# CS2302 Lab 1

Introduction – The problem of this lab is: learn to create the shapes given and tackle that problem in our own way using our problem-solving skills, and practice solving these problems using recursion. The given problems go from experimenting with shapes, to create new shapes using our own skills.

Proposed solution design and implementation, problems 1 – The solution I came up with for the problem given to us was 4 recursive calls for every square in every corner. The problem declared x and y variables, instead of using an array and use 4 as the rate of change. The recursive method had the same base case as the square file given to us (if n>0), and will allow the code to run an n amount of times and update the n-1, and will stop once n = 0. In the method we created xl(left side of the x-axis), xr(right side of the x-axis), yu(up side of the y-axis), and yd(down side of the y-axis). We used the rate of change to update the location of x and y, for xl we subtracted r from x to get the left side of the x-axis, for xr we added r to x to get the right side of the x-axis, for yu we added r to y to get the top value of the y-axis, and finally for yd we subtracted r from y to get the bottom value of the y-axis. After getting our corner values, we plot them into the graph, and we call the recursive calls. The recursive calls are 4 and each is different because they call each corner, along with calling a different corner, they all update r and divide it by 2 to make the squares smaller, and they also update n to decrease each time a recursive call is called.

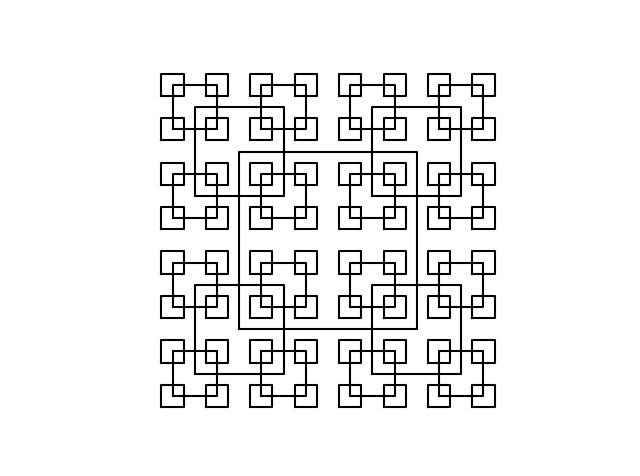
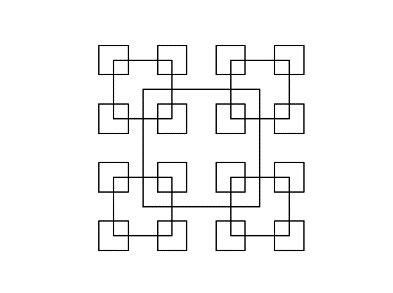
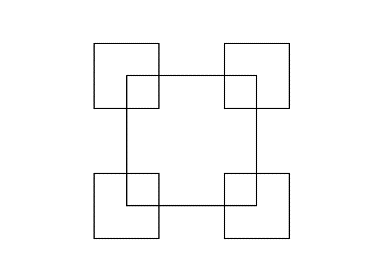
Proposed solution design and implementation, problems 2 – The next problem that was given to us, required us to update the location of the circle to remain on the left side of the x-axis. Using the circle file given to us, I updated only 2 thing to get my solutions. The things that were updated in this problem was center[0] and radius. The reason this 2 were updated was because center[0] is our x-axis of the circle, and radius determines the size. This time we used w to represent our rate of change, and what we did was multiply it twice to center[0] and radius. Having multiplied it twice, allowed the x-axis to remain almost unchanged, and for the radius, it also allowed it to decrease the size of the circle. After doing that, we called our recursive call n amount of times, and allowed us to do it because of our base case(if n>0) and (n-1) in the recursive call.

Proposed solution design and implementation, problems 3 – This problem was a binary tree and was definitely the one I had most fun with. This problem began with me initializing the first 2 points which were x and y. x was set equal to a number(can be any number) and y was set equal to x\*3 because it was 3 times the size as x in the image. After that, we called r our rate of change and set that equal to x/2, because that allowed us to make the tree smaller as we went down n amount of times. Once we reach the method, we plot twice, once for the right side of the tree, and again for the left side of the tree. In the plotting, we plotted differently because of the different sides, left was (x, x-r), (y, y-5), and right was (x, x+r), (y, y-5), the x-r got us the left side of the x-axis, while the x+r got us the right, and we plotted x and y to get the tree point where both sides met. After plotting, we recurse 2 times for every side, each recursive call having the new x (x+r) or(x-r) values, and new y value (y-5). I used -5 because that was half of the x input I chose (10).

Proposed solution design and implementation, problems 4 – This next problem was the one that I had most trouble with because I kept making an error that took a while to solve. I first began with the code given to us (circle.py) and modified it. The first thing we did was plot the main circle in the center, that remained unchanged, and so did the first recursive call. Next we created 4 more recursive calls which were similar to the center recursive except, modified to the 1/3, either to the x-axis, added or subtracted, or the y-axis, added or subtracted. To explain this a little further, I created 4 new variables; a, b, c, d in which I copied center into. After creating a, b, c, d, which were equal to center, I modified them by updating their x or y axis. For a I updated the x-axis (a[0]) by adding (radius\*.66), which moves the circle to the right. For b I updated the x-axis (b[0]) also, by subtracting(radius\*.66), which moves the circle to the left. For c I updated the y-axis (c[1]), by adding(radius\*2/3), which moves the circle upwards. For d I updated the y-axis also(d[1]), by subtracting(radius\*2/3), which moves the circle downwards. After these changes, we recurse these new values 4 times, meaning 1 time each, while updating radius by the value of w, the rate of change. Each recursive call is different like the values a, b, c, and d, and recurse an n amount of times. An example of the recursive call is: draw\_circles(ax, n-1, a, radius\*w, w). We kept a as it was modified, and we updated the radius to be shorter, and n was subtracted by 1 to stop at the base case.

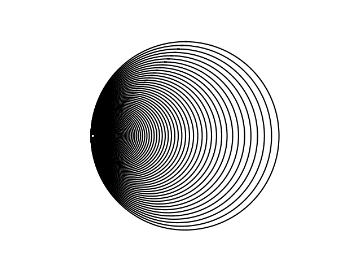
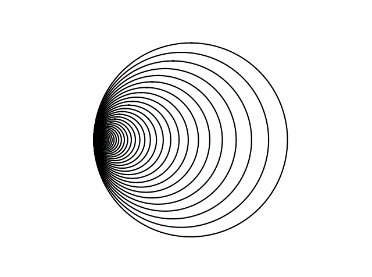
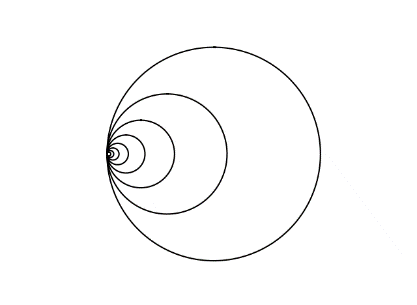
Experimental Problem 1 – With problem 1, I experimented very few times, and even 1 of them caused me to restart my computer because it froze. The reason my computer froze was because I tried to run the recursion 15 times. The inputs I chose were 2, 3, 4 for recursion times (n) because those were the ones we were looking for in the lab, as seen in the lab paper. The r (rate of change) that I chose was 4, because I felt most comfortable with because we were not recursing a lot of times. These were the solutions I got to the problems, but the first experiment I did caused my computer to freeze because I put 15 as the number of recursion times, instead as r.

1a) input 2 as n, and 4 as r 1b) input 3 as n, and 4 as r 1c) input 4 as n, and 4 as r



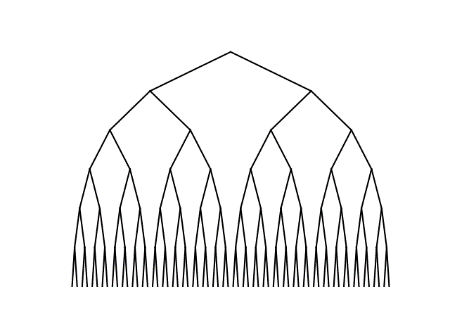
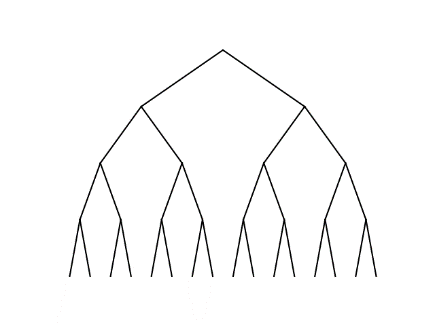
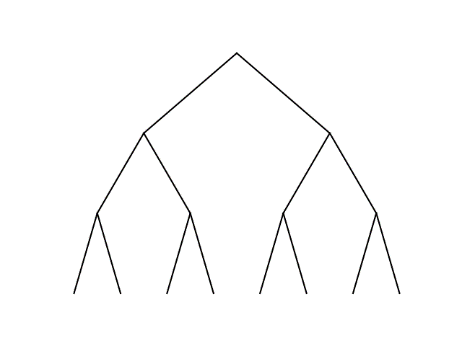
Experimental Problem 2 – With problem 2, I experimented a couple of times, because It was hard to get shapes b and c. Shape a was not very hard because I counted the number of circles and guessed w on the second try. I first began with guessing w as .5 because the circle in the lab paper looked like it was half the length of the origin circle, but that result gave me a very tiny circle about the quarter of the size. Next I tried .75 as the w and got the result I was searching for. The next circles I had to experiment a little because I did not know if I was right or very close to the answer. I looked at how the circle got smaller, and increased w as the circle was only decreasing very little. First I attempted .8 and that was decreasing too much, after that .9 and was getting close. After a couple more guesses, I ended at .95 and 100 circles, which gave me a close answer to the one needed. The next shape ( c ), I saw that the circles were getting smaller at a very shorter rate, so I increased w to .98 and got the answer I wanted, n remained unchanged.

2a) .75 as r, and n as 8 2b).95 as r, and n as 100 2c).98 as r, and n as 100



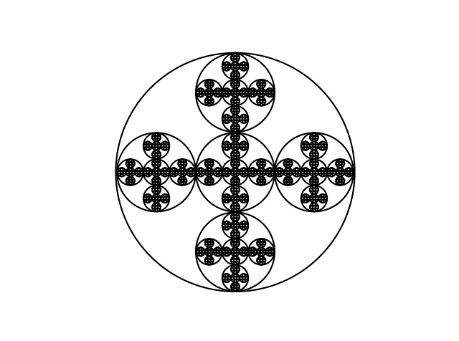
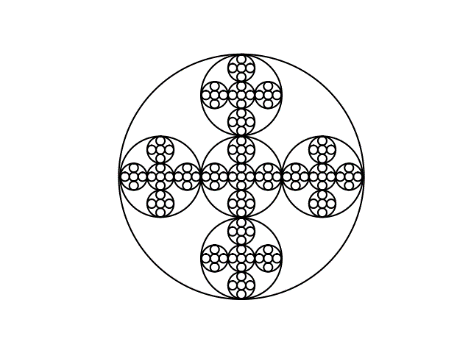
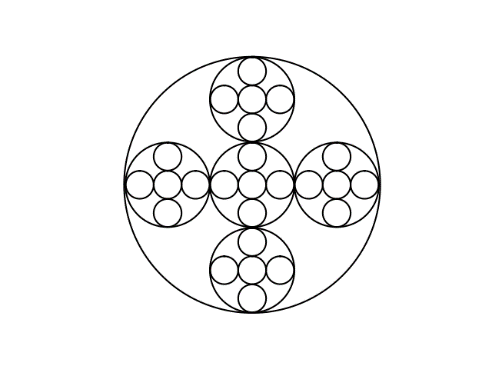
Experimental Problems 3 – With problems 3 I found it easy to find the solutions because they were all in counting range. I counted the amount of children and used that as the n input to find the solutions that were given. For a, I counted 8, which meant that the answer was 3 because 2^3 was 8 and that gave me the answer I was looking for. For b, I counted 16 and that mean the answer was 4 because 2^4 was 16 and I was right! For c, I stopped counting because it was longer, so I tried 5, which was wrong, and got 6 as the answer.

3a) 3 as n 3b) 4 as n 3c) 6 as n



Experimental Problems 4 – With problem 4, it was simple and easy to get the solutions. I counted 3 circles for 4a and the input for n was indeed 3. For b I just increased it by 1, and the input was 4. Finally for c I increased it by 2 to be safe, and was wrong, the input was 5.

4a) 3 as n 4b)4 as n 4c) 5 as n



Conclusion – From this lab I learned a lot more things about python. One of the valuable things I learned was list.copy(input) which was really cool because it allowed me to copy a whole list instead of the index. I also learned lots of new things about python and got really familiar with it. I enjoyed this lab a lot and was a fun experience to experiment.

Appendix

Problems 1

|  |
| --- |
| #Cesar Lopez |
|  | #Lab 1 problems 1a, 1b, 1c |
|  | #CS2302 10:30 - 11:50 |
|  | import matplotlib.pyplot as plt |
|  | #Method we use to recurse. Takes input ax, x-axis, y-axis, number of recursion, and rate of change |
|  | def draw\_squares(ax, x, y, n, r): |
|  | if n>0: |
|  | #xl affects the left side of the square, but only the x - axis |
|  | xl = x-r |
|  | #lr affects the right side of the square, but only the x - axis |
|  | xr = x+r |
|  | #yd affects the bottom side of the square, but only the y - axis |
|  | yd = y-r |
|  | #yu affects the up side of the square, but only the y - axis |
|  | yu = y+r |
|  | #the plotting process starts with the bottom left the square, to top left, to top right, to bottom right, to back to the bottom left |
|  | ax.plot((xl, xl, xr, xr, xl),(yd, yu, yu, yd, yd), color='k') |
|  | #draws bottom left |
|  | draw\_squares(ax, xl, yd, n-1, r/2) |
|  | #draws top left |
|  | draw\_squares(ax, xl, yu, n-1, r/2) |
|  | #draws the top right |
|  | draw\_squares(ax, xr, yu, n-1, r/2) |
|  | #draws the bottom right |
|  | draw\_squares(ax, xr, yd, n-1, r/2) |
|  | plt.close("all") |
|  | fig, ax = plt.subplots() |
|  | #here we have our start point |
|  | x = 0 |
|  | y = 0 |
|  | #1a = (ax, x, y, 2, 4) |
|  | #1b = (ax, x, y, 3, 4) |
|  | #1c = (ax, x, y, 4, 4) |
|  | draw\_squares(ax, x, y, 4, 4) |
|  | ax.set\_aspect(1.0) |
|  | ax.axis('off') |
|  | plt.show() |
|  | fig.savefig('squares.png') |

Problems 2

|  |
| --- |
| #Cesar Lopez |
|  | #Lab 1 Problems 2a, 2b, 2c |
|  | #CS2302 10:30-11:50 |
|  | import matplotlib.pyplot as plt |
|  | import numpy as np |
|  | import math |
|  | #create circle method, copied from the circle file on cs materials |
|  | 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 |
|  | #draw circle method, similar to the one given in class |
|  | def draw\_circles(ax, center, radius, w, n): |
|  | if n>0: |
|  | #plots the new circles based on the recursion |
|  | x,y = circle(center,radius) |
|  | ax.plot(x,y,color='k') |
|  | #Center is changed, but only 0, because that is how we keep our circles in the same position. |
|  | center[0] = center[0] \* (w \* w) |
|  | #here we have our recursive case, which updates our radius just like our center[0], to keep circles in the same position. |
|  | draw\_circles(ax, center, radius\*w\*w, w, n-1) |
|  | plt.close("all") |
|  | fig, ax = plt.subplots() |
|  | #2a = (ax, [100,0], 100, .75, 8) |
|  | #2b = (ax, [100,0], 100, .95, 100) |
|  | #2c = (ax, [100,0], 100, .98, 100) |
|  | draw\_circles(ax, [100,0], 100, .98, 100) |
|  | ax.set\_aspect(1.0) |
|  | ax.axis('off') |
|  | plt.show() |
|  | fig.savefig('circles.png') |

Problems 3

|  |
| --- |
| #Cesar Lopez |
|  | #Lab 1 Problem 3 |
|  | import matplotlib.pyplot as plt |
|  | #Here we have our drawing method. |
|  | def binaryTree(ax, x, y, r, n): |
|  | if n > 0: |
|  | #with this line of code, we plot our left side of the tree |
|  | ax.plot((x, x-r),(y,y-5),color='k') |
|  | #with this line of code, we plot our right side of the tree |
|  | ax.plot((x, x+r),(y,y-5),color='k') |
|  | #this is our first recursive case, which recurses to the right side of the binary tree |
|  | binaryTree(ax, x+r, y-5, r/2, n-1) |
|  | #this is our second recursive case, which recurses to the left side of the binary tree |
|  | binaryTree(ax, x-r, y-5, r/2, n-1) |
|  | plt.close("all") |
|  | #x and y are our initial point on the binary tree. |
|  | x = 10 |
|  | y = x\*3 |
|  | #r is our rate of change in going down the tree |
|  | r = x/2 |
|  | fig, ax = plt.subplots() |
|  | #3a = (ax, x, y, r, 3) |
|  | #3b = (ax, x, y, r, 4) |
|  | #3c = (ax, x, y, r, 6) |
|  | binaryTree(ax, x, y, r, 6) |
|  | plt.axis('off') |
|  | plt.show() |
|  | fig.savefig('binary.png') |

Problems 4

|  |
| --- |
| #Cesar Lopez |
|  | #Lab 1 Problem 4 |
|  | #CS2302 10:30 - 11:50 |
|  | import matplotlib.pyplot as plt |
|  | import numpy as np |
|  | import math |
|  | #circle method used in class to create circles. |
|  | 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) |
|  | #Used to plot the circles |
|  | ax.plot(x,y,color='k') |
|  | #This method calls the center circle that is only affected by w. |
|  | draw\_circles(ax,n-1, center,radius\*w,w) |
|  | #center is copied into variable a, b, c, and d for the right, left, up, and down circles. |
|  | a = list.copy(center) |
|  | b = list.copy(center) |
|  | c = list.copy(center) |
|  | d = list.copy(center) |
|  | #this gets us the right circle |
|  | a[0] = a[0]+(radius\*.66) |
|  | #this gets us the left circle |
|  | b[0] = b[0]-(radius\*.66) |
|  | #Uses upper circle information to plot |
|  | draw\_circles(ax, n-1, a, radius\*w, w) |
|  | #uses lower circle information to plot |
|  | draw\_circles(ax, n-1, b,radius\*w,w) |
|  | #this gets us the upper circle |
|  | c[1] = c[1] + (radius\*2/3) |
|  | #this gets us the bottom circle |
|  | d[1] = d[1] - (radius\*2/3) |
|  | #uses right circle information to plot |
|  | draw\_circles(ax, n-1, c, radius\*w, w) |
|  | #uses left circle information to plot |
|  | draw\_circles(ax, n-1, d, radius\*w, w) |
|  | plt.close("all") |
|  | fig, ax = plt.subplots() |
|  | #4a = (ax, 3, [100, 0], 100, .33) |
|  | #4b = (ax, 4, [100, 0], 100, .33) |
|  | #4c = (ax, 5, [100, 0], 100, .33) |
|  | draw\_circles(ax, 5, [100, 0], 100, .33) |
|  | ax.set\_aspect(1.0) |
|  | ax.axis('off') |
|  | plt.show() |
|  | fig.savefig('circles.png') |

I certify that this project is entirely my own work. I wrote, debugged, and tested the code being presented, performed the experiments, and wrote the report. I also certify that I did not share my code or report or provided inappropriate assistance to any student in the class. C.L