Robert Marc

80487972

CS 2302

MW 1:30-3:00

Lab 1 Report

Introduction

The purpose of this lap was to take sample recursive code that drew squares and circles and modify them in such a way as to draw new shapes and patterns as described in the lab problem sheet. The first sub-problem was to write code that draws a square, and then a square at each corner, continuing for as many levels as requested. The second sub-problem was to draw a circle with a number of circles within its boundary that progressively got smaller and moved over to the left. The third problem was to draw a branching binary tree. The fourth and final problem was to draw a circle with five circles arranged in a plus sign within it, and five circles arranged in a plus sign within those, and so on.

Proposed Solutions

For the first problem, I took the original code given to us and just added an additional argument to the based method that defined the size of the square to be drawn. Within the draw\_squares method, I have a while loop within a for loop that sets up the square’s coordinate array to be used in the next call. The program draws the center square, then draws the squares clockwise starting from the top left corner. It moves to the requested depth before moving back up to draw the next square.

The second problem uses a simple formula to determine the weight for shrinking based on the number of circles to be drawn. It then calculates the difference in radius between the current circle and the next circle to be drawn and shifts the center of the next circle over by that difference so that the left edge of each circle matches up.

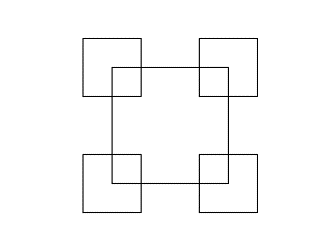
The third problem was solved by dividing the total size of the figure by the number of levels in the binary tree to determine each branches height. It then finds the positive and negative midpoints for the left and right branches and draws those, then builds the tree to the required depth starting with the leftmost branch of each level then moving back up to draw the right branch for the level above it, then back down to continue the left side. It repeats this process until the whole tree is drawn.

The fourth problem was solved by using the example code from the second problem and the solution for the first problem as a basis for this solution. The main method draws a circle at the requested position and radius. The way the recursive calls work for this program has the large central circle drawn first, then it draws the central circle of the ‘plus’ sign first, all the way to the bottom, completes the bottom-most ‘plus’ sign, then moves up a level to the draw the northern most, back down to draw the ‘plus’, and then back up. It moves clockwise around the ‘plus’ sign design, moving up and down, completing the design in segments.

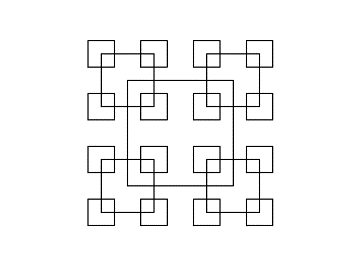
Experimental Results

Problem 1

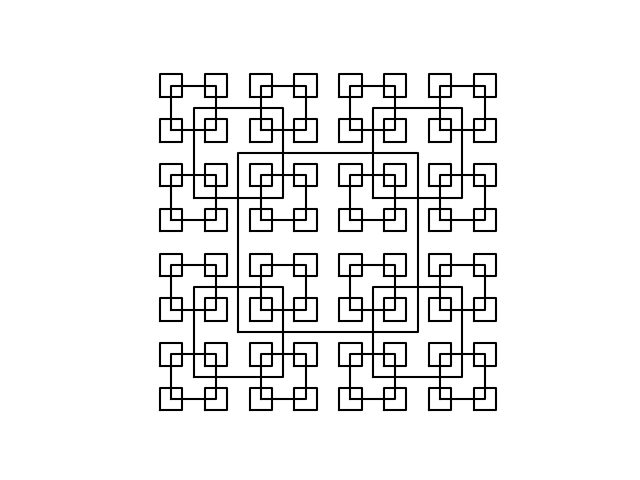
d = 2



d = 3



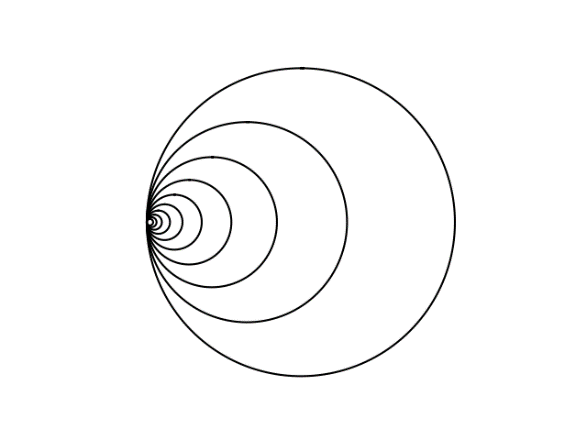
d = 4



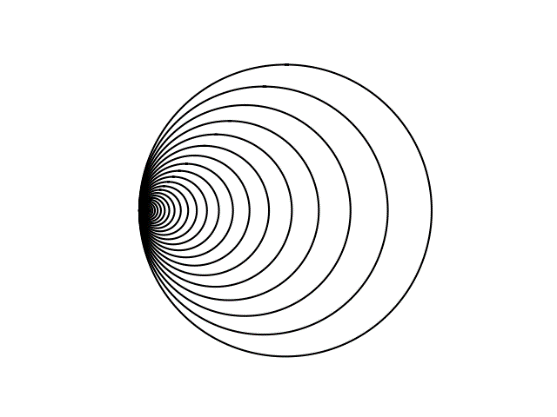
Any input greater than this saw exponentially longer run times. D = 6 takes nearly 30 seconds to complete.

Problem 2

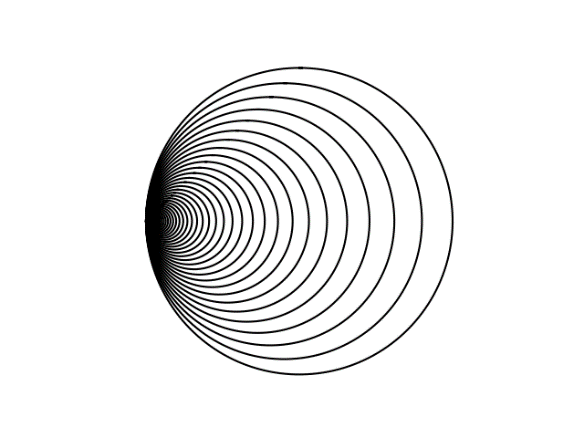
c = 10



c = 30

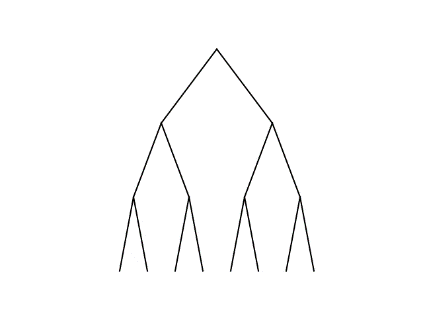


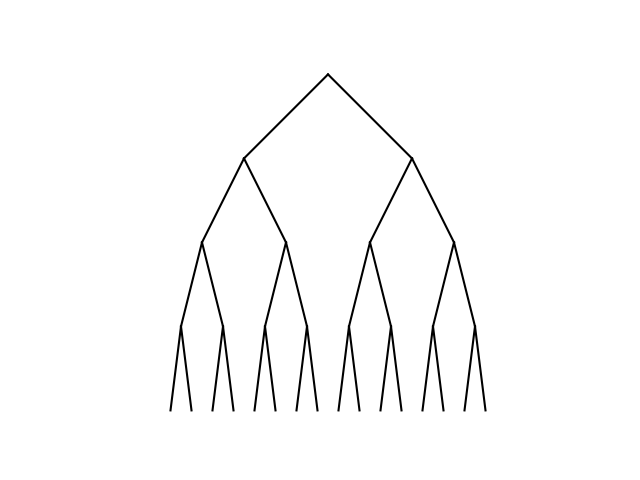
c = 50

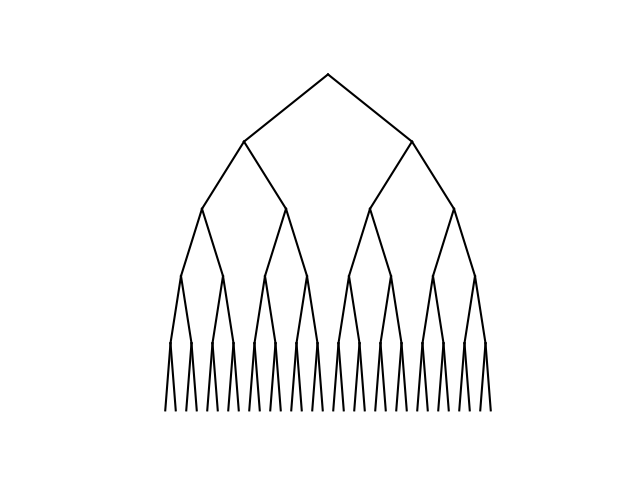


Due to the way the weight is calculated, the program hangs at any value of c greater than 78. The program will eventually complete, but my computer frequently thinks that the program has frozen.

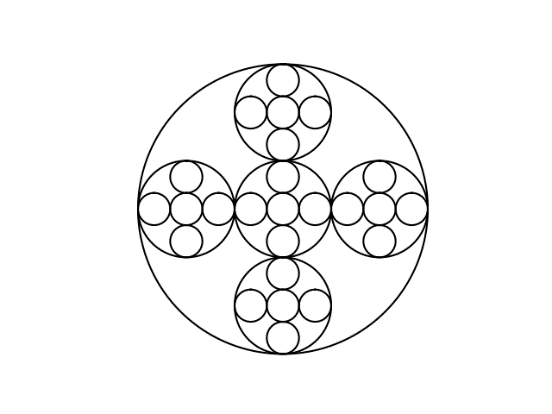
Problem 3

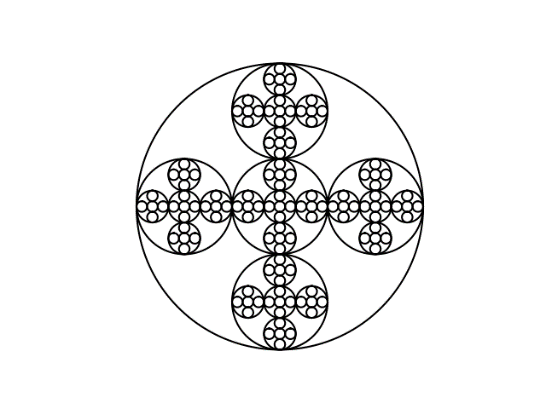
d = 3

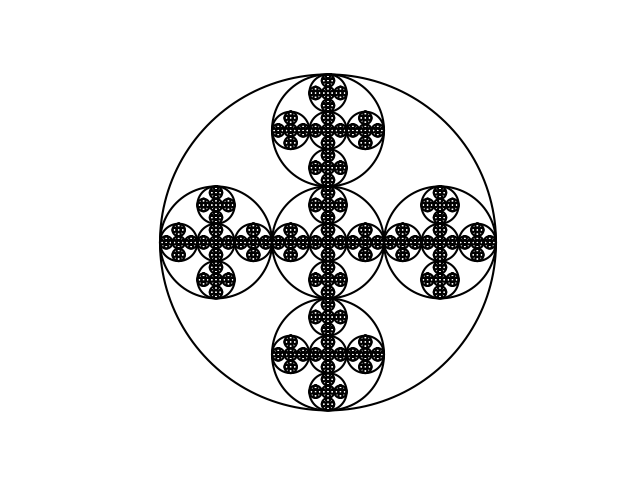
d = 4

d = 5

Problem 4

d = 3

d = 4

d = 5

Values of d greater than 5 starts to run into a similar problem that the solution to problem 1 had.

Conclusions

In this lab, I learned how to properly implement recursive calls. It was interesting that the problems that I thought I would have the greatest issue with were the ones that I very quickly found solutions to, but that the ones that seemed simple at first were far more arduous tasks. I also learned that methods that use recursive calls can be written in many different ways, but that it all relies on a solid base case to function properly.

Appendix

Problem 1 Code – draw\_squares.py

import numpy as np

import matplotlib.pyplot as plt

def draw\_squares(ax,d,p,s):

if d > 0:

ax.plot(p[:,0],p[:,1],color = 'k')

shrink = s/4

j = len(p)-2

for i in range(4):

while j >= 0:

nd = np.array([[p[j,0]-shrink,p[j,1]-shrink],[p[j,0]-shrink,p[j,1]+shrink],

[p[j,0]+shrink,p[j,1]+shrink],[p[j,0]+shrink,p[j,1]-shrink],

[p[j,0]-shrink,p[j,1]-shrink]])

draw\_squares(ax,d-1,nd,s/2)

j -= 1

plt.close("all")

orig\_size = 1000

p = np.array([[0,0],[0,orig\_size],[orig\_size,orig\_size],[orig\_size,0],[0,0]])

fig, ax = plt.subplots()

draw\_squares(ax,4,p,orig\_size)

ax.set\_aspect(1.0)

ax.axis('off')

plt.show()

fig.savefig('squares.png')

Problem 2 Code – draw\_circles.py

import matplotlib.pyplot as plt

import numpy as np

import math

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 draw\_circles(ax,n,center,radius,w):

if n>0:

x,y = circle(center,radius)

ax.plot(x,y,color='k')

center[0] = center[0]-(center[0]-center[0]\*w)

draw\_circles(ax,n-1,center,radius\*w,w)

plt.close("all")

fig, ax = plt.subplots()

c = 80

w = .5 + (c/200)

if w < 1:

draw\_circles(ax,c,[100,0],100,w)

ax.set\_aspect(1.0)

ax.axis('off')

plt.show()

fig.savefig('circles.png')

else:

print("Number of circles has caused the weight of radius reduction to be too high. Please choose a smaller number.")

Problem 3 Code – draw\_tree.py

import matplotlib.pyplot as plt

import numpy as np

def draw\_tree(ax,n,c,w,l):

if n>0:

x1 = c[0]-(w/4),c[0]

x2 = c[0]+(w/4),c[0]

y = c[1]-l,c[1]

ax.plot(x1,y,color='k')

ax.plot(x2,y,color='k')

draw\_tree(ax,n-1,[x1[0],y[0]],w/2,l)

draw\_tree(ax,n-1,[x2[0],y[0]],w/2,l)

plt.close("all")

orig\_size = 1000

c = [0,orig\_size]

d = 5

fig, ax = plt.subplots()

draw\_tree(ax,d,c,orig\_size,orig\_size/d)

ax.set\_aspect(1.0)

ax.axis('off')

plt.show()

fig.savefig('tree.png')

Problem 4 Code – draw\_nested\_circles.py

import matplotlib.pyplot as plt

import numpy as np

import math

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 draw\_nested\_circles(ax,n,center,radius):

if n>0:

x,y = circle(center,radius)

ax.plot(x,y,color='k')

new\_r = radius\*(1/3)

draw\_nested\_circles(ax,n-1,center,new\_r)

draw\_nested\_circles(ax,n-1,[center[0],center[1]+(2\*new\_r)],new\_r)

draw\_nested\_circles(ax,n-1,[center[0],center[1]-(2\*new\_r)],new\_r)

draw\_nested\_circles(ax,n-1,[center[0]+(2\*new\_r),center[1]],new\_r)

draw\_nested\_circles(ax,n-1,[center[0]-(2\*new\_r),center[1]],new\_r)

plt.close("all")

fig, ax = plt.subplots()

d = 5

draw\_nested\_circles(ax,d,[100,0],100)

ax.set\_aspect(1.0)

ax.axis('off')

plt.show()

fig.savefig('nested\_circles.png')