Rigoberto Quiroz

2/12/2019

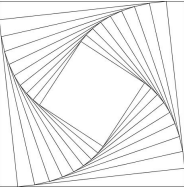
Lab1 Report

CS 2302 1:30PM – 2:50OM MW

Description:

For this lab I had to draw certain figures multiple times using recursion. Affecting their position and size to construct the desired image. Some of the figures are squares, or circles as shown in the example below.

EX:



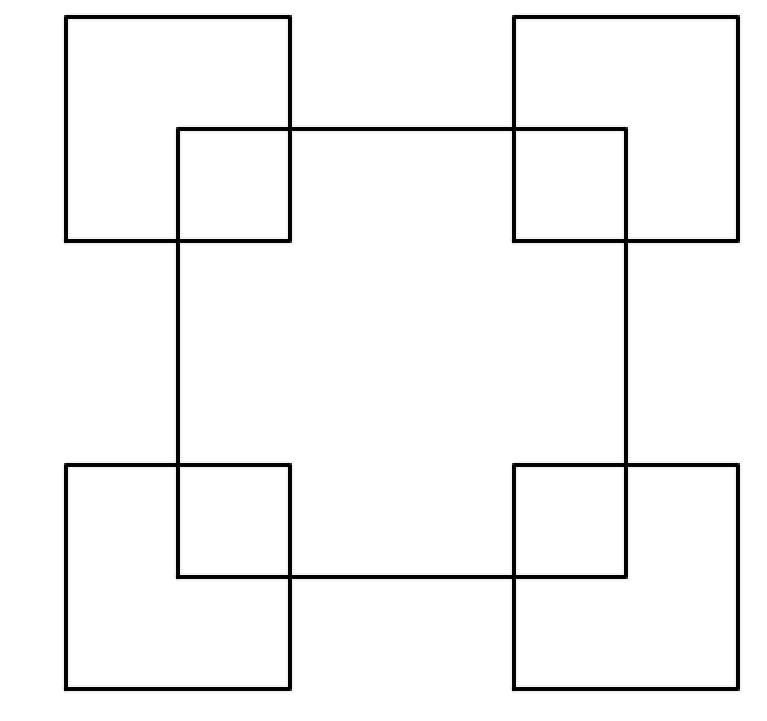
The way I was able or tried to solve this lab was by using a midpoint which I could reference. Then in each recursion call I would affect the coordinates of the following figures using my midpoint references, making my figures move to a desired direction or changing their size. In each recursion call I would update my midpoint to a new midpoint where a new figure would be drawn. For some of my methods (binaryTree(), draw\_Squares()) I would affect their midpoints and size using addition or subtraction, meanwhile in others methods I would affect their midpoints and size using multiplication or division.

**1 A – C**

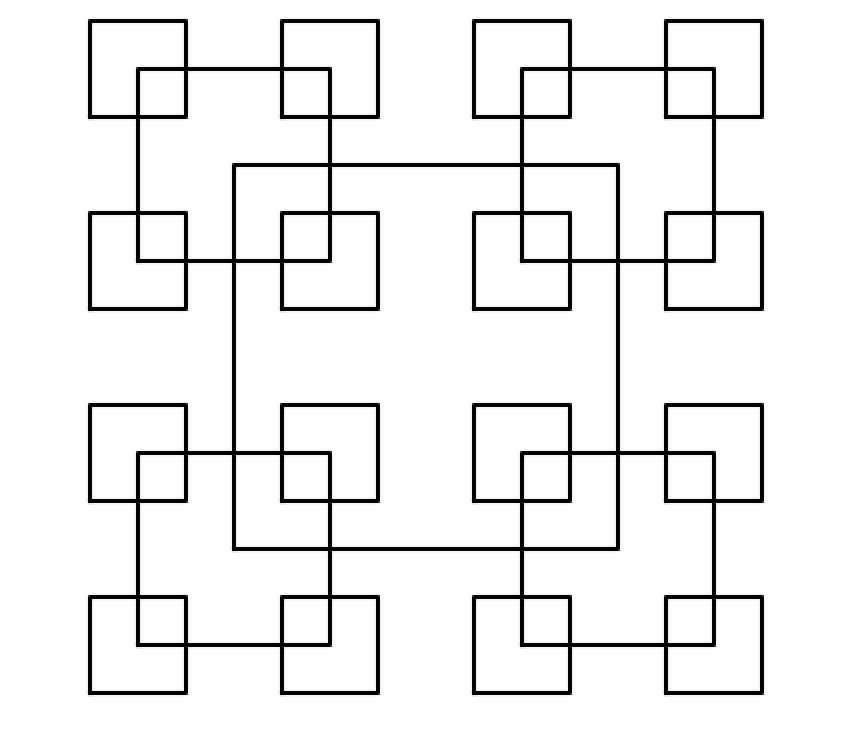
For the first problem, the program will ask the user how many recursion calls will the user make (1,2,3,etc.), then the program is going to store the number of recursions. The program will then call the method, “draw\_squares()”, and draw squares (midpoint is already defined in the program). The program will draw the figure according to the midpoint and make 4 recursion calls, each with a new midpoint (corners of the previously drawn square), subtract the number of recursions by one and alter ‘w’(affects how big the squares are being drawn). It will repeat this process until the number of recurions the user stated reaches zero.

**Output:**

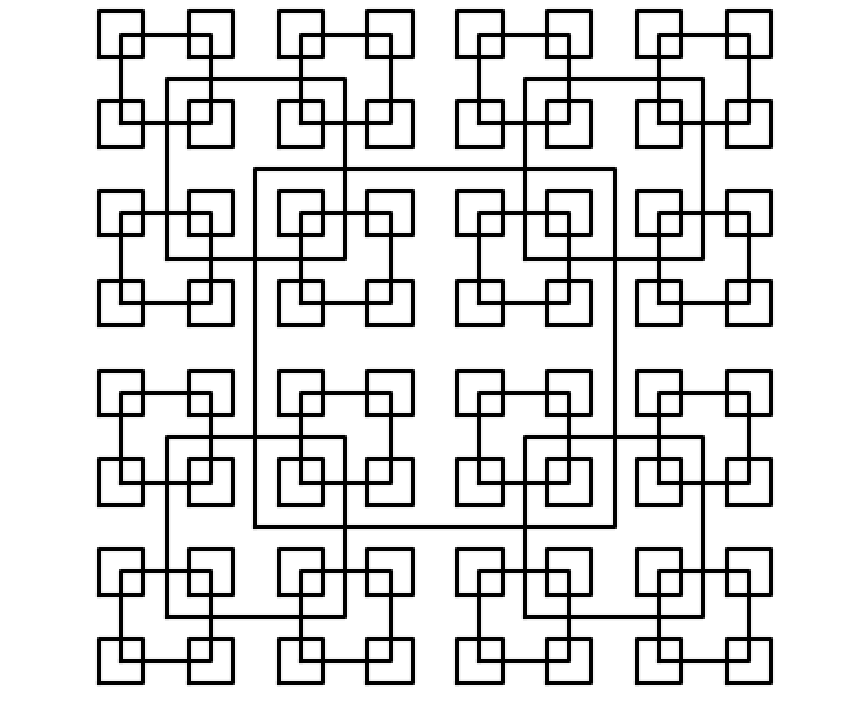
User input: 2



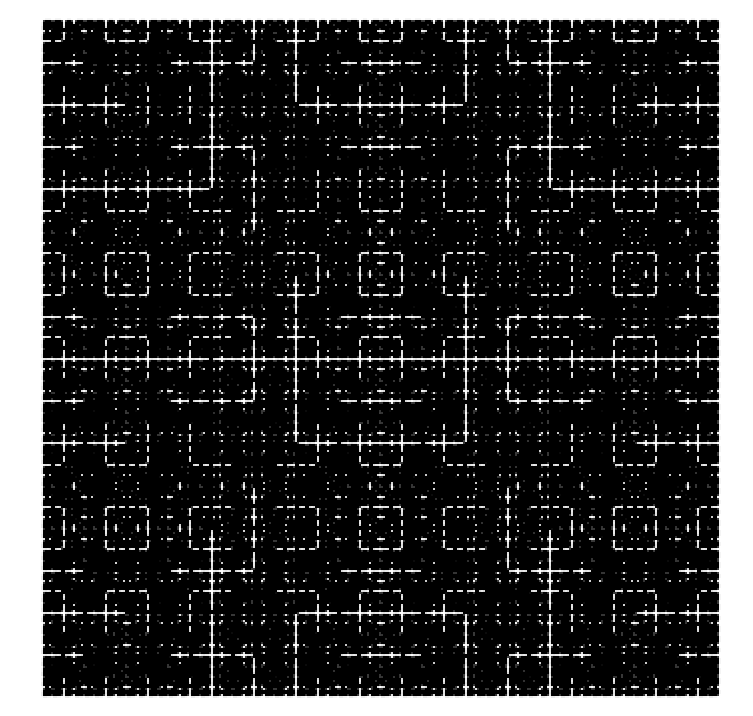
User input: 3



User input: 4



User input: >= 7

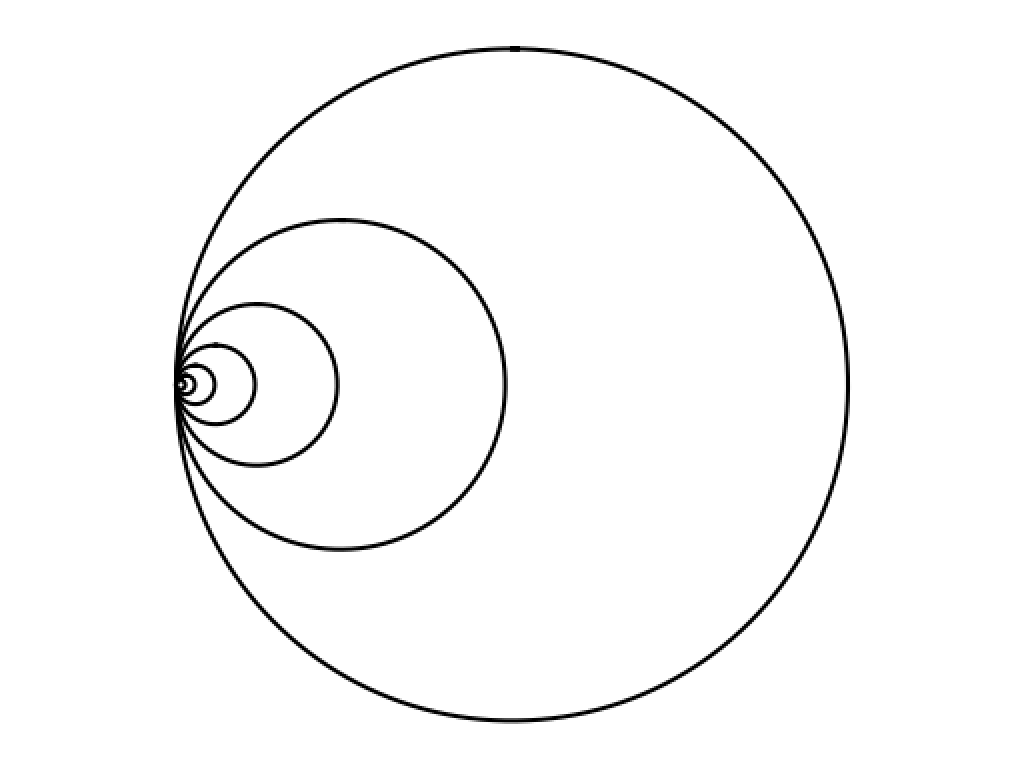


The greater the input(<7) the longer it will take to compute, since it would be drawing 4 squares at each corner of the original square. The image becomes more crowded with squares. The recursion cannot take negative numbers because the base case will never be reached, so it will never draw a square. The program will not take in doubles, strings or characters, an error will be outputted, it will only take integers as an input.

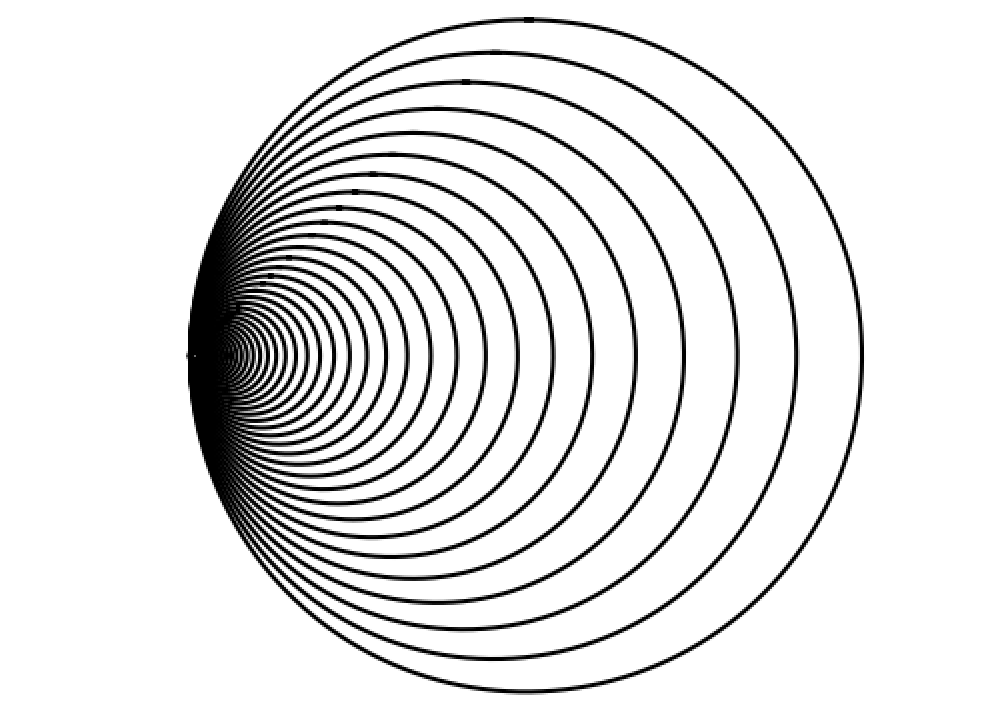
**2 A – C:**

The way I was able to solve this problem was by affecting the x coordinate and radius at the same rate. I did because the more calls we make, the smaller the inner circles will become. This is achieved by multiplying x coordinate by a small number to move it to the left. (No need to affect the y coordinate because it will move the inner circles up or down) Using that same small number we multiply the radius. This will cause the circles to move left and decrease at the same rate.

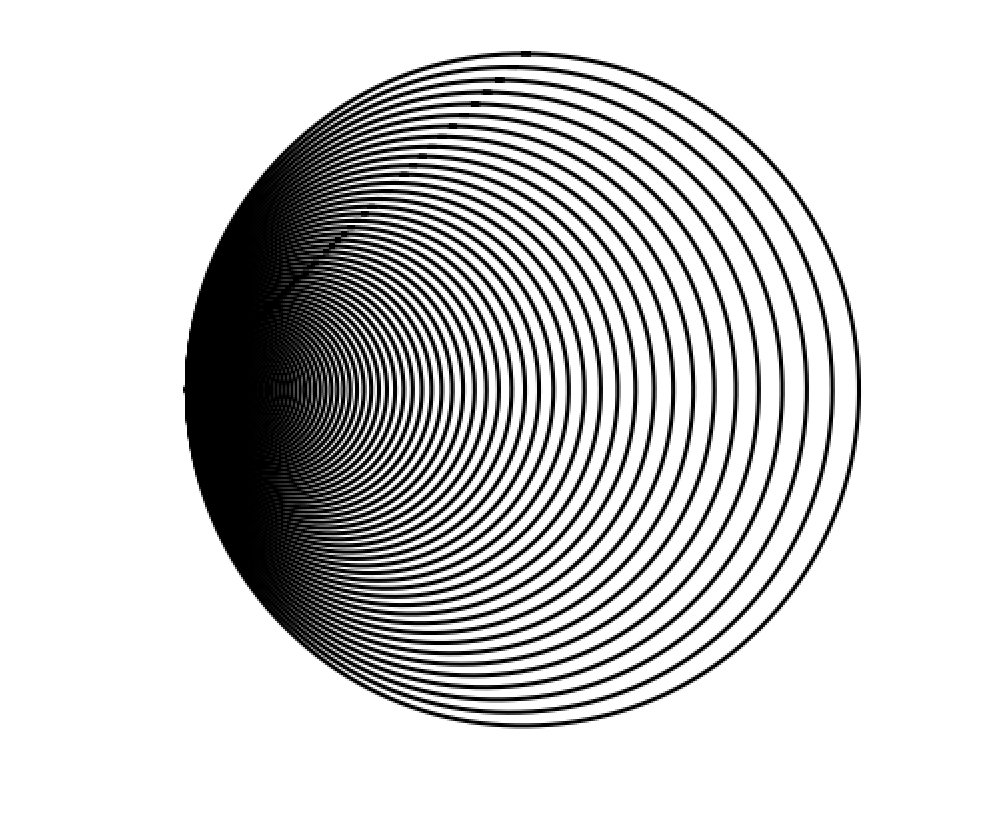
User Input: 7 and .7



User Input: 45 and .95



User Input: 220 and .98

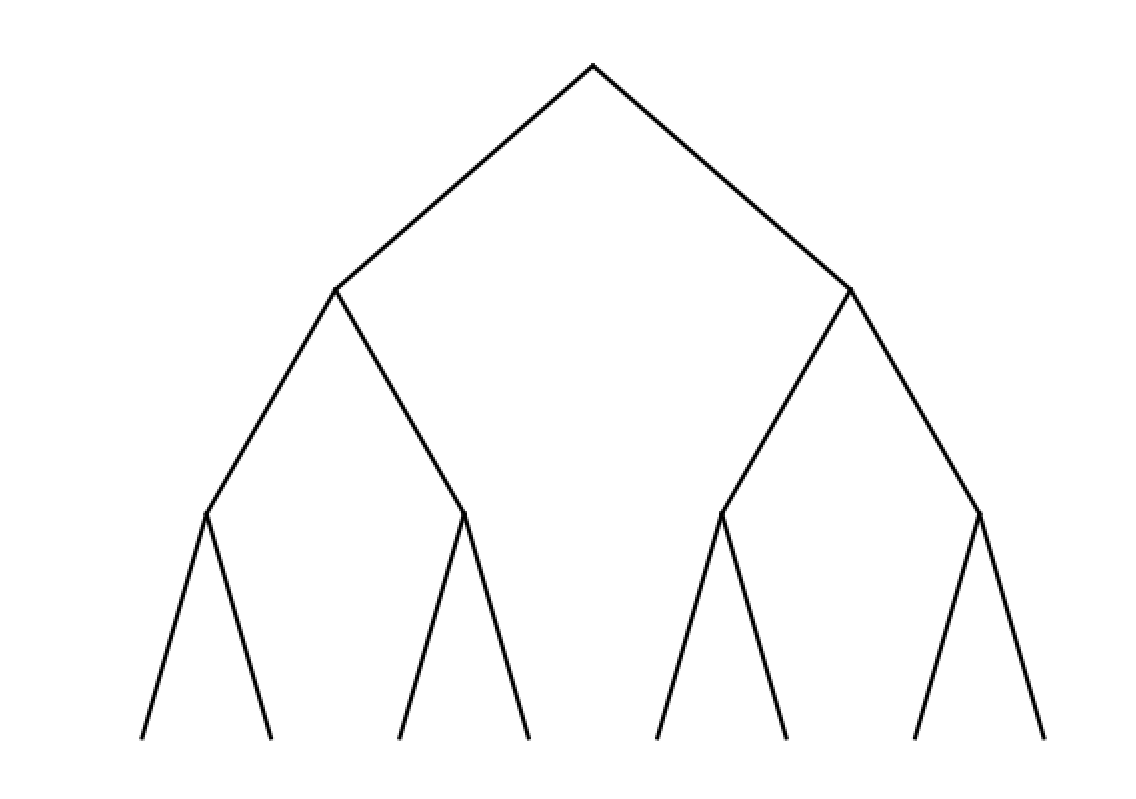


Depending how much the user affects the distance between each circle and how they are being moved to the left direction. The lower the value the more separated the circles become, the greater the less separated the circles become, as shown by the previous examples.

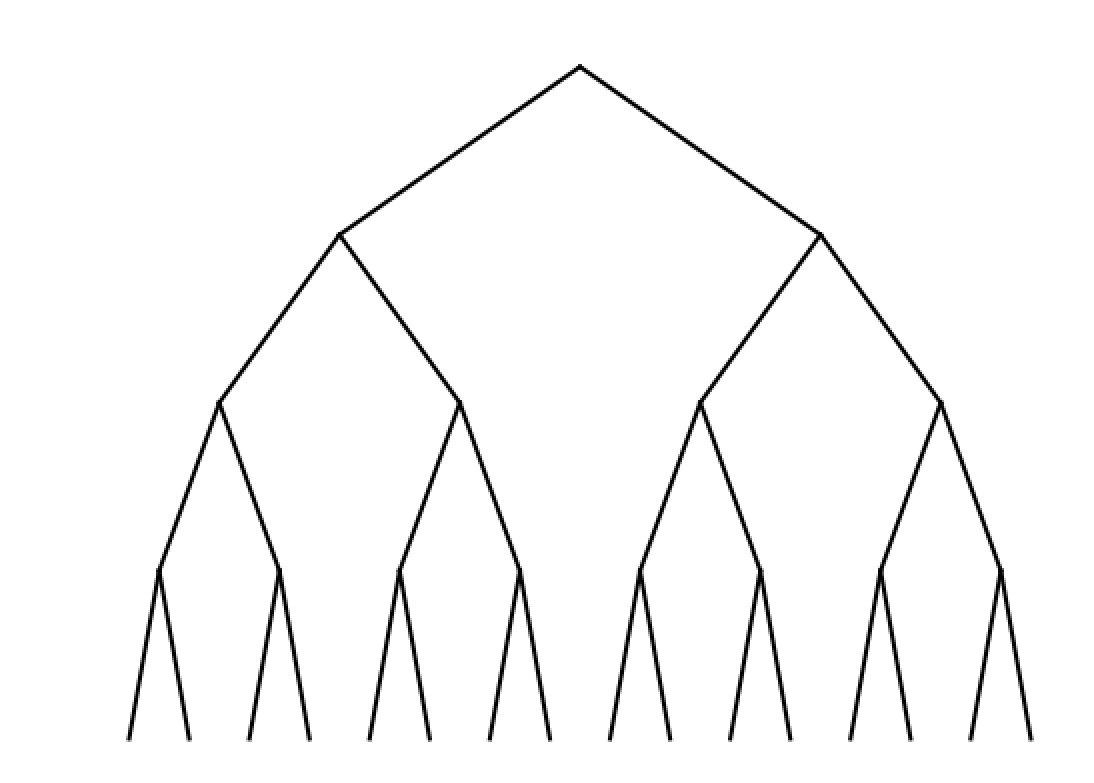
**3 A – C:**

The way I was able to solve this problem was by creating two variables (x and y) which is the center or the root of the tree. I also created two other variables that would change the direction of the branches being drawn and the angel in which they are being drawn, otherwise some of the branches would be generated start and would intersect at some point. While in the method, binaryTree(), it is going to draw a center point and the left branch, then it’s going to return back to the center point and draw the right branch. This method contains to recursive calls, one is affecting all the left branches, and the other affects the right branches. Each recursion call will take a new center point, depending on which branch it is(left or right), then, it will also take other variables so that the branches do not intersect with each other, affecting how much a branch moves to the left or to the right.

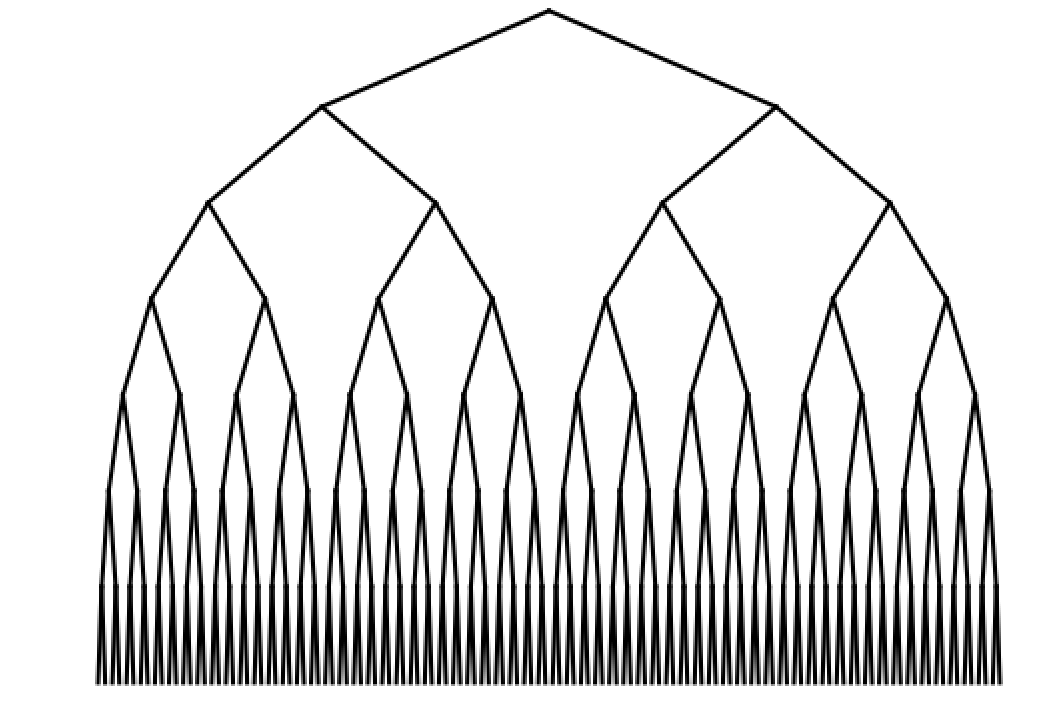
User Input: 3



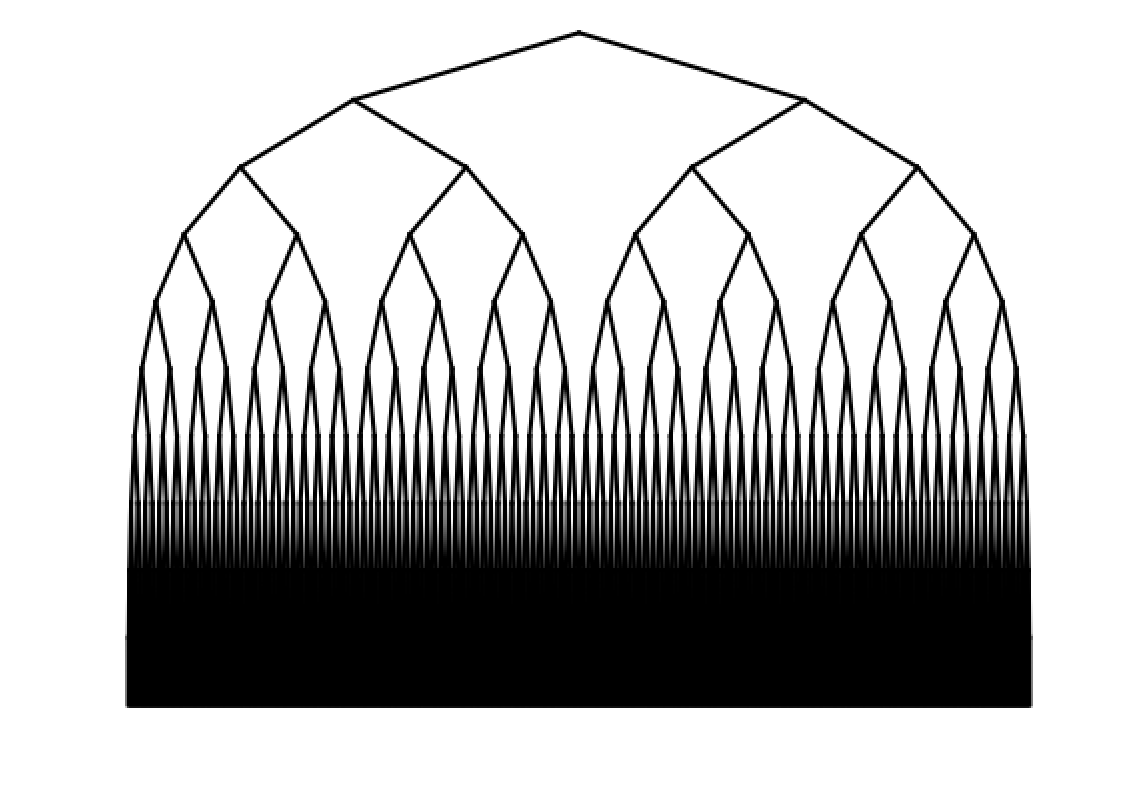
User Input: 4



User input: 7



User input: >= 10

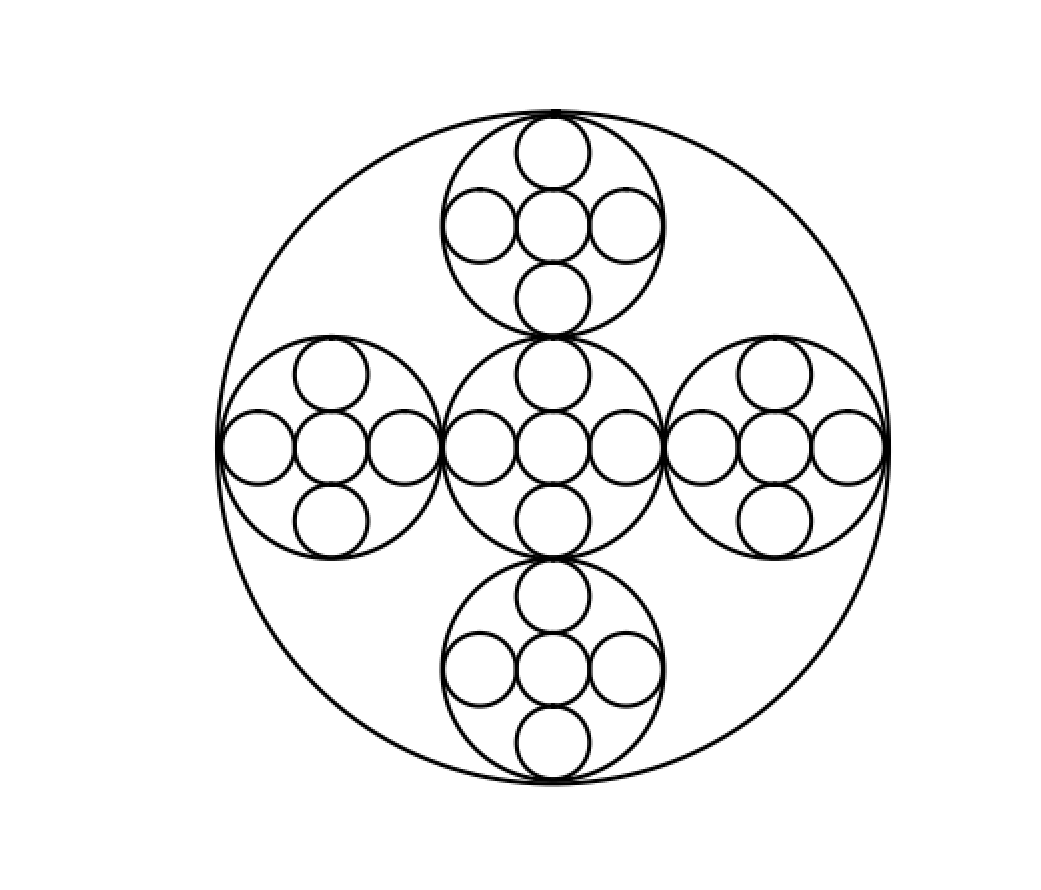


The greater the recursion call in (user input), the more branches will be drawn, the output is produced more slower and as you can notice from the image above, you can no longer see branches at the bottom because they are so many. If the user inputs a negative number of recursion calls then the tree will not be drawn because of the condition. If the user inputs strings, characters, or doubles the program will generate an error. The number of recursion calls must be an integer.

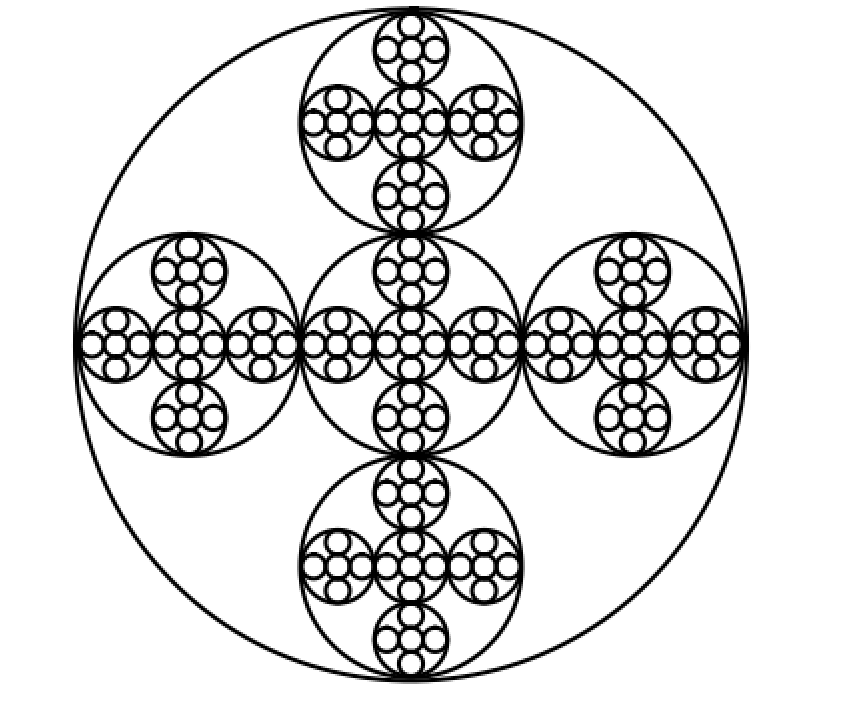
**4 A – C**

The way I was able to solve this problem was by creating a midpoint and radius which the original circle will be drawn. Then the program will ask the user how may calls they will make (Number of recursions). We will send all this information to out method, “draw\_circles”, which will draw the circle with the information provided, and then make four recursion calls, each call affecting five inner circles, forming a T like form. The way I was able to move those five inner circles was by affecting the original center point of the big circle. Since we have five circles inside, each circle must be the same size and be separated by the same distance. I separated the base circle in the parts, horizontally and vertically, each inner circle have one third(1/3) of space.

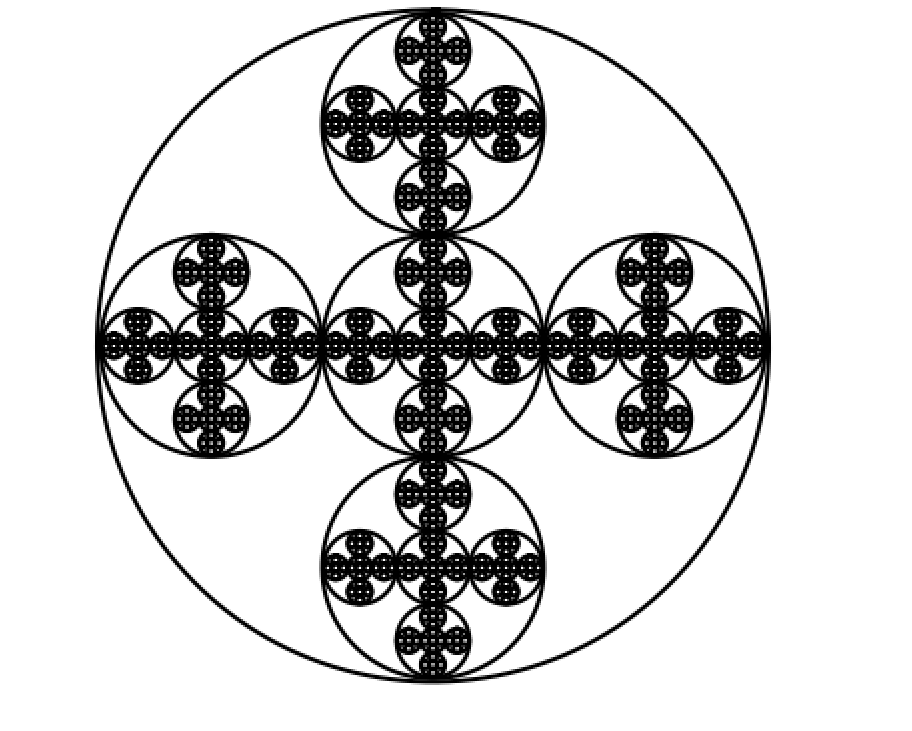
User Input: 3



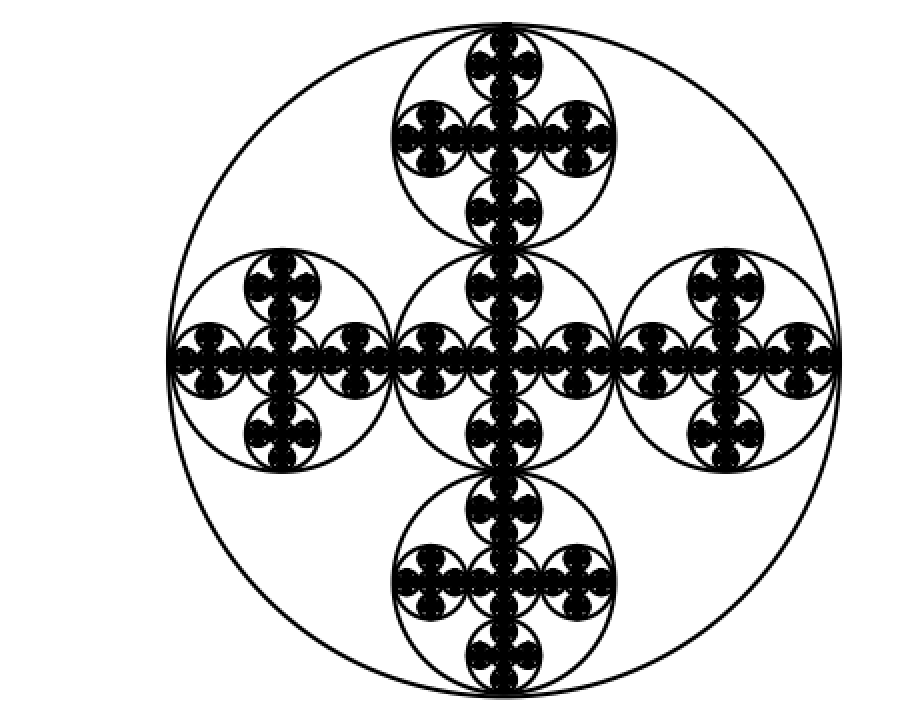
User Input: 4



User Input: 5



User Input: >=6



Once the user input gets to 6 or greater than the computer begins to take longer to compute the output and the circles become more darker towards to center due to the amount of circles that are being produced. If the user input is a negative number than the circle will not be drawn because of our base condition, and

What I learn from this lab or take away was a better understanding of recursion works. Better knowledge of matplotlib library and numpy for arrays. Although I found it difficult to implement arrays into my recursion and for most of my cases I used single variables that represented my midpoint and worked from there. I noticed that the more recursions we had the longer the computer took to compute the output because of their different runtimes each method has. The big take away was from this lab and the way I was able to solve these problems was by having a midpoint or a reference point which you can affect and use to set up future recursion points.

**Appendix:**

# Lab 1

# By: Rigobeto Quiroz

# Class: 1:30 PM - 2:50 PM MW

# This program draws squares at the corners of other square using recursion. The

# greater number of recursions the more squares the program is going to draw. Each

# recursion call > 1 creates 4 squares at the corners of the already sepcified squares.

# 1.A 2 5

# 1.B 3 5

# 1.C 4, 5

import numpy as np

import matplotlib.pyplot as plt

def draw\_squares(ax,x,y,n,w):

if n>0:

#Draws square according to midpoint(x,y)

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

#Bottom left Square

draw\_squares(ax,(x-w),(y-w),n-1,(w/2))

#Top left Square

draw\_squares(ax,(x-w),(y+w),n-1,(w/2))

# Top Right Square

draw\_squares(ax,x+w,y+w,n-1,(w/2))

# Bottom Right Square

draw\_squares(ax,x+w,y-w,n-1,(w/2))

plt.close("all")

fig, ax = plt.subplots()

# x,y - midpoint of Square

x = 0

y = 0

calls = int(input('Enter number of recusion calls?'))

draw\_squares(ax,x,y,calls,5)

ax.set\_aspect(1.0)

plt.axis('off')

plt.show()

#2A: 7, .7

#2B. 45, .95

#2C. 220, .98

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')

# Moves the center of the circle to the left. Only affects indexs 0, b/c of the x direction

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

# Makes Circle smaller

radius = radius \* w

# Each recusion call the circle becomes smaller and moves further left

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

plt.close("all")

calls = int(input('Please enter number of recursion calls.'))

# affects radius and how much the circles move to the left

rateOfChange = float(input('Please enter the rate of which the circles are going to be affected.'))

fig, ax = plt.subplots()

draw\_circles(ax,calls, [100,0], 100,rateOfChange)

ax.set\_aspect(1.0)

ax.axis('off')

plt.show()

fig.savefig('circles.png')

# 3A. n = 3

# 3B. n = 4

# 3C. n = 7

import matplotlib.pyplot as plt

import numpy as np

def binaryTree(ax,x,y,dx,dy,n):

if n > 0:

#Draws root and left branch

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

#Draws root and right branch

ax.plot((x,x+dx),(y,y-dy),color='k')

#Recursion calls for right and left branches

binaryTree(ax,x-dx,y-dy,dx/2,dy,n-1)

binaryTree(ax,x+dx,y-dy,dx/2,dy,n-1)

plt.close("all")

calls = int(input('Enter number of recursion calls?'))

# x,y - coordinate for midpoints or root

x = 50

y = 100

# dx, dy - Affects the next figures Xand Y directions

dx = x / 2

dy = y \* calls

fig, ax = plt.subplots()

binaryTree(ax,x,y,x,dy,calls)

plt.axis('off')

plt.show()

import matplotlib.pyplot as plt

import numpy as np

import math

def circle(a,b,rad):

n = int(4\*rad\*math.pi)

t = np.linspace(0,6.3,n)

x = a+rad\*np.sin(t)

y = b+rad\*np.cos(t)

return x,y

def draw\_circles(ax,n,a,b,radius,w):

if n>0:

x,y = circle(a,b,radius)

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

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

draw\_circles(ax,n-1,a+(radius\*.66),b,radius\*.33,w)

draw\_circles(ax,n-1,a-(radius\*.66),b,radius\*.33,w)

draw\_circles(ax,n-1,a,b+(radius\*.66),radius\*.33,w)

draw\_circles(ax,n-1,a,b-(radius\*.66),radius\*.33,w)

plt.close("all")

fig, ax = plt.subplots()

calls = int(input('Enter number of recursion calls: '))

a = 100

b = 0

radius = 100

draw\_circles(ax,calls,a,b,radius,.33)

ax.set\_aspect(1.0)

ax.axis('off')

plt.show()

fig.savefig('circles.png')