Ian Rigsbee

80602795

CS2302 Olac Fuentes

Lab 1 report

**Introduction**

For the problem at hand we had to recreated four different shapes. A square that has four other squares in the corner that will recursively show up in all four corners, a circle with its center point shifted to the left, a circle that had 5 squares inside the circle that also have 5 circles inside those circles, and a binary search tree. We were given sample code that would aid us in order to get these shapes. One for squares that would spiral inside the original circle, and one for a circle that had more circles inside it that focused down to its center point.

**Design and Implementation**

For the squares I decided to make 4 new squares that would be in each corner of the original square. To do this I would add or subtract from the original square array and by doing so would shift the squares to a specific corner. As for the size of the squares I thought it would be best to divide the size of the square by three since dividing by two would make the squares touch each other. With these four squares I need four recursive calls, one for each square. While also decreasing the amount of times we go through recursion. I designed it so that I could use the same parameters as the sample code that was given to us instead of changing the parameters as that would lead to a more thoughtful design rather than the straightforward approach.

For the shifted circle problem I tried numerous approaches from messing with the center point that was being called, to the radius and w parameters; but what I found was the easiest approach was actually just adding the radius to x or subtracting from it. If I add to x with radius then the program will shift the center to the left, if I subtract then it will shift to the right. I also wanted to make sure to use the same parameters just so I could keep it simple along with using the same sample code method that was given to us.

Next was the circles with the five additional circles inside it. Right away from my experience with the squares problem I decided to try the same approach to the problem. What I decided to change were the parameters to have an x and y coordinate for the center point. I would then make a center variable to place those x and y coordinates inside of in order to use the circle() method given to us. Since there were five circles I needed five recursive calls. One for the middle circle, one for the left, one for the right, top, and bottom. From the experience from the 2nd problem I knew adding or subtracting from the x and y values would shift the circle. So in those recursive calls I decided to either add or subtract from the x and y depending on where I needed it to be. I would then divide the radius by three since theres three circles from left to right and top to bottom. then along with adding and subtracting I had to make the circles fit inside each other rather than hanging in the corners and such, I would multiply the radius by 2/3. I played around with it a bit by trying different values but 2/3 would best fit the problem. I designed the method in a way that I could still take advantage of the circle() while also keeping it simple with the parameters.

The binary search tree was possibly the hardest to do. I wanted to design it in a way that I would have two recursive calls and only one ax.plot. to do so I made the origin and upside down V for the origin and left and right children. From there I would I knew I needed to find the distance between the two points. So I took the right child and the origin and subtracted, then divided it by 2. I then needed to created the V’s for the left and right child so I decided to try it by just either subtracting everything by one or adding everything by one depending on the child. I needed two recursive calls. One for the left and one for the right. Sadly I didn’t get what I was intending to get. I still cant figure out why. I know I needed to use the distance I created but that would give me weird output that wasn’t what I needed.

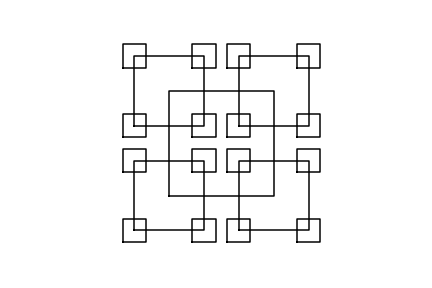
**Experimental Results**

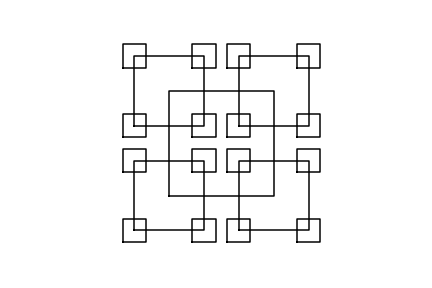
With squares I used the same array that was with the sample code and the same original size

It would look like this:

orig\_size = 100

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

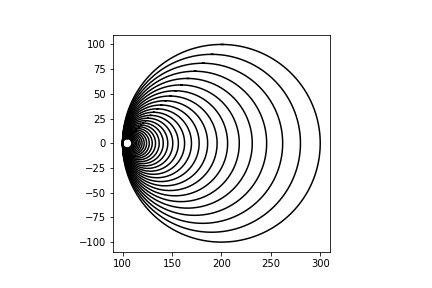
This gave me

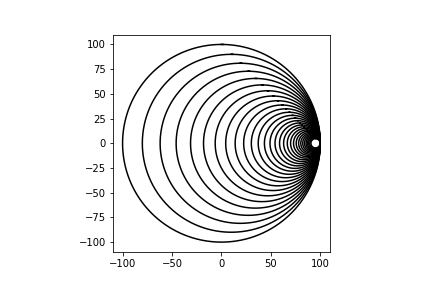
Trying again with orig\_size = 1000 gave me 

So even with an extremely large size the program will still function the same

With circles I had

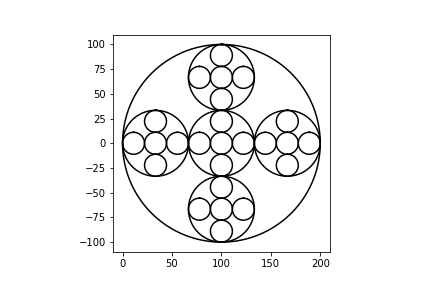
ax.plot(x+radius,y,color='k')



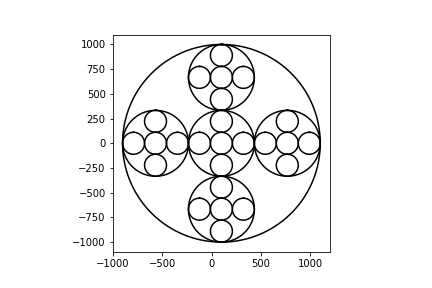
Doing the program again with ax.plot(x-radius,y,color='k') gives me

So it just confirms that it shifts the center point.

With circles the original input was

draw\_circles(ax, 3, 100,0, 100,.9) this will give

if I change the radius to a number greater than 100 I get

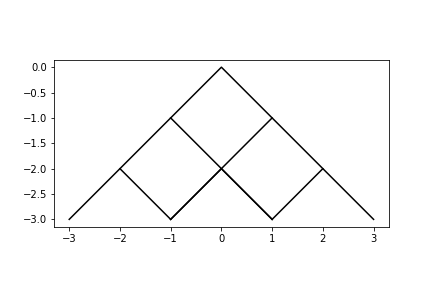
draw\_circles(ax, 3, 100,0, 1000,.9)

the program will still work the same even with a large radius

with the BST I couldn’t get it to look exactly like a BST but with the call

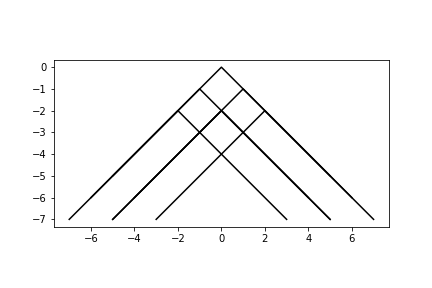
origin = np.array([[-1,-1],[0,0],[1,-1]])

drawbst(3,origin,ax)



If I change the points of the origin to

origin = np.array([[-5,-5],[0,0],[5,-5]])



The BST method doesn’t really work as intended.

**Conclusion**

This lab has really made me understand recursion in much greater detail as beforehand I would usually only use one recursive call and that would be it, but I needed to think more about how many calls were needed for the problem at hand. I also got a taste of plotting as I’ve never done this sort of thing before with recreating shapes. I just wish I got to understand the BST problem a little more then maybe I could’ve gotten it to look like a BST instead of what it is now. Although for someone who isn’t used to plotting points and using math in their code I have to say this was a very interesting lab.

**Appendix**

**Source code for the squares problem**

import matplotlib.pyplot as plt

import numpy as np

#for the method i decided to use p and change its values for each corner

#so for p[0][1] and p[1][1] I would make a new array from those two points

#and add or subtract. only problem was getting the newsize of the square

def draw\_squares(ax,n,p,originalsize):

if n>0:

#new distance

newsize = originalsize/3

#four arrays that plot the new squares on the corners of the original square and the recursive squares

q = np.array([[p[0,0]-newsize,p[0,1]-newsize],[p[0,0]-newsize,p[0,1]+newsize],[p[0,0]+newsize,p[0,1]+newsize],

[p[0,0]+newsize,p[0,1]-newsize],[p[0,0]-newsize,p[0,1]-newsize]])

s = np.array([[p[1,0]-newsize,p[1,1]-newsize],[p[1,0]-newsize,p[1,1]+newsize],[p[1,0]+newsize,p[1,1]+newsize],

[p[1,0]+newsize,p[1,1]-newsize],[p[1,0]-newsize,p[1,1]-newsize]])

o = np.array([[p[2,0]-newsize,p[2,1]-newsize],[p[2,0]-newsize,p[2,1]+newsize],[p[2,0]+newsize,p[2,1]+newsize],

[p[2,0]+newsize,p[2,1]-newsize],[p[2,0]-newsize,p[2,1]-newsize]])

l = np.array([[p[3,0]-newsize,p[3,1]-newsize],[p[3,0]-newsize,p[3,1]+newsize],[p[3,0]+newsize,p[3,1]+newsize],

[p[3,0]+newsize,p[3,1]-newsize],[p[3,0]-newsize,p[3,1]-newsize]])

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

draw\_squares(ax,n-1,q,newsize)

draw\_squares(ax,n-1,s,newsize)

draw\_squares(ax,n-1,o,newsize)

draw\_squares(ax,n-1,l,newsize)

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,3,p,orig\_size)

ax.set\_aspect(1.0)

ax.axis('off')

plt.show()

fig.savefig('squares.png')

**Source code for the shifted circles problem**

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

#simply adding the radius to x or y will shift it in a certain direction

#adding radius to x will make the circle center shift to the left

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

if n>0:

x,y = circle(center,radius)

#shifts to the left or right. doing same thing to y shifts it up or down

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

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

plt.close("all")

fig, ax = plt.subplots()

draw\_circles(ax, 30, [100,0], 100,.9)

ax.set\_aspect(1.0)

#ax.axis('off')

plt.show()

fig.savefig('shiftedcircles.png')

**Source code for the five circles problem**

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)

z = center[0]+rad\*np.sin(t)

c = center[1]+rad\*np.cos(t)

return z,c

#this method takes x and y instead of center but will be turning those two points

#into a center to use the circle() function. the method makes 5 smaller circles within

#the original circle. one in the middle,to the left,right,up and down. the cirlces will have their

#x and y coordinates either added or subtracted with radius \*2/3 simply because having

#1/3 will move the cirlces closer to the middle point which i dont want

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

if n>0:

#created center for use of circle()

center = [x,y]

z,c = circle(center,radius)

ax.plot(z,c,color='k')

#adding and subtracting to shift the circles within the original circle

draw\_circles(ax,n-1,x,y,radius/3,w)

draw\_circles(ax,n-1,x+radius\*2/3,y,radius/3,w)

draw\_circles(ax,n-1,x,y+radius\*2/3,radius/3,w)

draw\_circles(ax,n-1,x-radius\*2/3,y,radius/3,w)

draw\_circles(ax,n-1,x,y-radius\*2/3,radius/3,w)

plt.close("all")

fig, ax = plt.subplots()

draw\_circles(ax, 3, 100,0, 1000,.9)

ax.set\_aspect(1.0)

#ax.axis('off')

plt.show()

fig.savefig('circles.png')

**Source code for the BST**

import matplotlib.pyplot as plt

import numpy as np

# tried making it like squares where I can just change up some of the values to better fit

#the upside down V for the tree but i got stuck. i created the distance but whenever used it just

#makes the graph look out of place.

def drawbst(n,p,ax):

if n>0:

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

#distance = (p[2,0]-p[1,0])/2

# add and subtract values to get the children

leftchild = np.array([[p[0,0]-1,p[0,1]-1],[p[1,0]-1,p[1,1]-1],[p[2,0]-1,p[2,1]-1]])

rightchild = np.array([[p[0,0]+1,p[0,1]-1],[p[1,0]+1,p[1,1]-1],[p[2,0]+1,p[2,1]-1]])

drawbst(n-1,leftchild,ax)

drawbst(n-1,rightchild,ax)

plt.close("all")

fig, ax = plt.subplots()

origin = np.array([[-5,-5],[0,0],[5,-5]]) #upside down V for tree origin

drawbst(3,origin,ax)

ax.set\_aspect(1.0)

#ax.axis('off')

plt.show()

fig.savefig('BST.png')