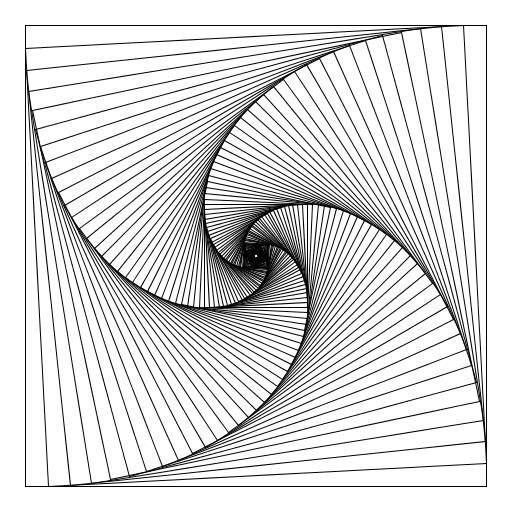
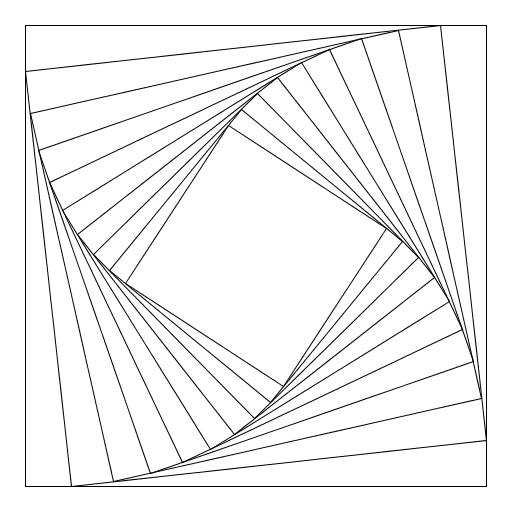
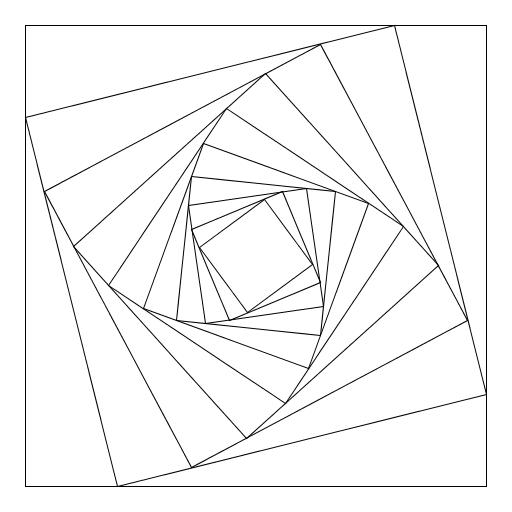
Lab 1 Report

# 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.

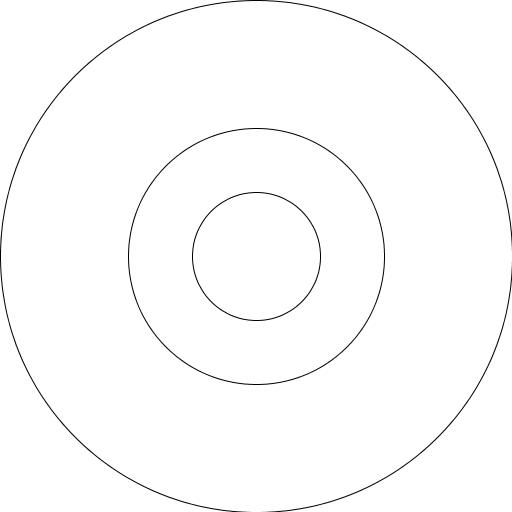
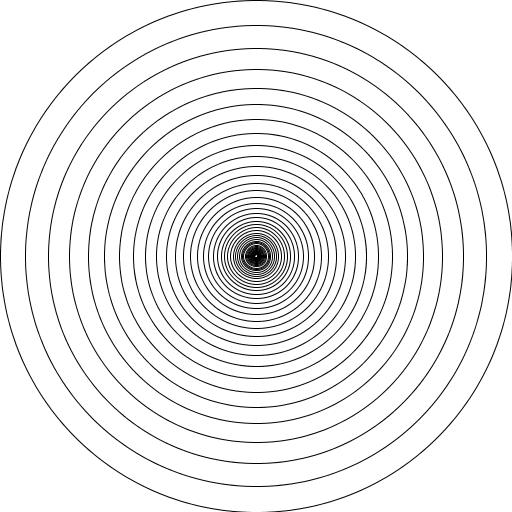
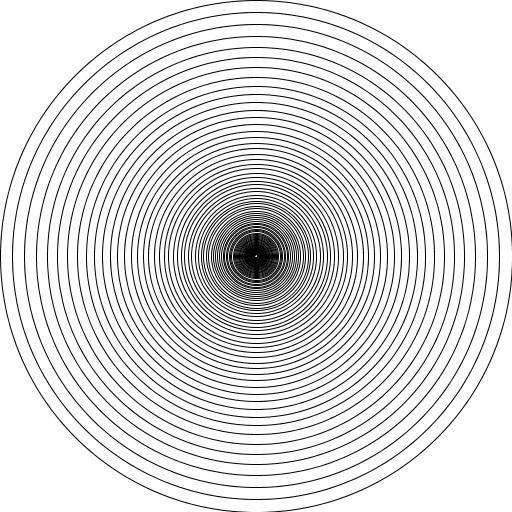
# Proposed Solution & Implementation

Problem 1)



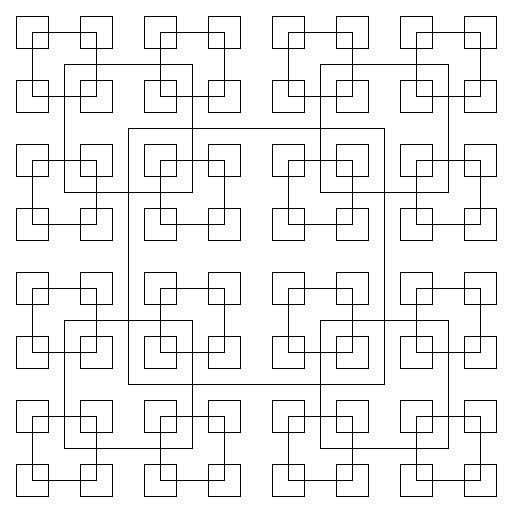
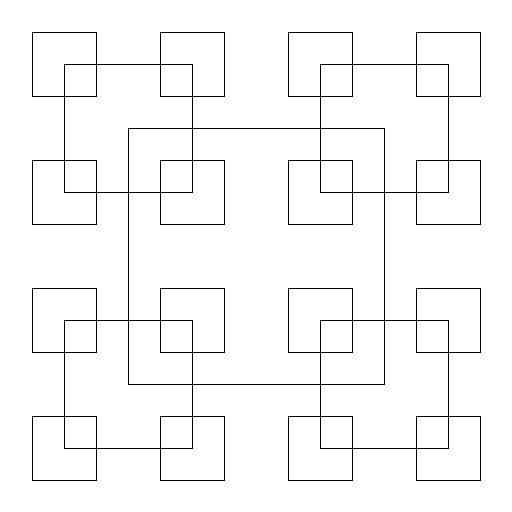
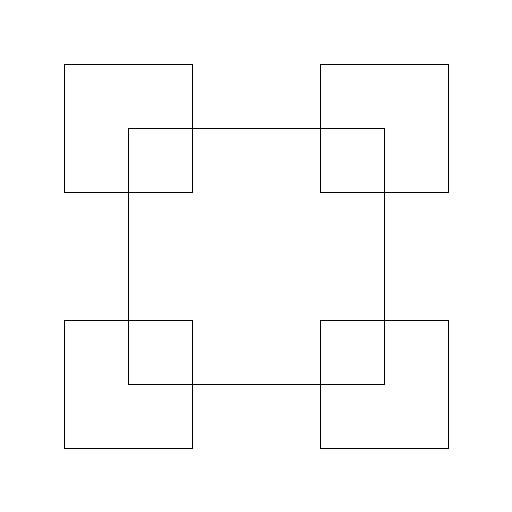
In this portion of the lab we were tasked in drawing the above shapes recursively.

Problem 2)

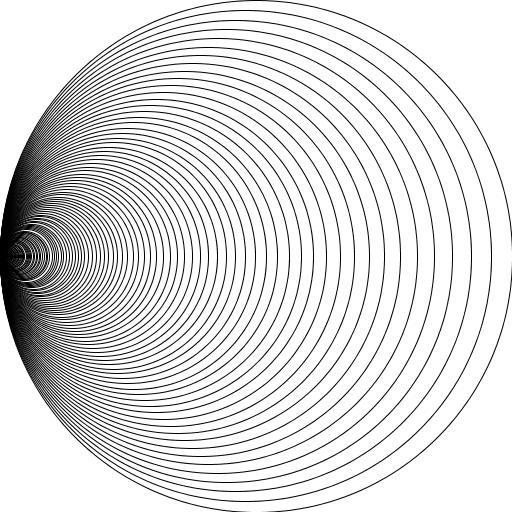
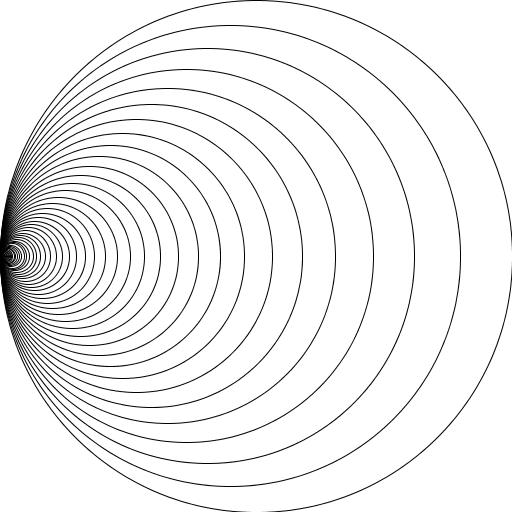
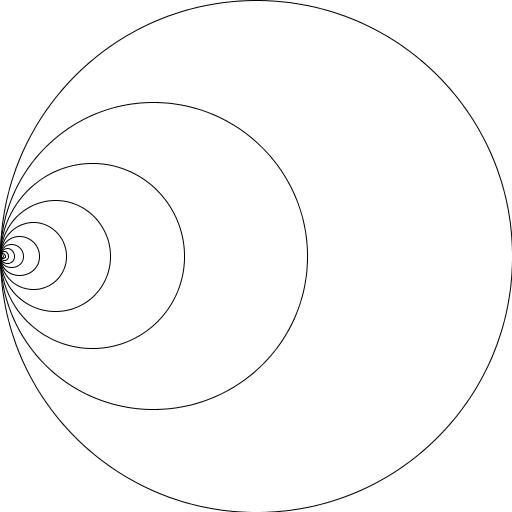
In this portion of the lab we were tasked in drawing the above shapes recursively.

Problem 3)



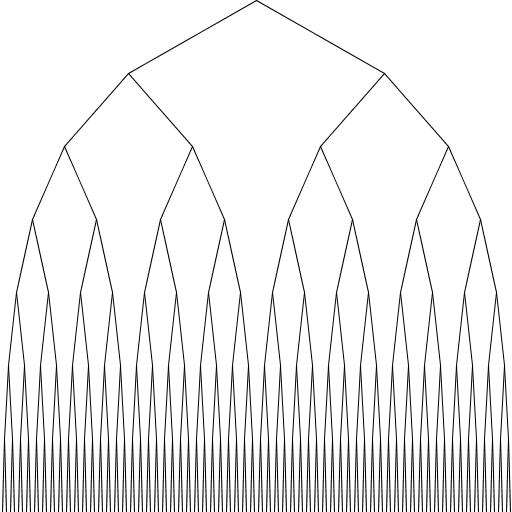
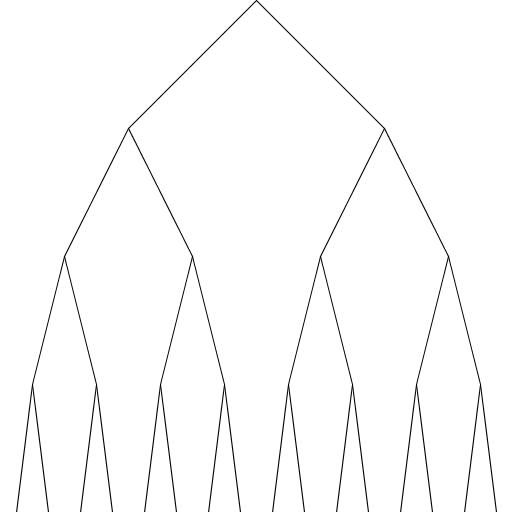
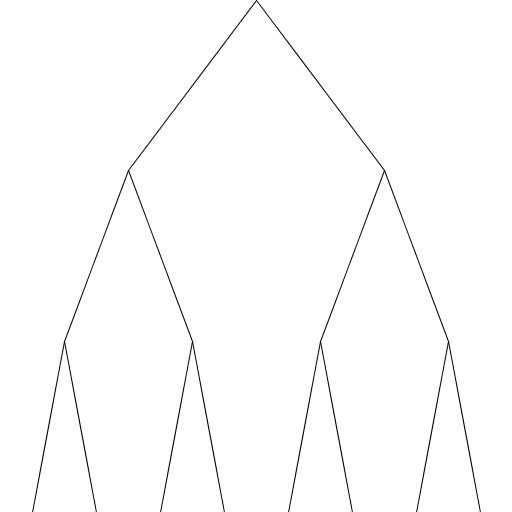
In this portion of the lab we were tasked in drawing the above shapes recursively.

Problem 4)



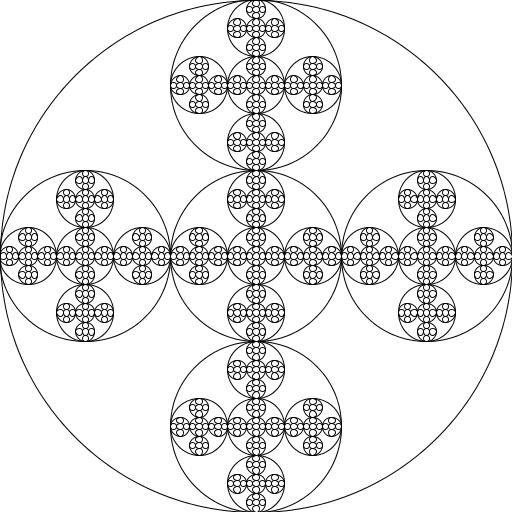
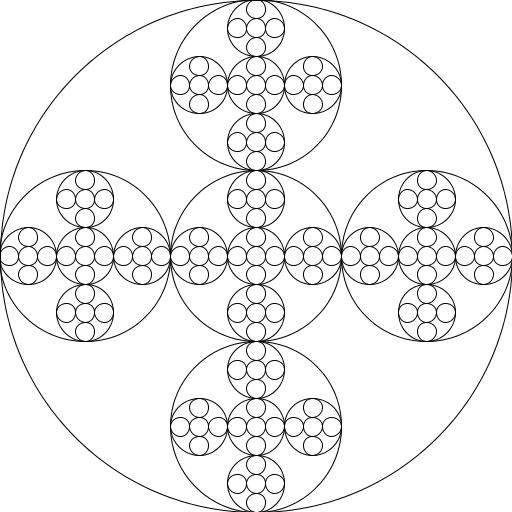
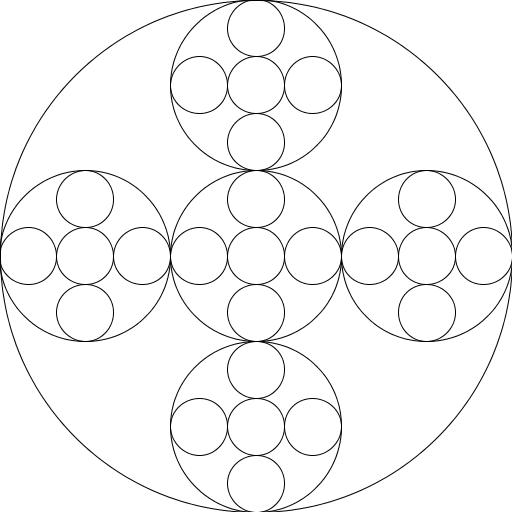
In this portion of the lab we were tasked in drawing the above shapes recursively.

Problem 5)



In this portion of the lab we were tasked in drawing the above shapes recursively.

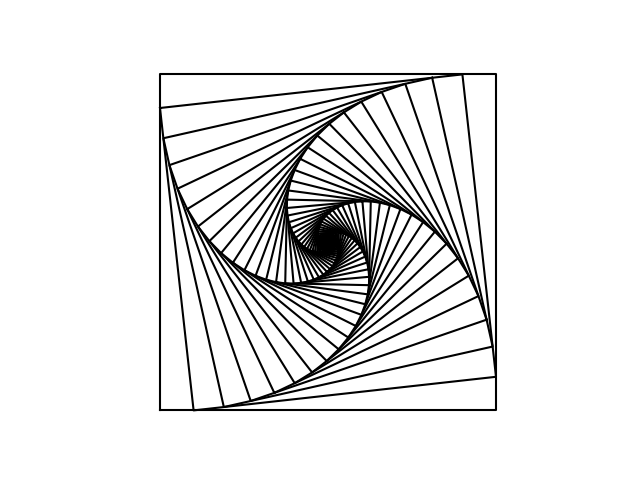
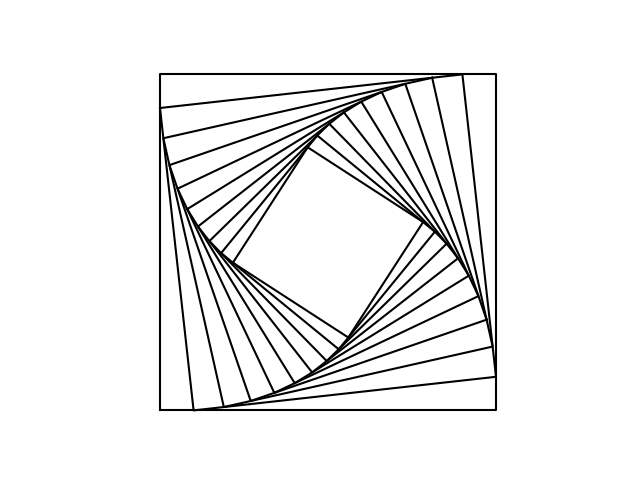
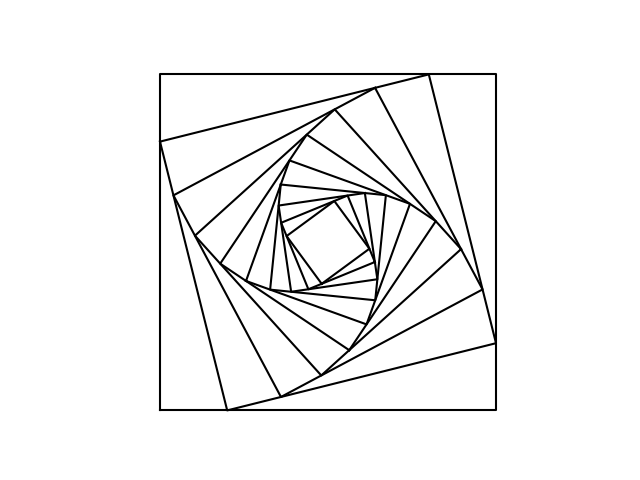
Problem 6)



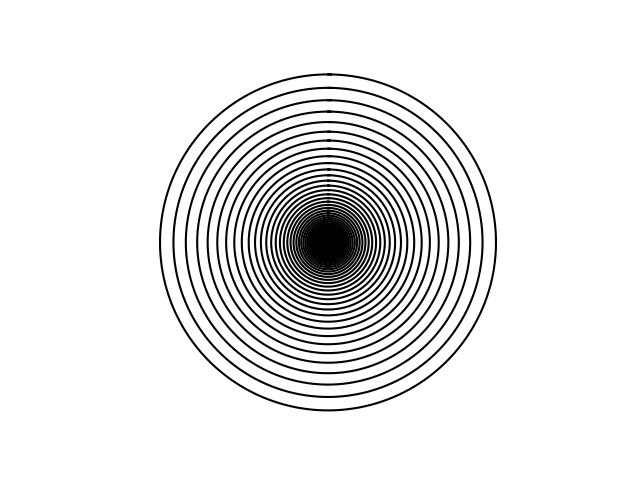
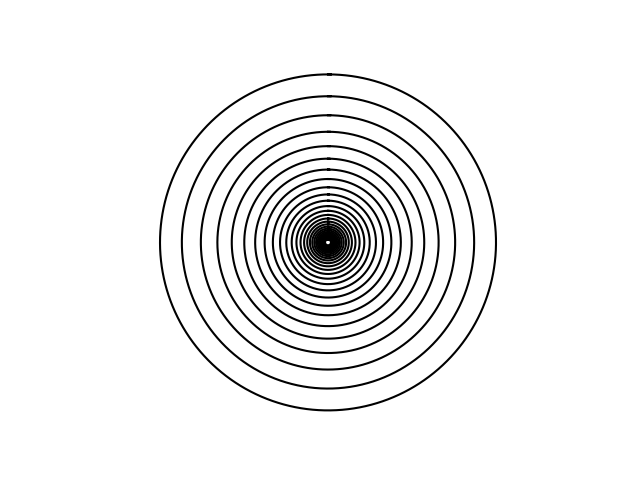
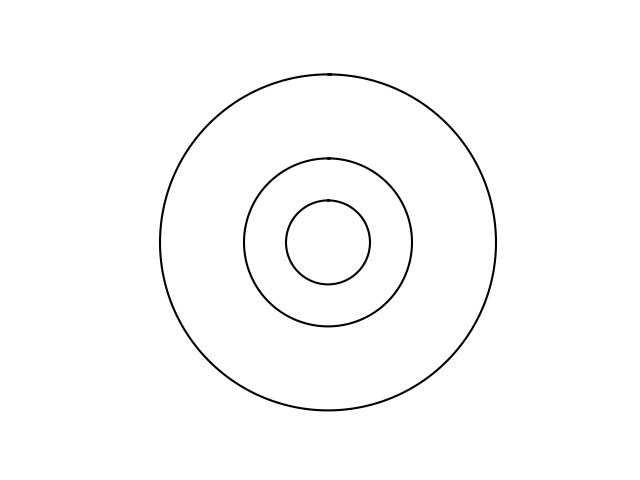
In this portion of the lab we were tasked in drawing the above shapes recursively.

# Experimental Results

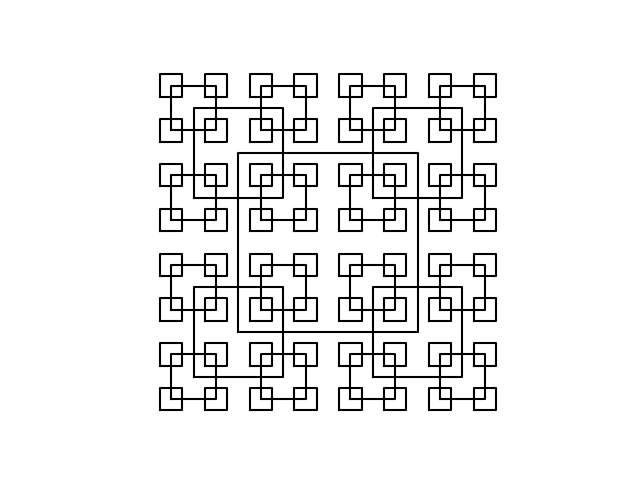
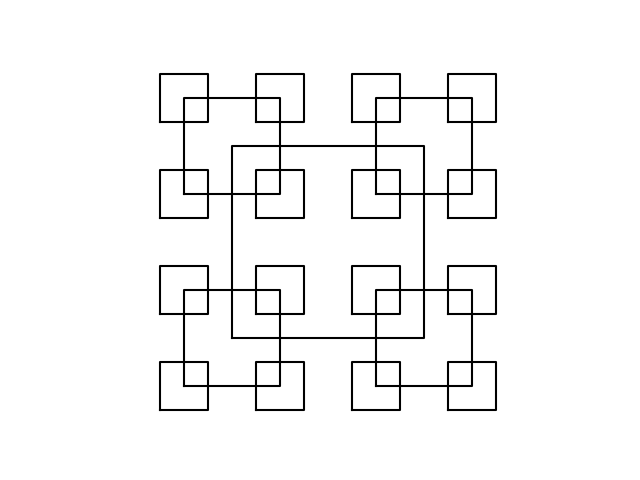
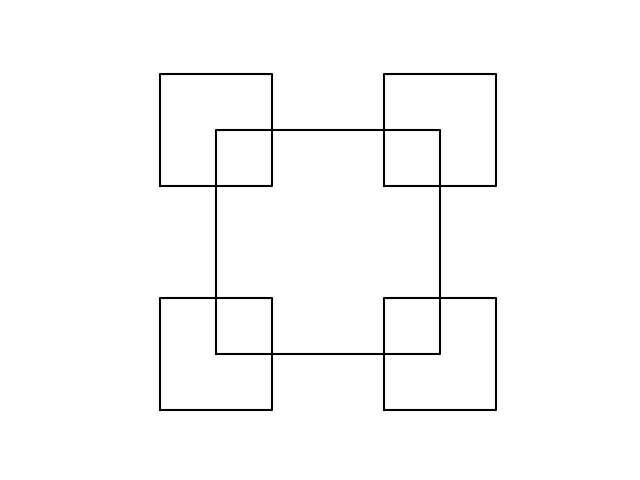
Problem 1)



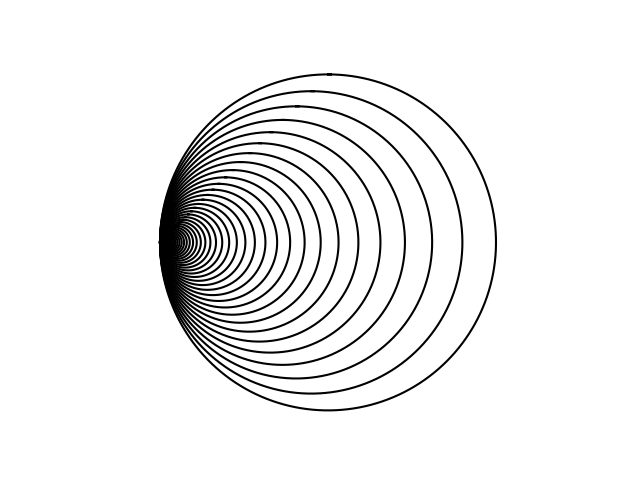
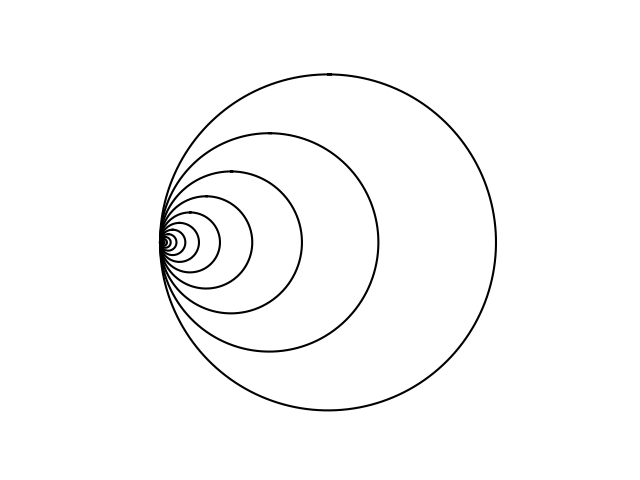
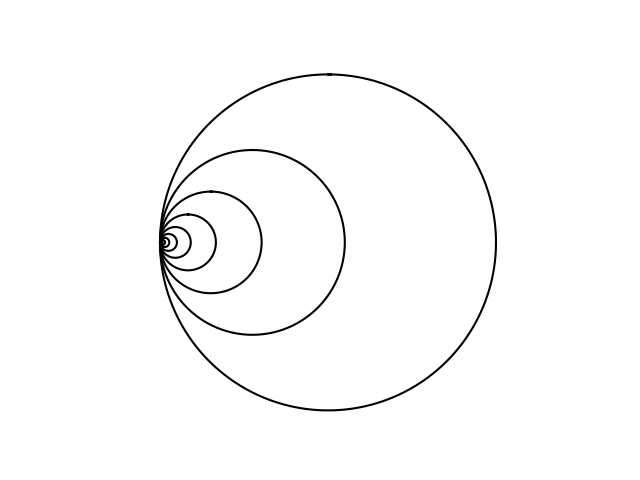
Problem 2)



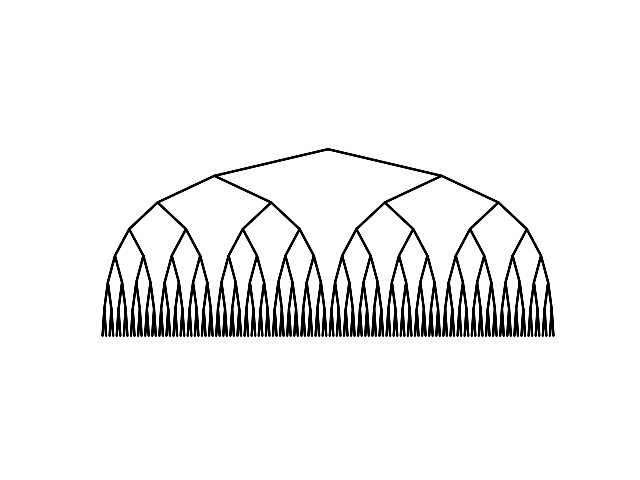
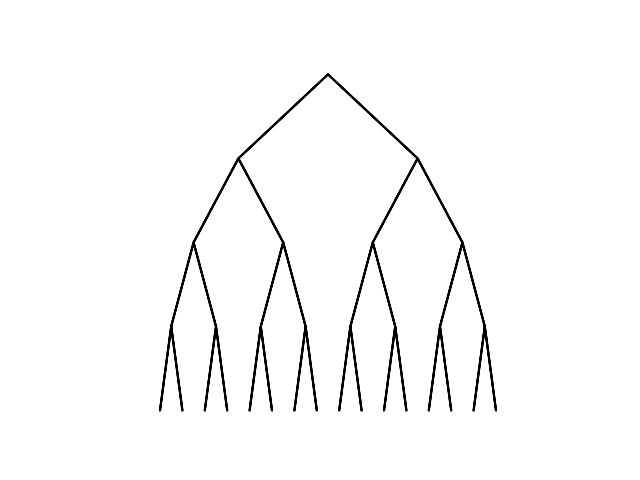
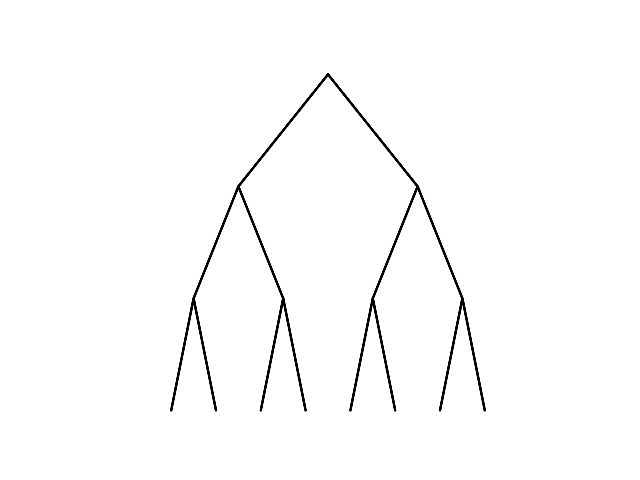
Problem 3)



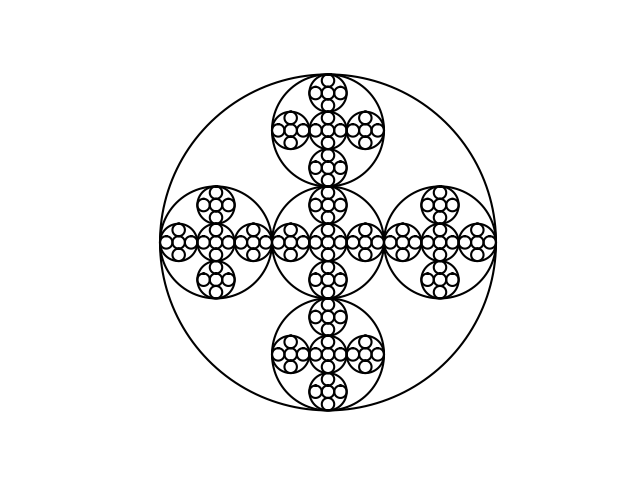
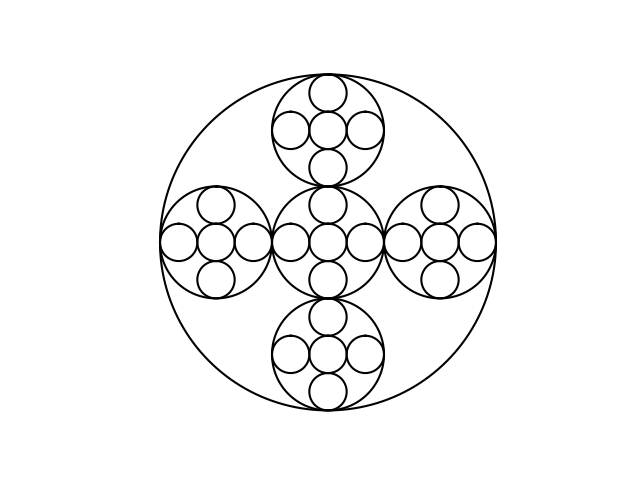
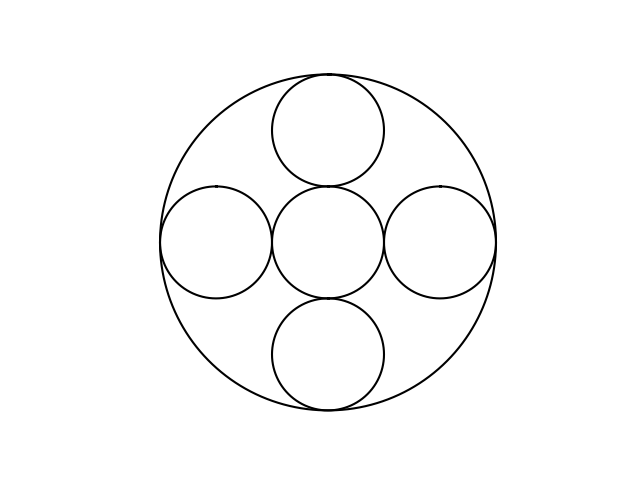
Problem 4)



Problem 5)



Problem 6)



# Conclusion

# 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('on') 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('on') 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('on') 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('on') 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('on') 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('on') ax.set\_aspect(1.0) drawMultiCirlces(ax,3, [100,100], 100) plt.show() fig.savefig('Lab1\_Output\_Images/lab1\_6c.png') |