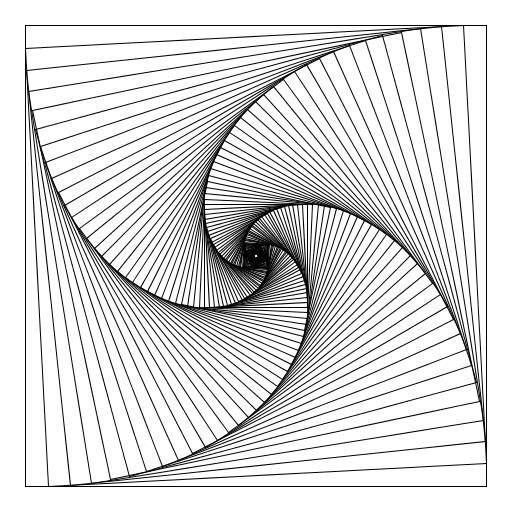
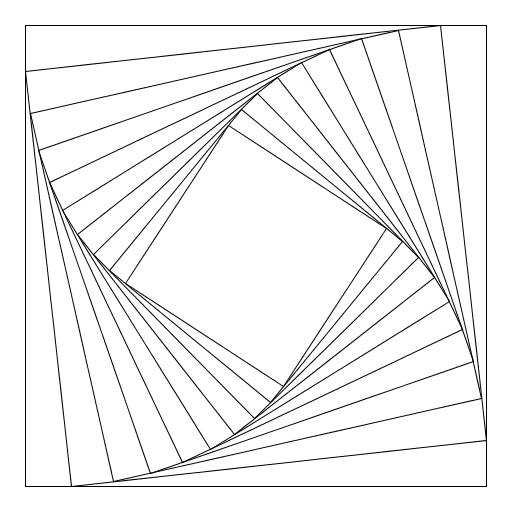
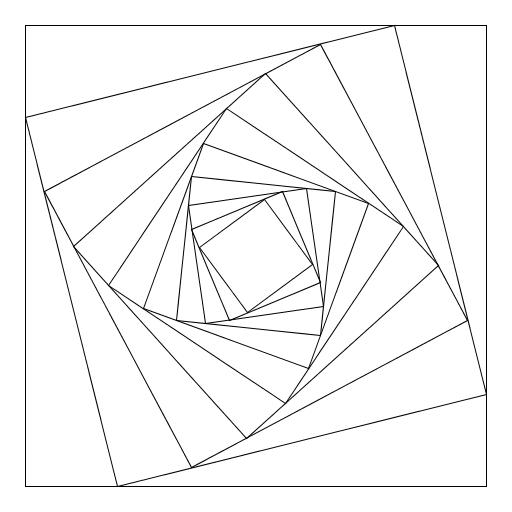
Lab 1 Report

# Introduction

In this Lab, I will discuss how to use recursion to produces complex and intricate figures with diverse algorithmic methods. The primary purpose of the Lab was to properly understand how recursion works and how it can be an effective tool in a programmer's arsenal. The fundamental idea behind recursion is a "base case" that ultimately all recursive methods will arrive at.

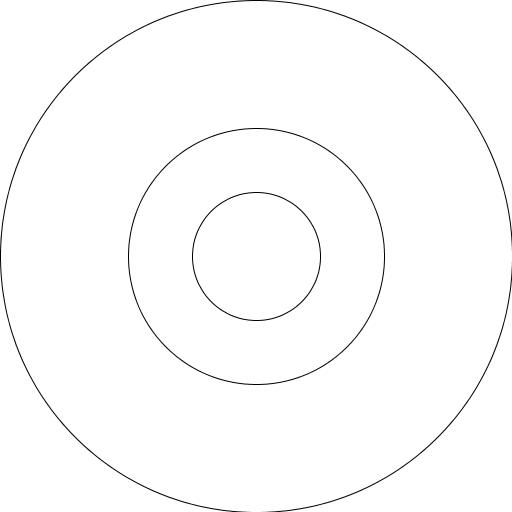
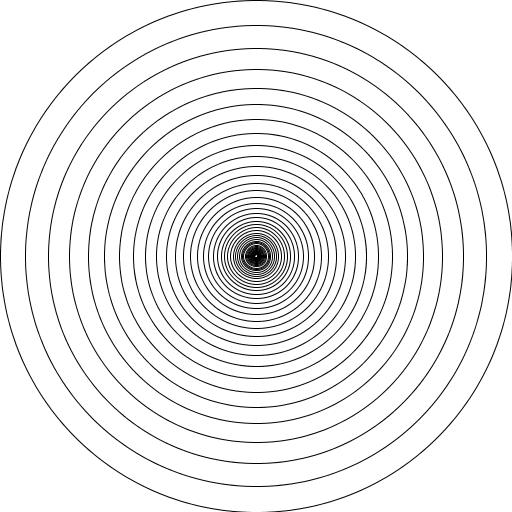
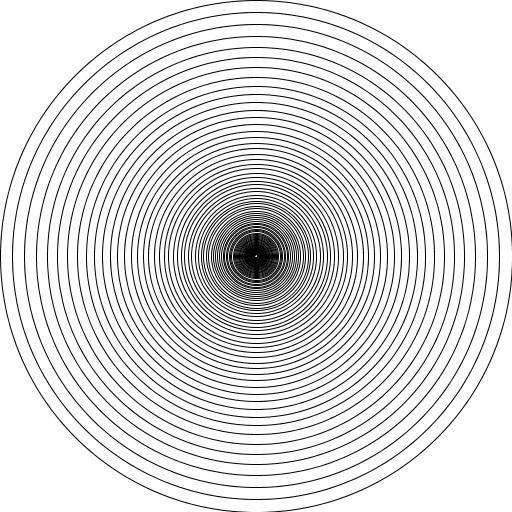
# Proposed Solution & Implementation

Problem 1)



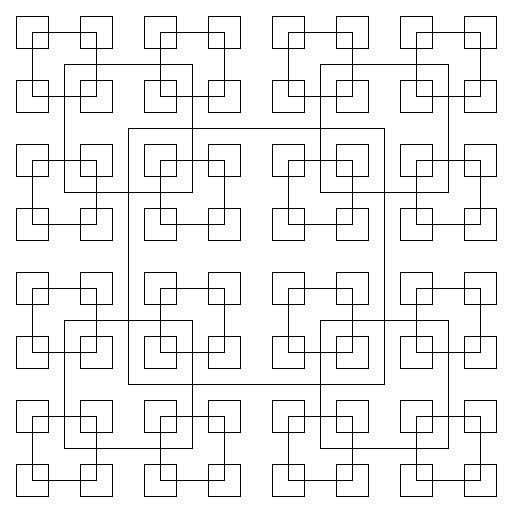
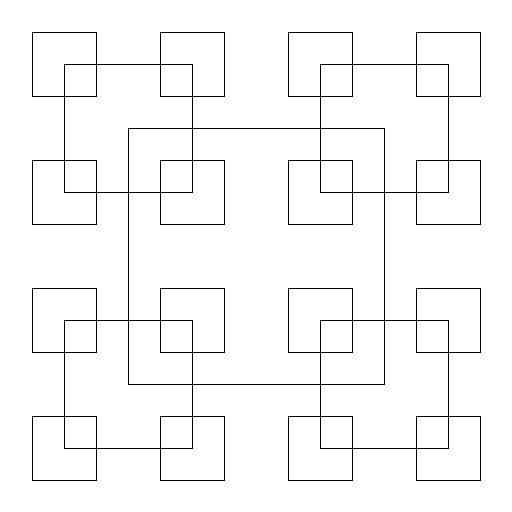
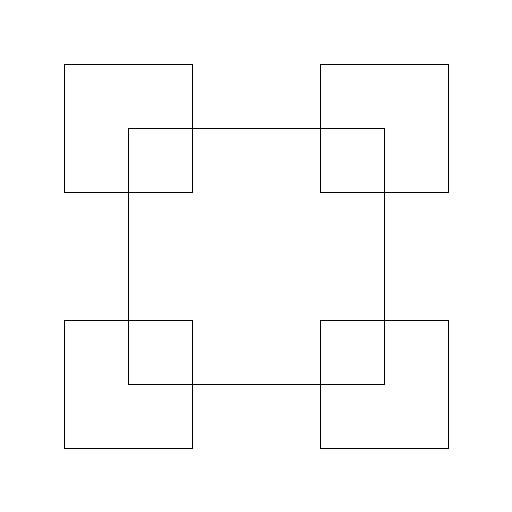
In this portion of the lab we were tasked in drawing the above shapes recursively. My approach to the solution was first noticing how each corner of the next square was a certain distance of the edge of the square before it. In this case the distance we’ll call it ‘w’ varies from . With this in mind I was able to derive that if you multiple each vertex with ‘w’ it reduces the by that factor. Although that was only part of the solution, the other portion required the square to be slanted by a certain amount every recursive call. I found out that if you multiply each vertex by (1 - ‘w’) it shifts it slightly according to what ‘w’ is set to. Therefore the derived solution would be as follows: , where p is the array of vertices of the current square. The recursive call would be calling itself passing n-1, the new array, and the predetermined value for ‘w’

Problem 2)

In this portion of the lab we were tasked in drawing the above shapes recursively. To start off I needed to find how each circle was being recursively drawn. To create the circle initially what needs to be known is the center point of the circle and the radius. I noticed that each circle’s radius is being reduced by a certain factor, we’ll call it ‘w’. Another detail that was noticed was the fact that each newly created circle had the same center point as it’s previous circle. So we can conclude that each newly created circle has the same center point but different radii. Therefore the solution would be as follows: where ‘c’ is the x,y coordinates of the center point and ‘r’ is the radius of the current circle. So the recursive call would be calling itself passing the center point and radius \* w.

Problem 3)

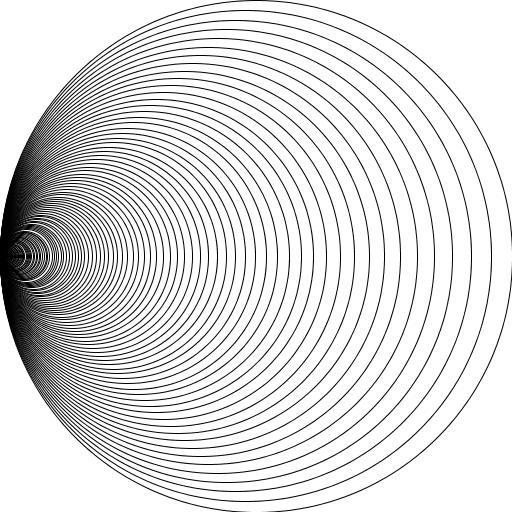
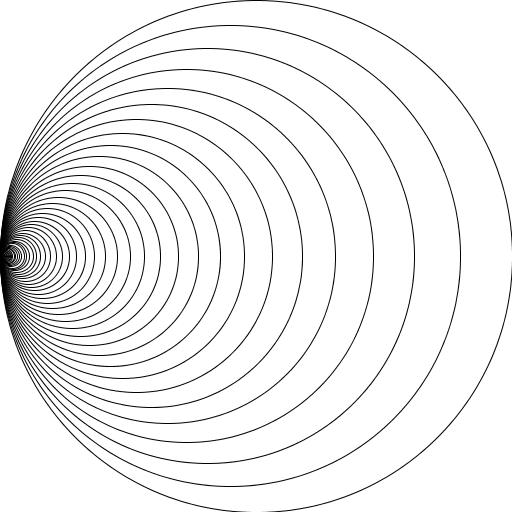
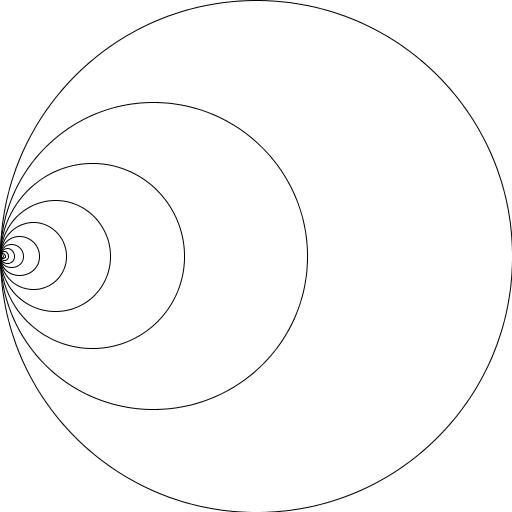


In this portion of the lab we were tasked in drawing the above shapes recursively. This problem was a bit more challenging than compared to the previous problems. The first thing that was noticed was that each square had four smaller squares on each of the corners of the current square. The approach I took to solve this problem was similar to how problem two was solved. In such that each square should have a radius and center point. Although the center point was not necessary to solve this problem, it was nice to keep in mind and check myself. So in order to draw the four smaller squares it was necessary to find the radii of the smaller squares. To do that I took the side lengths of the current square, which was the difference of one vertex to the other. Next, I took that side length and took the square root of the side length squared divided by two to find the radius of the current square: . From there I created a new array of vertices, ‘p’. Starting from the bottom left vertex I did the follow calculations to get the new square’s vertices.

New square ‘q’ = , ,, ,

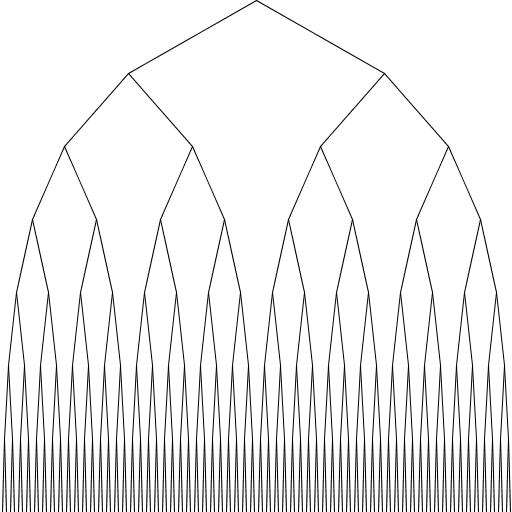
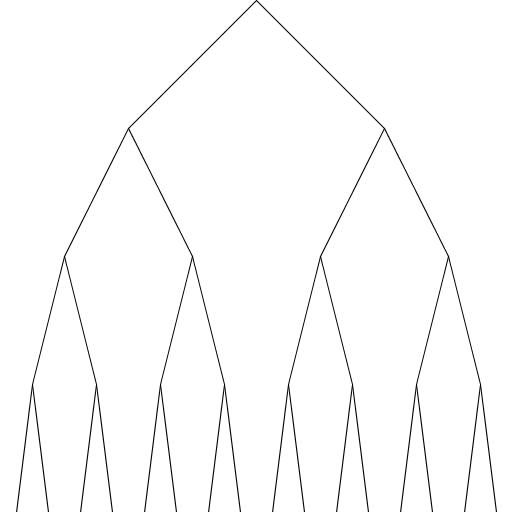
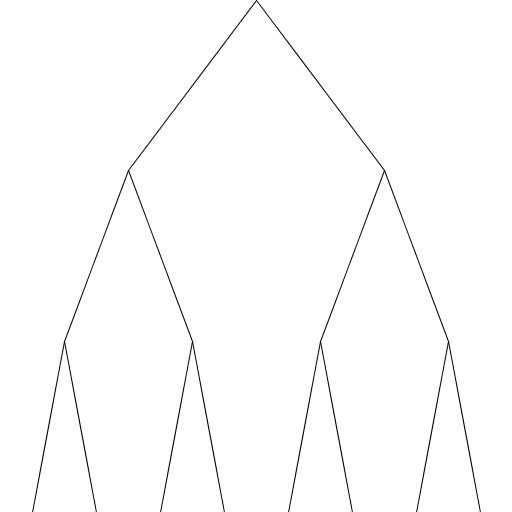
Where ‘x’ is the array index of the first square, which represents each vertex of the current square. The reasoning behind the above solution was because I noticed that each newly created vertex had a distance of either plus or minus the radius divided by 2. Therefore, knowing how each new square was created we can call the recursive function passing ‘n-1’, and q.

Problem 4)



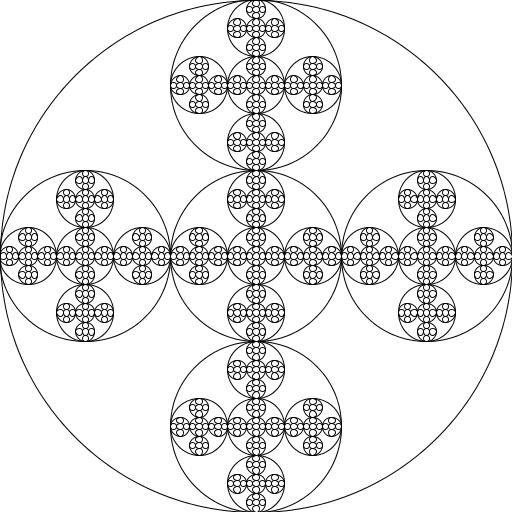
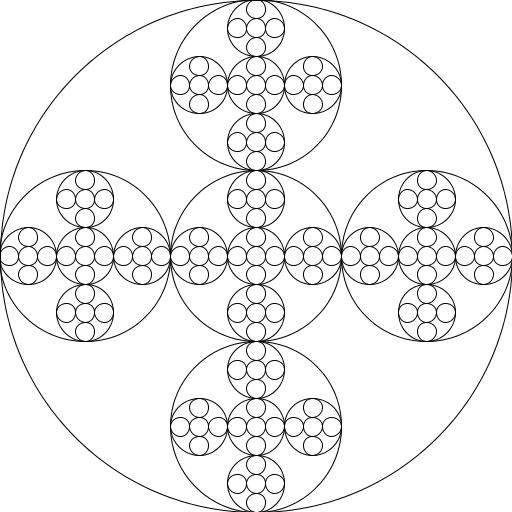
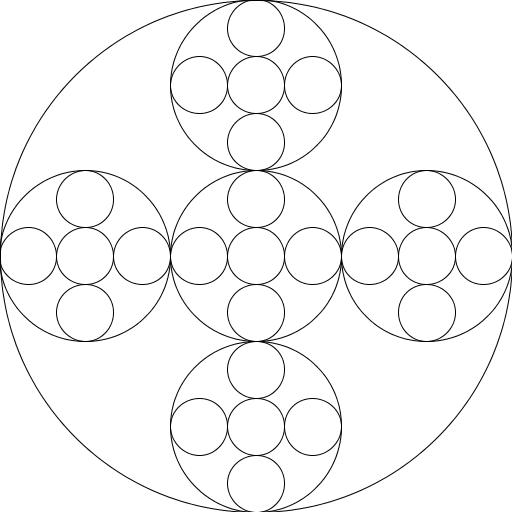
In this portion of the lab we were tasked in drawing the above shapes recursively. The solution to this problem is similar to that of problem 2 but with the center point differing. The approach I took was to shift the center point and radius by a certain factor, we’ll call it ‘w’ where . Therefore, the recursive call would be passing n-1, center \* w, radius \* w.

Problem 5)



In this portion of the lab we were tasked in drawing the above shapes recursively. This problem was probably the most challenging problem to solve. In order to draw the above figures we needed to know some basic fundamentals about tree’s. That is in any given tree will have at most leaves, where ‘n’ is the number of layers or generations a tree will have. Another thing to note about tree’s is that the order of which path to take is crucial in how the tree will get plotted. In this case we need to plot the root node, followed by the left child, back to the parent, to the right child, and then back to the parent again. This traversal is similar to the preorder traversal in which the root gets accessed first, then the left child, and finally the right child. With that all in mind my approach was to first figure out how to determine where to plot the left and right child points. My initial approach was to take the x coordinate of the parent and subtract it from the x coordinate of the parent divided by 2 or, . With the right child taking the x coordinate adding with the x coordinate divided by 2 or, . The issue with this approach was that it worked well for the left side of the tree, but for the right side of the tree it would fail as you can see in the experiments portion of the paper . The reason it failed was because while it worked in the left side, it could not match the same y-coordinate values since the x-coordinate values differed in the right side. Therefore scrapping that approach I decided to just add each child by the following: , where x is the x-coordinate of the current parent, y is the y-coordinate of the current parent, n is the number layers for the tree, and h being the predetermined height of the tree. To get the height of the tree I just took the difference of y-coordinate and y times n and inverted it since it would produce a negative result . Knowing all this I appended to the array ‘p’ for left child by recursively calling the function passing n-1, the left child, and h. Next I appended the parent and followed it by appending the right child by calling the recursive function passing n-1, the right child, and h. I finally appended the parent node to complete the plotting.

Problem 6)



In this portion of the lab we were tasked in drawing the above shapes recursively. This problem was similar to problem 3 and problem 5 in which we were tasked to draw a main circle followed by five more smaller circles. It’s similar because it takes getting the radius and shifting them by a factor of a third since from one end of the circle to the other are three circles. Therefore we can say for each new circle the radius is . Next was to figure out where the center point of each newly created circle was. In order to do this I noticed that each center point of each new circle was shifted horizontally and vertically by the average of the first circle’s radius and the newly created circles’ radius. Therefore starting at the center circle, we called the recursive function passing n-1, the current center point since it didn’t change, and the new radius. The circle to the right of it was the x-coordinate of the center point of the main circle difference the average radius or . The circle above the center circle was the y-coordinate summed with the average radius or . The next circle going clockwise was the x-coordinate summed with the average radius or, . Finally the last circle was the difference of the y-coordinate and the average radius or, .

# 

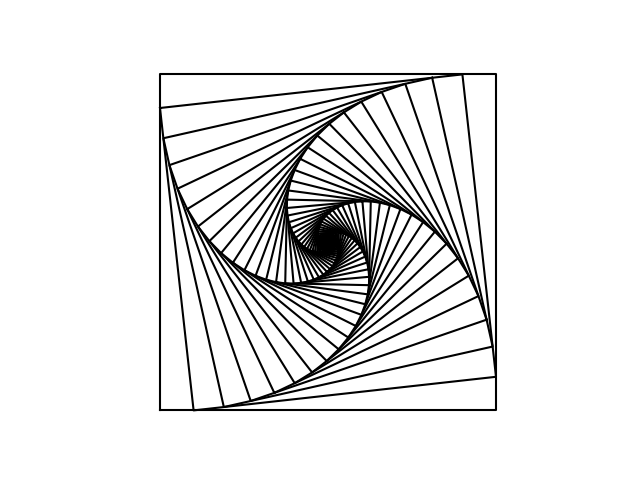
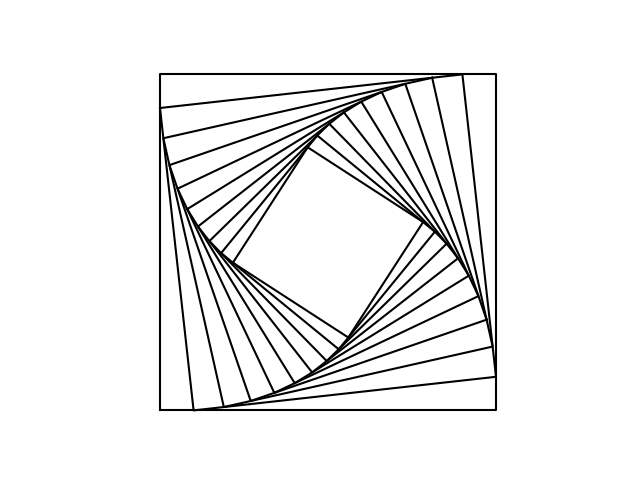
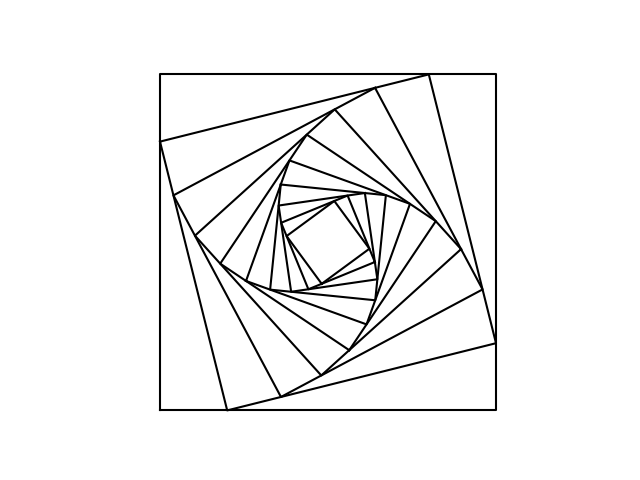
# 

# 

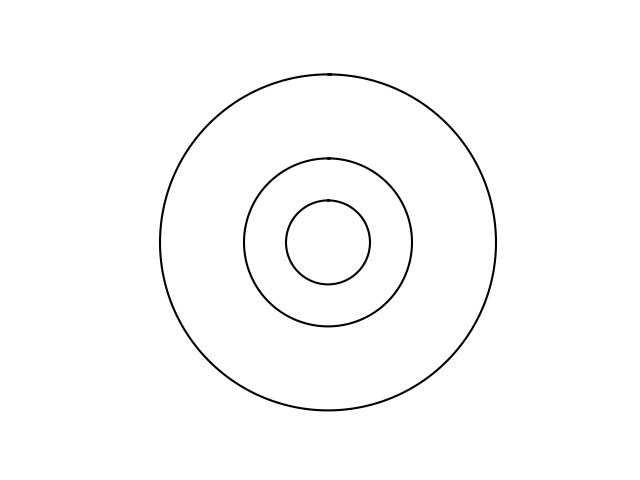
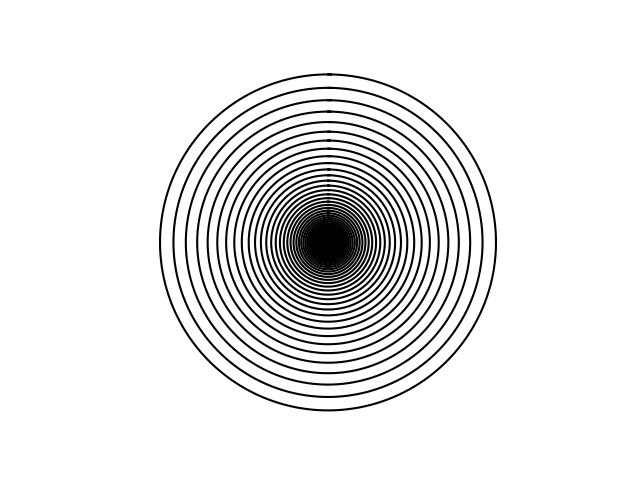
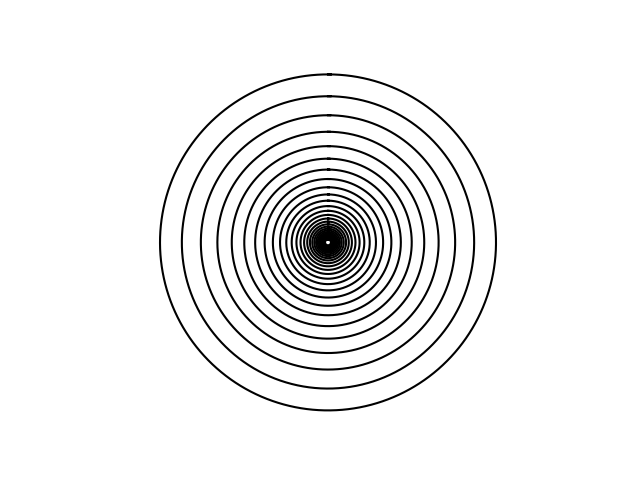
# 

# Experimental Results

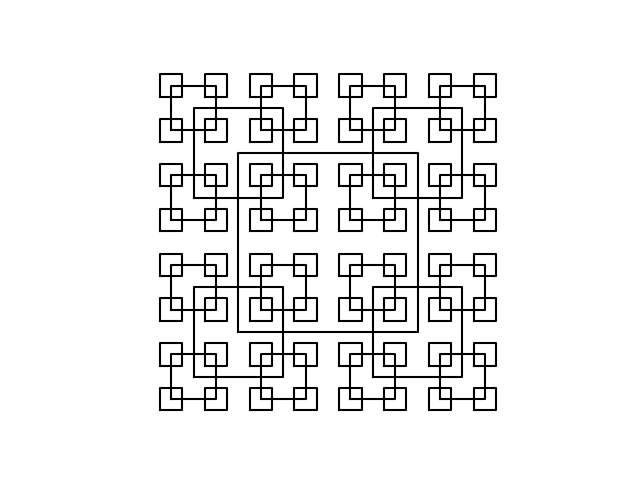
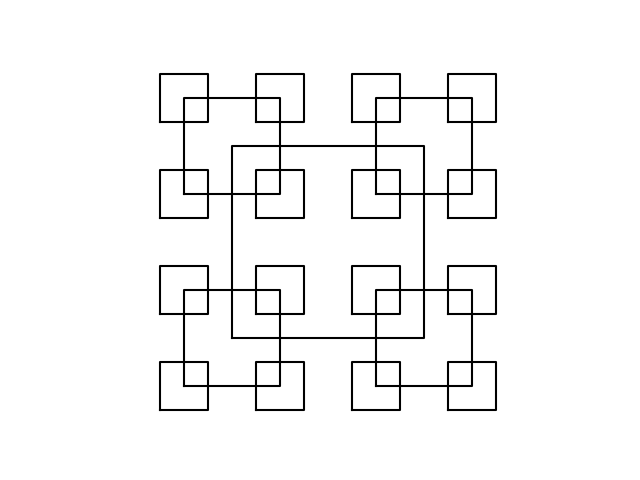
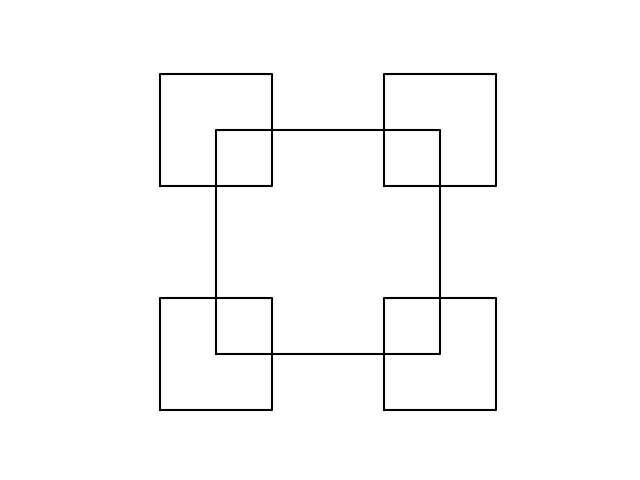
Problem 1)



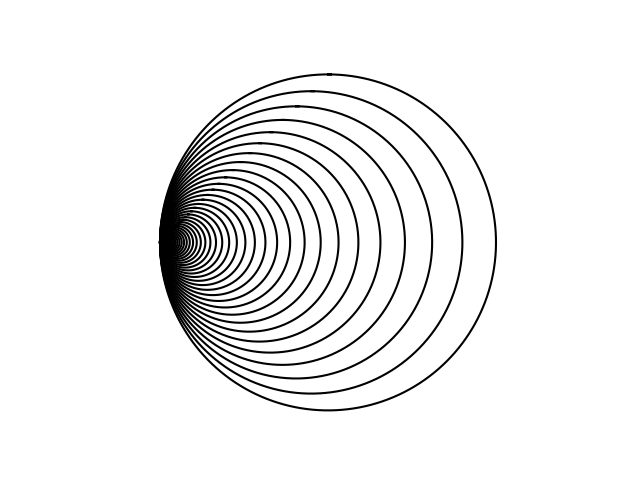
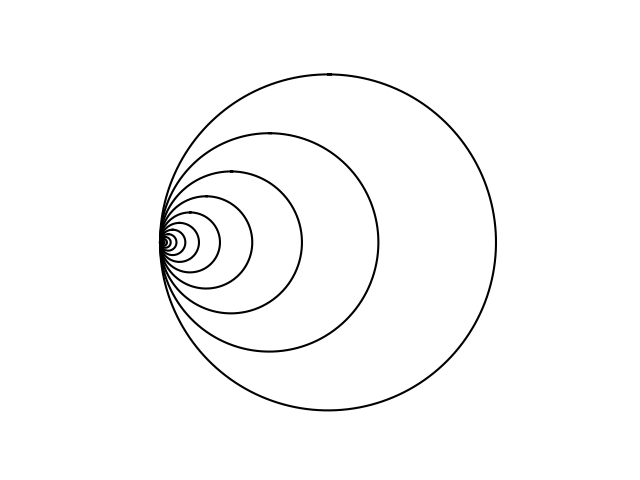
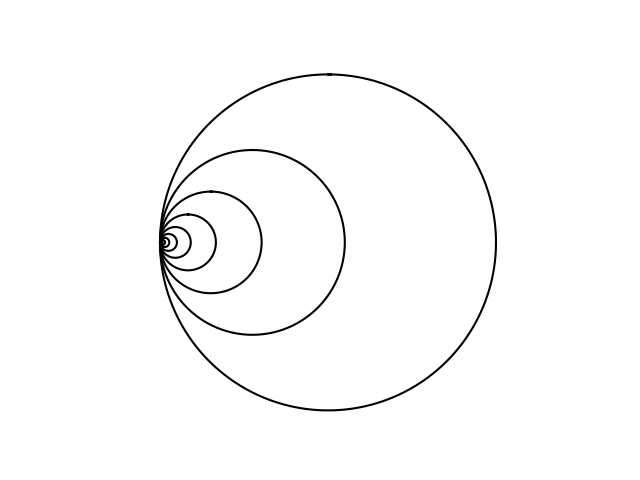
Problem 2)

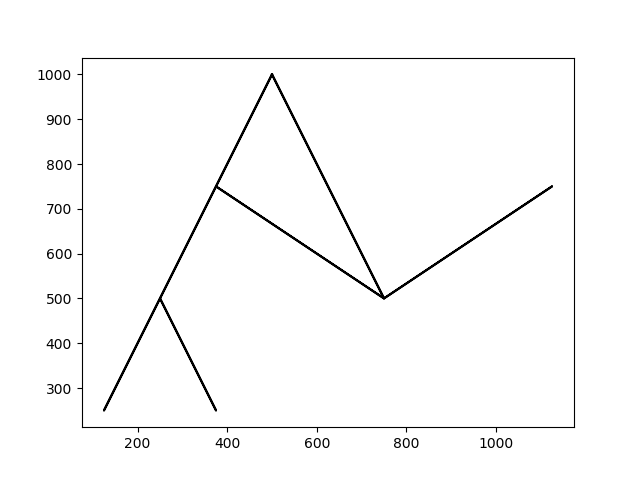
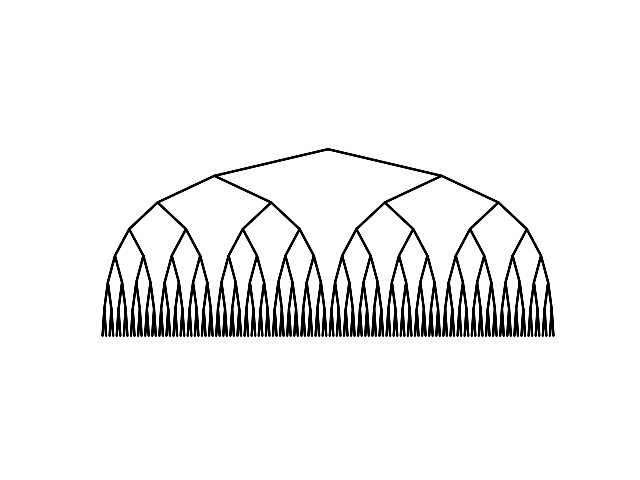
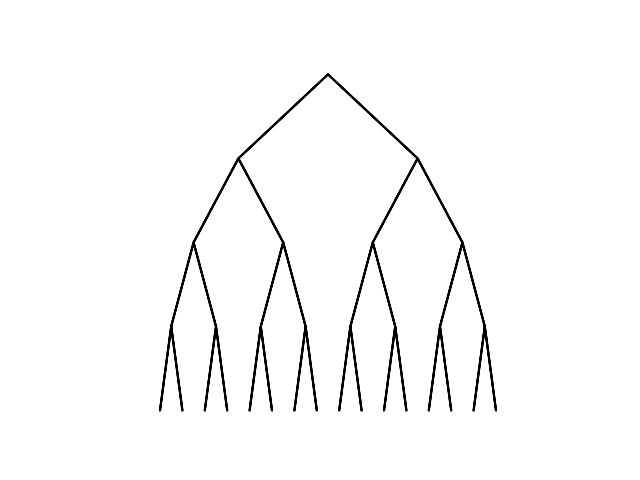
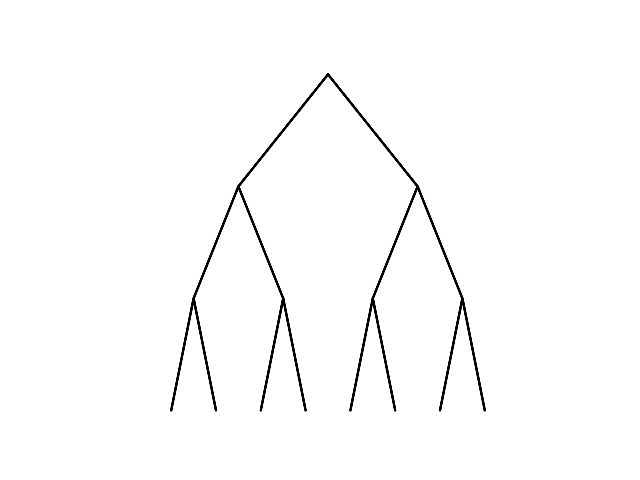
Problem 3)



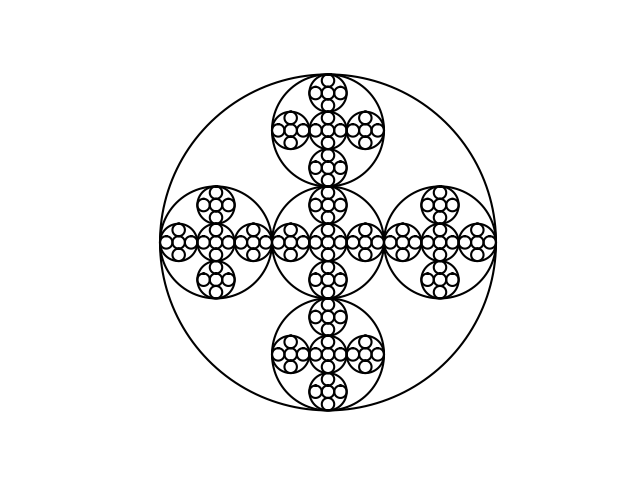
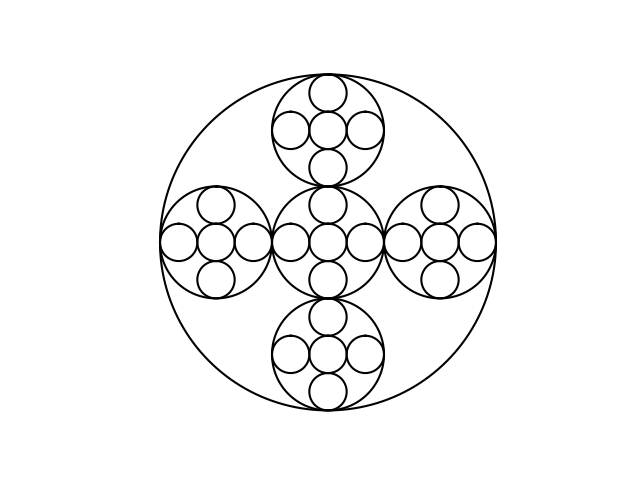
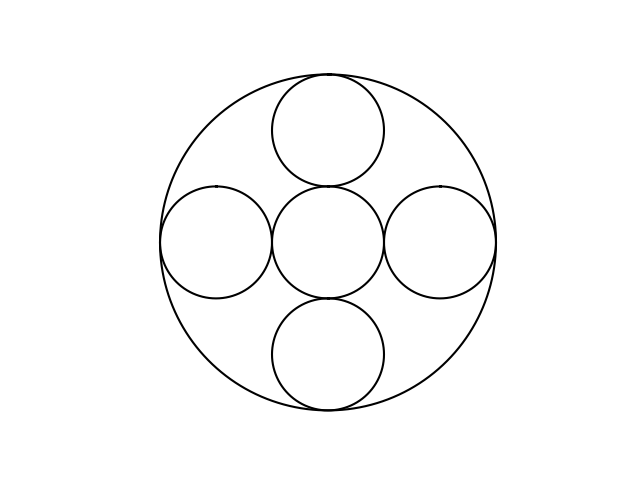
Problem 4)



Problem 5)

fig 5a.

Problem 6)



# 

# 

# Conclusion

In conclusion, I learned how recursion works and how it can be used not only form complex figures but also used to solve complicated problems. I also learned the importance of using a base case and a recursive case to effectively use recursion.

# Appendix

|  |
| --- |
| """ Created on Sun Feb 3 20:02:10 2019  @author: Esteban Andres Bustos Class: CS2302 MWF 1:30 - 3:20pm Last Modified: Feb 10 2019 """  #!/usr/bin/env python3 # -\*- coding: utf-8 -\*-  import numpy as np  import math import matplotlib.pyplot as plt import os  def drawSquares(ax,n,p,w):  """  Plots nothing if array has either no value or N == 0  """  if n > 0 and len(p) != 0:  i1 = [1,2,3,0,1]    "Base Case"  ax.plot(p[:,0], p[:,1], color='k')    "Recursive Call"  q = p\*w + p[i1] \* (1-w)  drawSquares(ax,n-1,q,w)  else:  print('Array Empty!')   def drawMultiSquares(ax,n,p):  """  Plots nothing if array has either no value or N == 0  """  if n>0 and len(p) != 0:  #print('Square Layer %d\n' % n)   #print('plotting coordinates:\n', p)    "Base Case"    """  Base Case is plotting all the points in the first square.  """    ax.plot(p[:,0],p[:,1], color='k')    """  Gets the side length of the current and finds the radius by squaring the length^2 divided by 2  """  a = p[1] - p[0]  rad = (math.sqrt(a[1]\*\*2)) / 2    #mid = p[0] + rad  #print('\nLength of square sides: %d\n' % a[1])  #print('Midpoint of square: \n', mid)  #print('\nRadius of Square: %d\n' % rad)    "Recursive Call"    """  Recursively adding new square points to new array 4 times and passes to recursive function to plot,  but doesn't create any more squares if N < 1  """    if n > 1:  for x in range(4):  q = np.array([[p[x,0] - rad/2, p[x,1] - rad/2],  [p[x,0] - rad/2, p[x,1] + rad/2],  [p[x,0] + rad/2, p[x,1] + rad/2],  [p[x,0] + rad/2, p[x,1] - rad/2],  [p[x,0] - rad/2, p[x,1] - rad/2]])  #print('Created square %d, now going to plot square at point %d ...' %(x+1, x))  drawMultiSquares(ax,n-1,q)  else:  """  Doesn't create more squares if n < = 0  """  return #print('Done with square. Returning to previous call.\n')  else:  print('Array Empty!')     def circle(center, rad):  n = int(4\*rad\*math.pi)  t = np.linspace(0,6.3,n)  x = center[0]+rad\*np.sin(t)  y = center[1]+rad\*np.cos(t)  return x,y  def drawCircles(ax,n,center,radius,w):  """  Plots nothing if array has either no value or N == 0  """  if n>0:  "Base Case"  x,y = circle(center,radius)  ax.plot(x,y,color='k')    "Recursive Call"  drawCircles(ax,n-1,center,radius\*w,w)  else:  print('Array Empty!')   def drawShiftedCircles(ax,n,center,radius,w):    """  Plots nothing if array has either no value or N == 0  """  #print('\nCircle Layer %d' % n)  if n > 0:  "Base Case"  x,y = circle(center, radius)  ax.plot(x,y,color='k')  "Recursive Call"  #print('Initial Center Point:', center)  #print('Radius: %d' % radius)    """  Recursively shifts and creates the new center point of the circle by the variable amount 'w' where 0 < w < 1  """  for x in range(len(center)):  center[x-1] = center[x-1] \* w  #print('New Center Point:', center)  drawShiftedCircles(ax,n-1,center,radius\*w,w)  else:  print('Array Empty!')  def createTree(ax,n,p,h):  #print('\nN = %d' % n)    if n > 0:  "Recursive Case"  #print('Current Tree:')    """  Creates a Left & Right child array  with the following information:    Left Child Coordinates = (X Value of current node) - 2^n , (Y Value of current node) - (height of tree)  Right Child Coordinates = (X Value of current node) + 2^n , (Y Value of current node) - (height of tree)  """    y = p[0,1]  x = p[0,0]    left = np.array([[x - (2\*\*n), y-h]])  right = np.array([[x + (2\*\*n), y-h]])     #print('\nLeft Child: ')  #print(left)  #print('\nRight Child: ')  #print(right)    """  Appends current array with the return values of the left child recursively until n = 0  """    "Left Child"      #print('\nAppending and going to left child')  p = np.append(p, createTree(ax,n-1,left,h), axis=0)    """  Gets the current parent node as to keep the plot order intact  """    "Parent Node"  parent = np.array([[p[0,0], p[0,1]]])  #print('\nAppending parent: ')  #print(parent)  p = np.append(p, parent, axis=0)    """  Appends current array with the return values of the left child recursively until n = 0  """   "Right Child"  #print('\nAppending and going to right child')  p = np.append(p, createTree(ax,n-1,right,h), axis=0)    """  Gets the current parent node as to keep the plot order intact  """    "Parent"  parent = np.array([[p[0,0], p[0,1]]])  #print('\nAppending parent: ')  #print(parent)  p = np.append(p, parent, axis=0)    return p  else:  "Base Case"    """  Returns the current array 'p' if either at a leaf or if N was 0 to start with  """  #print('At leaf returning')  #print(p)  return p   """ Function used to plot all values in the array.  Plots nothing if array has either no value or N == 0 """ def drawTree(ax,n,p):  if n == 0 or len(p) == 0:  #print('At root')  #print(p)  return  else:  y = -1 \* (p[0,1] - (p[0,1] \* n))  p = createTree(ax,n,p,y)  #print('\nDone with creating tree: ')  #print(p)  ax.plot(p[:,0],p[:,1], color='k')     def drawMultiCirlces(ax,n,center,radius):  #print('\nCurrent Layer: %d' % n)  if n > 0:  "Recursive Case"    "Draws Initial Circle"  x,y = circle(center,radius)  ax.plot(x,y,color='k')    """  Circles' new radius is as follows:  New Radius = Current Radius / 3  Reasoning behind it is to make sure the main circle is divided by 3 parts. The left, center, and right circle.  """  new\_rad = radius/3  Rad\_ave = (radius + new\_rad) / 2  #print('Current Radius: %d\nNew Radius: %d\nRadius Average: %d' %(radius,new\_rad,Rad\_ave))  tmp\_x = center[0]  #print('\nTemp X Value: %d' % tmp\_x)  tmp\_y = center[1]  #print('Temp Y Value: %d\n' % tmp\_y)    """  Creates the 5 circles  """  c1 = center  #print('Center Circle 1: ', c1)  drawMultiCirlces(ax,n-1,c1,new\_rad)  #print('Back from Circle 1')    c2 = np.array([tmp\_x - Rad\_ave, tmp\_y])  #print('Center Circle 2: ', c2)  drawMultiCirlces(ax,n-1,c2,new\_rad)  #print('Back from Circle 2')    c3 = np.array([tmp\_x, tmp\_y + Rad\_ave])  #print('Center Circle 3: ', c3)  drawMultiCirlces(ax,n-1,c3,new\_rad)  #print('Back from Circle 3')    c4 = np.array([tmp\_x + Rad\_ave, tmp\_y])  #print('Center Circle 4: ', c4)  drawMultiCirlces(ax,n-1,c4,new\_rad)  #print('Back from Circle 4')    c5 = np.array([tmp\_x, tmp\_y - Rad\_ave])  #print('Center Circle 5: ', c5)  drawMultiCirlces(ax,n-1,c5,new\_rad)  #print('Back from Circle 5')  else:  "Base Case"  #print('At N = 0, return back to previous call...\n')  x,y = circle(center,radius)  ax.plot(x,y,color='k')   ############################ functions end here ###########################################  plt.close("all") orig\_size = 1000  path = "Lab1\_Output\_Images"  try:  os.mkdir(path) except OSError:  print("Failed to create directory '%s' as it already exists" % path) else:  print("Created path %s successfully" % path)   # Different Lab Figures:  ### Problem 1 ## Squares  "a" p = np.array([[0,0],[0,orig\_size],[orig\_size,orig\_size],[orig\_size,0],[0,0]]) fig, ax = plt.subplots() ax.axis('off') ax.set\_aspect(1.0) drawSquares(ax,10,p,.2) plt.show() fig.savefig('Lab1\_Output\_Images/lab1\_1a.png')  "b" p = np.array([[0,0],[0,orig\_size],[orig\_size,orig\_size],[orig\_size,0],[0,0]]) fig, ax = plt.subplots() ax.axis('off') ax.set\_aspect(1.0) drawSquares(ax,10,p,.1) plt.show() fig.savefig('Lab1\_Output\_Images/lab1\_1b.png')  # c p = np.array([[0,0],[0,orig\_size],[orig\_size,orig\_size],[orig\_size,0],[0,0]]) fig, ax = plt.subplots() ax.axis('off') ax.set\_aspect(1.0) drawSquares(ax,100,p,.1) plt.show() fig.savefig('Lab1\_Output\_Images/lab1\_1c.png')  ## Circles  "a" fig, ax = plt.subplots() ax.axis('off') ax.set\_aspect(1.0) drawCircles(ax, 3, [100,0], 100,.5) plt.show() fig.savefig('Lab1\_Output\_Images/lab1\_2a.png')  "b" fig, ax = plt.subplots() ax.axis('off') ax.set\_aspect(1.0) drawCircles(ax, 30, [100,0], 100,.87) plt.show() fig.savefig('Lab1\_Output\_Images/lab1\_2b.png')  "c" fig, ax = plt.subplots() ax.axis('off') ax.set\_aspect(1.0) drawCircles(ax, 100, [100,0], 100,.92) plt.show() fig.savefig('Lab1\_Output\_Images/lab1\_2c.png')  ## Problem 2  "a" p = np.array([[0,0],[0,orig\_size],[orig\_size,orig\_size],[orig\_size,0],[0,0]]) fig, ax = plt.subplots() ax.axis('off') ax.set\_aspect(1.0) drawMultiSquares(ax,2,p) plt.show() fig.savefig('Lab1\_Output\_Images/lab1\_3a.png')  "b" p = np.array([[0,0],[0,orig\_size],[orig\_size,orig\_size],[orig\_size,0],[0,0]]) fig, ax = plt.subplots() ax.axis('off') ax.set\_aspect(1.0) drawMultiSquares(ax,3,p) plt.show() fig.savefig('Lab1\_Output\_Images/lab1\_3b.png')  "c" p = np.array([[0,0],[0,orig\_size],[orig\_size,orig\_size],[orig\_size,0],[0,0]]) fig, ax = plt.subplots() ax.axis('off') ax.set\_aspect(1.0) drawMultiSquares(ax,4,p) plt.show() fig.savefig('Lab1\_Output\_Images/lab1\_3c.png')  ## Problem 3  "a" fig, ax = plt.subplots() ax.axis('off') ax.set\_aspect(1.0) drawShiftedCircles(ax,10,[100,0], 100,.55) plt.show() fig.savefig('Lab1\_Output\_Images/lab1\_4a.png')  "b" fig, ax = plt.subplots() ax.axis('off') ax.set\_aspect(1.0) drawShiftedCircles(ax,55,[100,0], 100,.65) plt.show() fig.savefig('Lab1\_Output\_Images/lab1\_4b.png')  "c" fig, ax = plt.subplots() ax.axis('off') ax.set\_aspect(1.0) drawShiftedCircles(ax,65,[100,0], 100,.90) plt.show() fig.savefig('Lab1\_Output\_Images/lab1\_4c.png')  ## Problem 4  "a" p = np.array([[5,5]]) fig, ax = plt.subplots() ax.axis('off') ax.set\_aspect(1.0) drawTree(ax,3,p) plt.show() fig.savefig('Lab1\_Output\_Images/lab1\_5a.png')  "b" p = np.array([[5,5]]) fig, ax = plt.subplots() ax.axis('off') ax.set\_aspect(1.0) drawTree(ax,4,p) plt.show() fig.savefig('Lab1\_Output\_Images/lab1\_5b.png')  "c" p = np.array([[5,5]]) fig, ax = plt.subplots() ax.axis('off') ax.set\_aspect(1.0) drawTree(ax,7,p) plt.show() fig.savefig('Lab1\_Output\_Images/lab1\_5c.png')  ## Problem 5  "a" fig, ax = plt.subplots() ax.axis('off') ax.set\_aspect(1.0) drawMultiCirlces(ax,1, [100,100], 100) plt.show() fig.savefig('Lab1\_Output\_Images/lab1\_6a.png')  "b" fig, ax = plt.subplots() ax.axis('off') ax.set\_aspect(1.0) drawMultiCirlces(ax,2, [100,100], 100) plt.show() fig.savefig('Lab1\_Output\_Images/lab1\_6b.png')  "c" fig, ax = plt.subplots() ax.axis('off') ax.set\_aspect(1.0) drawMultiCirlces(ax,3, [100,100], 100) plt.show() fig.savefig('Lab1\_Output\_Images/lab1\_6c.png') |