CS2302 - Data Structures

Spring 2019

Lab No.1 Report

Introduction

For this lab we are making use of recursion and libraries to create interesting figures, such as squares, trees and circles. We are given 2 examples of code to be analyzed and used for other similar shapes. The objective is to make correct use of recursion when trying to create figures inside of previously created figures that are again created inside previous ones and so on.

Design & Implementation

To solve the problem at hand we first need to understand how the matplotlib works. For example, to create a square an array of vectors is given, then these vectors become connected and this is what creates the shape. After understanding this can then move onto how to use recursion for drawing future shapes.

Problem #1

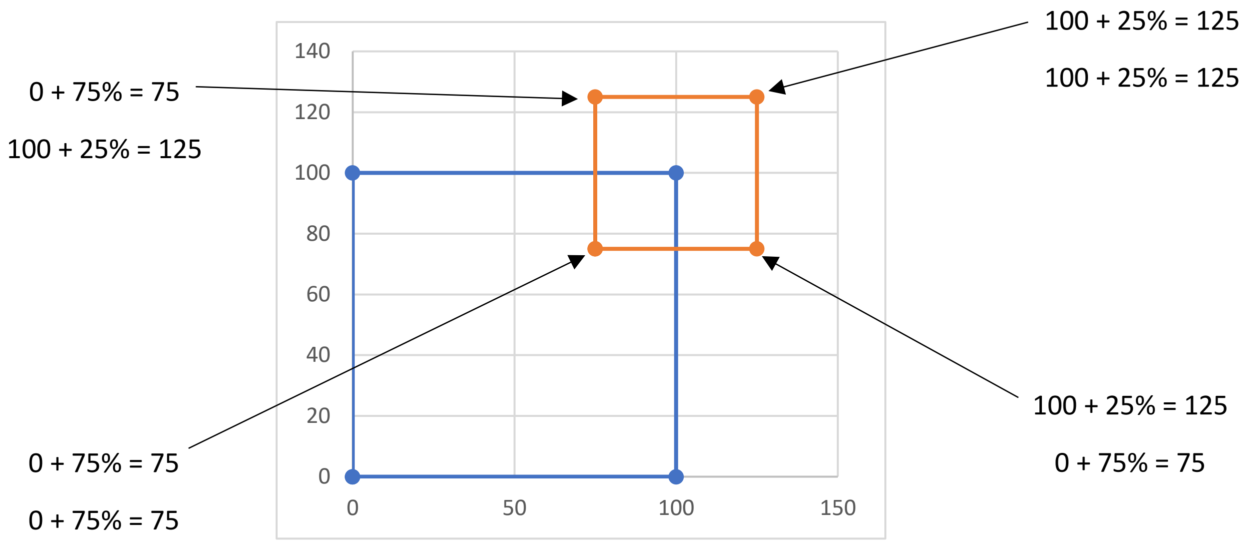
We need to build squares that are 2 times smaller than the original one and which centers reside on the edges of the original square. Also, the vertices of the new squares move around the graph in the x and y axis for 25% or 75% of their original position, be it negative or positive change.

Following the previous rules, we then need to create a recursive method that takes for parameters the plots of the figure, the depth level or copies, the original 2D array of vectors used and the length of the figure. The original length of the figure will be used to obtain the change of 25% and 75% in x and y positions. The depth level value will be user input.

$$Method = \ def\ draw\_squares(ax, d, v, w)$$

The recursive method will then have a base case of when the depth level reaches 0 to stop. It will copy the given array of vectors 4 times, one for each new square to be created and do calculations based on the original length to modify the 2d array of vectors.

The upper right square vector change can be seen in the following figure:



Here is the original vector data and the change for the upper right square of the figure:

X	Y
0	0
0	100
100	100
100	0
0	0

$\longrightarrow$

X	Y
75	75
75	125
125	125
125	75
75	75

Following this logic then we just need to modify the 4 copies of the original array to become the smaller squares for each edge, each vector always decreasing or increasing by 25% or 75%.

The method will then plot the previous given square or original one and then recurse 4 times to draw each new square inside the newly created one, but this time taking the new square vector as the original one, reducing the depth by one and halving the original length since we are now using a smaller square.

$$draw\_squares(ax, d-1, bl, w/2)$$

Depth – 1, so that we reach base case

Smaller 2D array to be drawn and modified

Original Length / 2, to make new calculations on the smaller square

Problem #2

This one comes simpler, we just want to move the center of the circle in the x-axis and also reduce its radius by the same amount. The method will take for parameters the plots of the figure, the depth level, an array for x and y coordinates of the circle's center and a rate of change. The depth level and rate of change values will be user input.

$$Method = \ def \ draw\_circles2(ax, n, center, radius, w)$$

We want for each circle to not surpass the boundary of the original one, so each new circle will be smaller than the original one. Changing the radius and center value by a fraction of the original one. Since this method takes into account dozens of circles we need its depth level to be separated from the other methods or the console will crash.

Same as with the squares before we will reduce its depth level by 1 until reaching base case and exiting recursion. Center at 0 is the change in the x- axis and it is being changed by "w" a variable with a value that the user will enter. For each next circle we will just keep the newly modified center and radius and continue to pass it on to the next call until reaching base case.

$$draw\_circles2(ax, n-1, [center[0] * w, center[1]], radius * w, w)$$

Problem #3

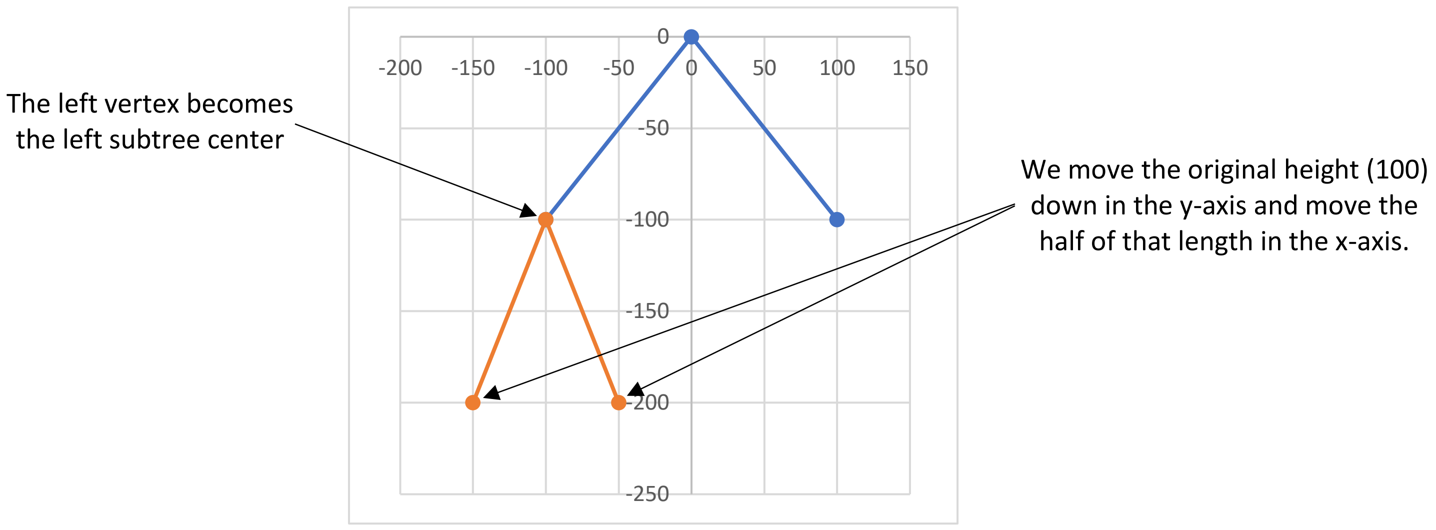
We need to construct trees that go down on the y-axis by the same amount each time, but their length will be reduced by half each time. The new center point for the newly created subtrees will be the left vertices and right vertices respectively.

The method will take for parameters the figure plots, depth level of the figure, a 2D array of the vertices coordinates and an array for the original length of the tree and the original length halved. The depth level for this method is the same user input for the square method.

$$Method = \ def\ draw\_trees(ax, d, v, w)$$

The method will create 2 copies of the given 2D array, one for the left subtree vertices and one for the right subtree vertices. We know that the y-values will always continue reducing by the original length and therefore we have the original length always at hand, the x-values on the other hand will always be the half of the tree length. We will reduce or increase the x values by this half.

The left subtree vector change can be seen in the following figure:



We use the same logic for the right subtree. Changing the values in the newly 2D arrays for the right and left subtrees and then drawing the original given subtree. We call the recursive method 2 times passing the right subtree 2D array into one call and the left one into the other call. Reducing the depth by one and halving the length to obtain the new one.

$$draw\_trees(ax, d-1, rs, [w[0], w[1]/2])$$

Pass on the new subtree

The x-axis decrease will always halve.

The y-axis decrease will always be the same

<u>Problem #4</u>

Here we have to create a circle that has inside 5 smaller circles, and these will also have smaller circles and so on. We can see that 3 circles will always be on a row be it from left to right or up to down, this means their x and y axis of their centers will be on the same line as the center circle. The new circles radius will be a third of the original one, since we are trying to fit 3 of them. The vertices change in x and y will be a third of the diameter also.

Then a recursive method is made that takes as parameters the plot lines of the figure, the depth, an array containing the center coordinates and the radius.

$$Method = def\ draw\_circles(ax, n, center, radius)$$

Inside this method first we will calculate the changes needed to create the new circles. We store in an array the change of radius and also the amount to be added to the x and y axis, this is the radius divided by 3 and diameter divided by 3 respectively.

$$r\_chg\ =\ [radius/3, (2 * radius/3)]$$

We then draw the original circle using the given radius and center, then draw the smaller center one, the right one, the left one and so on.

The center circle will not move so we just reduce its radius to a third.

$$x, y \ = \ circle(center, r\_chg[0])$$

The right circle will also have its radius reduced but its x coordinates must change, we need to add a third of the original circle diameter. Each next new circle will always continue the same logic.

$$x, y \ = \ circle([\underset{\substack{\uparrow \\ \text{Add a third of the diameter} \\ \text{to the x-axis coordinate}}}{center[0] + r\_chg[1]}, center[1]], \underset{\substack{\uparrow \\ \text{Reduce the radius} \\ \text{to a third}}}{r\_chg[0]})$$

We then make a recursive call for the newly created circle, taking for parameters the new modified center and radius. We repeat this same process for each other circle until we reach our base case.

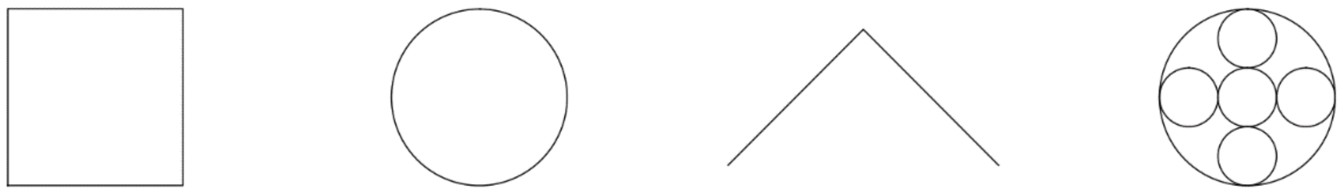
$$draw\_circles(ax, n-1, [center[0] + r\_chg[1], center[1]], r\_chg[0])$$

Design

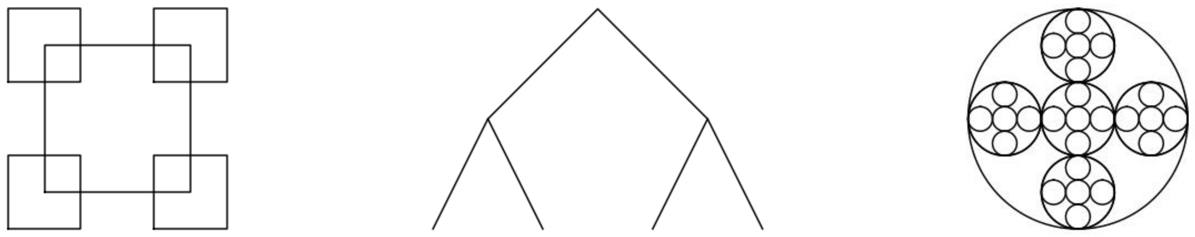
Finally, the code also has a sort of user interface that will ask the user for the depth level for figures 1, 2 & 3 since they are similar enough. The depth level for figure 2 will be asked on a separate manner since we need much more levels to appreciate the recursion, the ratio of radius change for this figure will also be asked.

Experimental Results

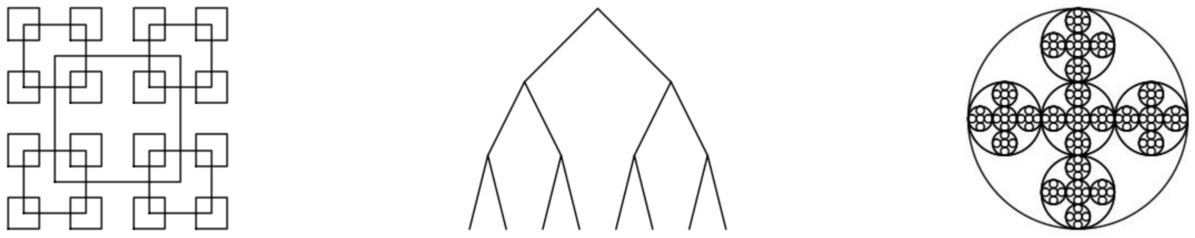
When choosing depth level 1 for all figures the following results are shown in their respective problem order:



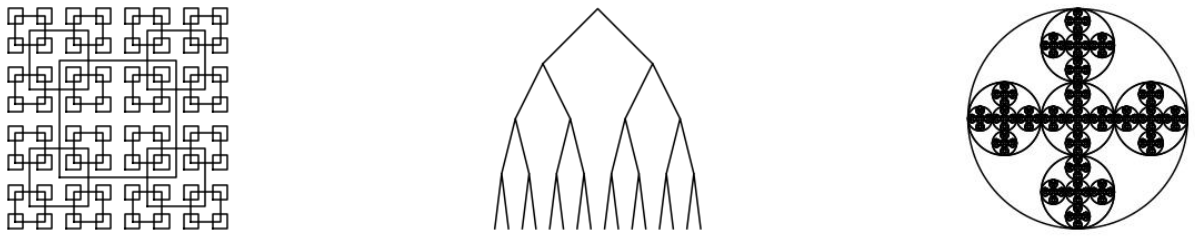
Depth level 2 for figures 1, 3 & 4:



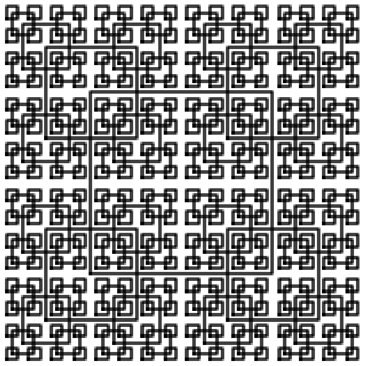
Depth level 3 for figures 1, 3 & 4:

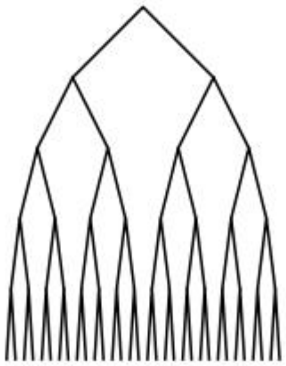


Depth level 4 for figures 1, 3 & 4:

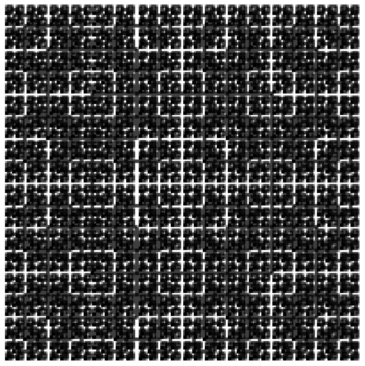


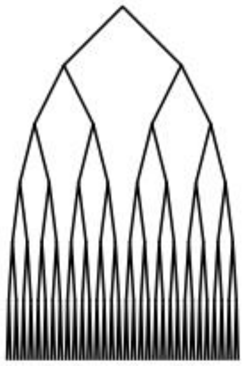
Depth level 5 for figures 1 & 3, figure 4 cannot longer be appreciated:





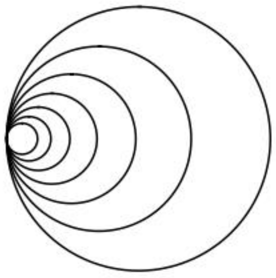
Depth level 6 for figures 1 & 3:

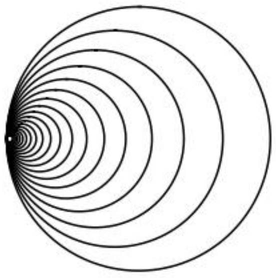




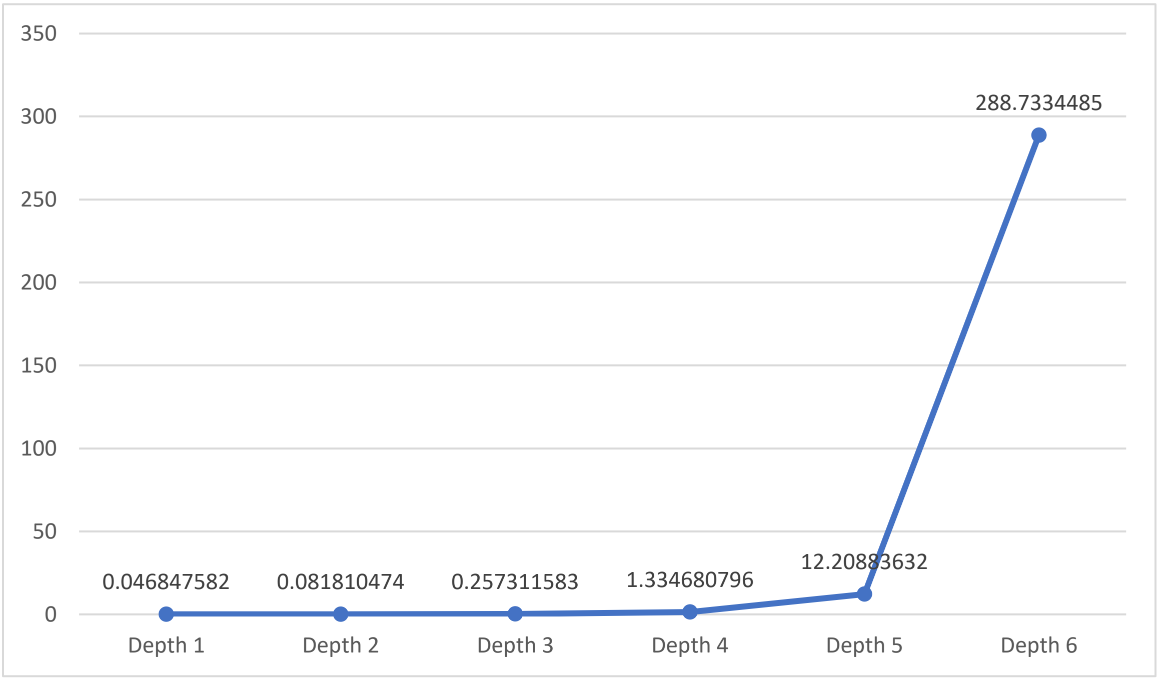
Depth level 7 & ratio = 0.7 for figure 2:

Depth level 20 & ratio = 0.82 for figure 2:





Here are seen the different runtimes of the program across these different depths in seconds:



Conclusion

To effectively use and understand recursion one must practice it. To create the different assigned shapes we needed to make use of recursive methods. I now better understand that when a recursive call ends it returns the control to the previous one and so on, I made use of this strategy to draw the shapes while also changing the given arrays of coordinates. Also the runtimes of the recursive methods I made show how exponential is the growth is when the method calls itself more and more times since after each call the work keeps on expanding.

Use of the libraries numpy and matplotlib helped me in better comprehend them. Since I needed to know that a given 2D array of vertices come together using such library. If I hadn't then other method would have to be used when trying to create the figures.

Furthermore, I understand more python language since starting this project and how to solve given problems. The changes in the language are not as big as I expected.