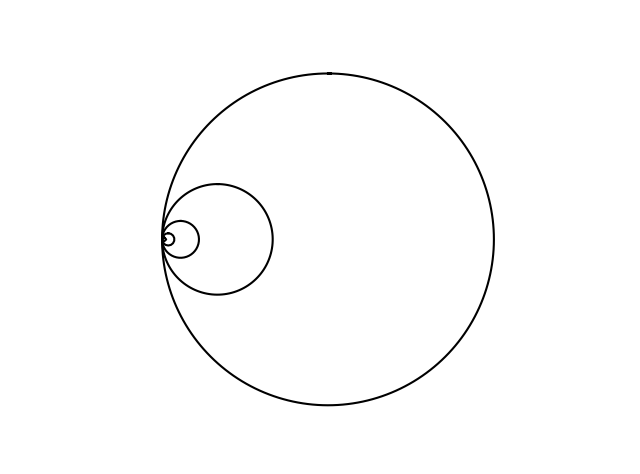
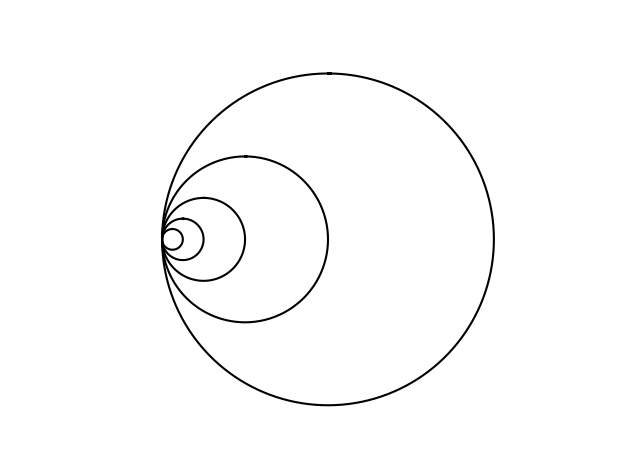


1. For the fourth figure, I decided to test the limits of the method as well as the orientation of the shapes drawn.
   1. In this experiment, I input increasing values of ‘n’ to test the limits of the method. I found that as ‘n’ increased, the smallest circles of the drawing began to shape into squares. The program also took significantly longer to run as I reached n = 6 or n = 7, around these values are where the program would tend to crash. After inputting n = 7, I had to force the program to stop part way through the run. The result looks stretched out horizontally, but I’m not sure why. I unstretched the last two so they would be easier to view:



* 1. In this experiment, I decided to recreate the figures in part 2 of the lab, similar to how I recreated the figures in part 4 using the code from part 2. The major change is the omission of the ‘w’ variable. The values by which ‘r’ is multiplied must be hard-coded, though. It would be difficult to manipulate those numbers. The cases that worked are ones where the number that I divided ‘r’ by is 1 more than the number by which I multiply ‘r’ when I find the origin of the next circle. I found that these cases worked out well:





Conclusions:

I learned a lot about how recursion works, especially now that I got to play around with things I didn’t really get to use before. I understand the limitations of recursion and even the limitations of the tools I used. I found how useful coordinate plotting with Spyder and the matplotlib functions are when visualizing fractals or recursion.

This lab also helped me to understand the syntax of Python 3. Before this semester, I had only worked with Java and had barely worked using Python.

Appendix:

|  |  |
| --- | --- |
| import matplotlib.pyplot as plt  import numpy as np  import math  #---------------------------------------------------------------------------------  #PART 1: SQUARES ON CORNERS - EXPERIMENTAL  #---------------------------------------------------------------------------------  def draw\_squares\_rec(ax,x,y,size,n):  #base case  if n>0:  ax.plot([x,x+size,x+size,x-size,x-size,x],  [y+size,y+size,y-size,y-size,y+size,y+size], color='k')  #instead of placing the squares on corners, place them so that their sides/corners overlap...  #...it created a GRID of squares! The height and width are (2^n)/2  draw\_squares\_rec(ax,x+size/2,y+size/2,size/2,n-1)  draw\_squares\_rec(ax,x+size/2,y-size/2,size/2,n-1)  draw\_squares\_rec(ax,x-size/2,y-size/2,size/2,n-1)  draw\_squares\_rec(ax,x-size/2,y+size/2,size/2,n-1)  fig, ax = plt.subplots()  draw\_squares\_rec(ax,0,0,100,3)  ax.axis('off')  ax.set\_aspect(1.0)  plt.show()    fig, ax = plt.subplots()  draw\_squares\_rec(ax,0,0,100,5)  ax.axis('off')  ax.set\_aspect(1.0)  plt.show()  #---------------------------------------------------------------------------------  #PART 2: CASCADING CIRCLES - EXPERIMENTAL  #---------------------------------------------------------------------------------  def circle(center,rad):  n = int(4\*rad\*math.pi)  #I can change how much of the circle is made, it can be made into just a slice:  #t = np.linspace(0,3,n)  t = np.linspace(0,6.3,n)  x = center[1]+rad\*np.sin(t)  y = center[0]+rad\*np.cos(t)  return x,y  def draw\_circles(ax,n,center,r,w):  if n>0:  x,y = circle(center,r)  ax.plot(x,y,color='k')    #editing the third parameter allows me to change the pattern of circles  #right-left-right-left... pattern  if n%2 == 0:  draw\_circles(ax,n-1,[center[0], center[1]-r\*w+r],r\*w,w)  else:  draw\_circles(ax,n-1,[center[0], center[1]+r\*w-r],r\*w,w)  fig, ax = plt.subplots()  draw\_circles( ax, 100, [0,0], 100, .9)  ax.set\_aspect(1.0)  ax.axis('off')  plt.show()  fig, ax = plt.subplots()  draw\_circles( ax, 10, [0,0], 100, .7)  ax.set\_aspect(1.0)  ax.axis('off')  plt.show()  fig, ax = plt.subplots()  draw\_circles( ax, 4, [0,0], 100, .5)  ax.set\_aspect(1.0)  ax.axis('off')  plt.show()  def draw\_circles2(ax,n,center,r,w):  if n>0:  x,y = circle(center,r)  ax.plot(x,y,color='k')    #recreates PART 4 figure  draw\_circles2(ax,n-1,[center[0], center[1]],r\*w,w)  draw\_circles2(ax,n-1,[center[0]-r+r\*w, center[1]],r\*w,w)  draw\_circles2(ax,n-1,[center[0], center[1]-r+r\*w],r\*w,w)  draw\_circles2(ax,n-1,[center[0]+r-r\*w, center[1]],r\*w,w)  draw\_circles2(ax,n-1,[center[0], center[1]+r-r\*w],r\*w,w)  fig, ax = plt.subplots()  draw\_circles2( ax, 3, [0,0], 100, 1/3)  ax.set\_aspect(1.0)  ax.axis('on')  plt.show()  fig, ax = plt.subplots()  draw\_circles2( ax, 4, [0,0], 100, 1/3)  ax.set\_aspect(1.0)  ax.axis('on')  plt.show()  fig, ax = plt.subplots()  draw\_circles2( ax, 5, [0,0], 100, 1/3)  ax.set\_aspect(1.0)  ax.axis('on')  plt.show()  #---------------------------------------------------------------------------------  #PART 3: BINARY TREE REPRESENTATION - EXPERIMENTAL  #---------------------------------------------------------------------------------  def draw\_bin\_tree(ax,x,y,n,size): #n = height of the tree  #base case  if n>0:  #draw the initial shape (2 lines)  ax.plot([x,x-(2\*\*n),x,x+(2\*\*n)],[y,y-size,y,y-size], color='k')  #draw the next 2 shapes, originating from either leaf of the initial shape  draw\_bin\_tree(ax,x-(2\*\*n),y-size,n-1,size)  draw\_bin\_tree(ax,x+(2\*\*n),y-size,n-1,size)  fig, ax = plt.subplots()  draw\_bin\_tree(ax,0,0,4,100)  ax.set\_aspect(.2)  ax.axis('on')  plt.show()    fig, ax = plt.subplots()  draw\_bin\_tree(ax,0,0,8,100)  ax.set\_aspect(.2)  ax.axis('on')  plt.show()  #fig, ax = plt.subplots()  #draw\_bin\_tree(ax,0,0,16,100)  #ax.set\_aspect(.2)  #ax.axis('on')  #plt.show()  #---------------------------------------------------------------------------------  #PART 4: SIX CIRCLES IN ONE - EXPERIMENTAL  #---------------------------------------------------------------------------------  def draw\_circles(ax,n,center,r):  if n>0:  x,y = circle(center,r)  ax.plot(x,y,color='k')  r = r/3  x,y = circle(center,r)    draw\_circles(ax,n-1,center,r)  ax.plot(x, y, color='k')    draw\_circles(ax,n-1,[center[0]+r\*2, center[1]],r)  ax.plot(x, y+r\*2, color='k')    draw\_circles(ax,n-1,[center[0], center[1]+r\*2],r)  ax.plot(x+r\*2, y, color='k')    draw\_circles(ax,n-1,[center[0]-r\*2, center[1]],r)  ax.plot(x, y-r\*2, color='k')    draw\_circles(ax,n-1,[center[0], center[1]-r\*2],r)  ax.plot(x-r\*2, y, color='k')    fig, ax = plt.subplots()  draw\_circles( ax, 4, [0,0], 100) #keep raising 'n' to find the program's limit  ax.set\_aspect(1.0)  ax.axis('off')  def draw\_circles2(ax,n,center,r):  if n>0:  r = r/4 #+1  x,y = circle(center,r)  draw\_circles2(ax,n-1,[center[0], center[1]-r\*3],r)  # +1  ax.plot(x-r\*3, y, color='k')  # +1    fig, ax = plt.subplots()  draw\_circles2( ax, 5, [0,0], 100)  ax.set\_aspect(1)  ax.axis('off') | |
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|
|

I certify that this project is entirely my own work. I wrote, debugged, and tested the code being presented, performed the experiments, and wrote the report. I also certify that I did not share my code or report or provided inappropriate assistance to any student in the class.

* Abram Nguyen