Laila Romero

CS 2302 Data Structures

March 12, 2019

Lab 3

* Introduction

For this lab we were writing methods for binary search trees. We were supposed to write methods to display a graphic binary search tree, for an iterative search, to take an array and make a binary search tree, to take a tree and return a sorted array, and to print the keys at each depth in the tree.

• Proposed Solution Design and Implementation

For the first part of the lab I took part of my lab 1 source code which printed out the tree pattern with matplotlib. I removed any unnecessary lines and annotated the nodes to look exactly like the one in the lab by setting the centers of the annotated circles to be at the top of the trees.

For the iterative search part, I used a while loop which split the array in half continuously. It then compared the elements to the other elements in the array then returns the index if the element is there and if it is not it will return -1.

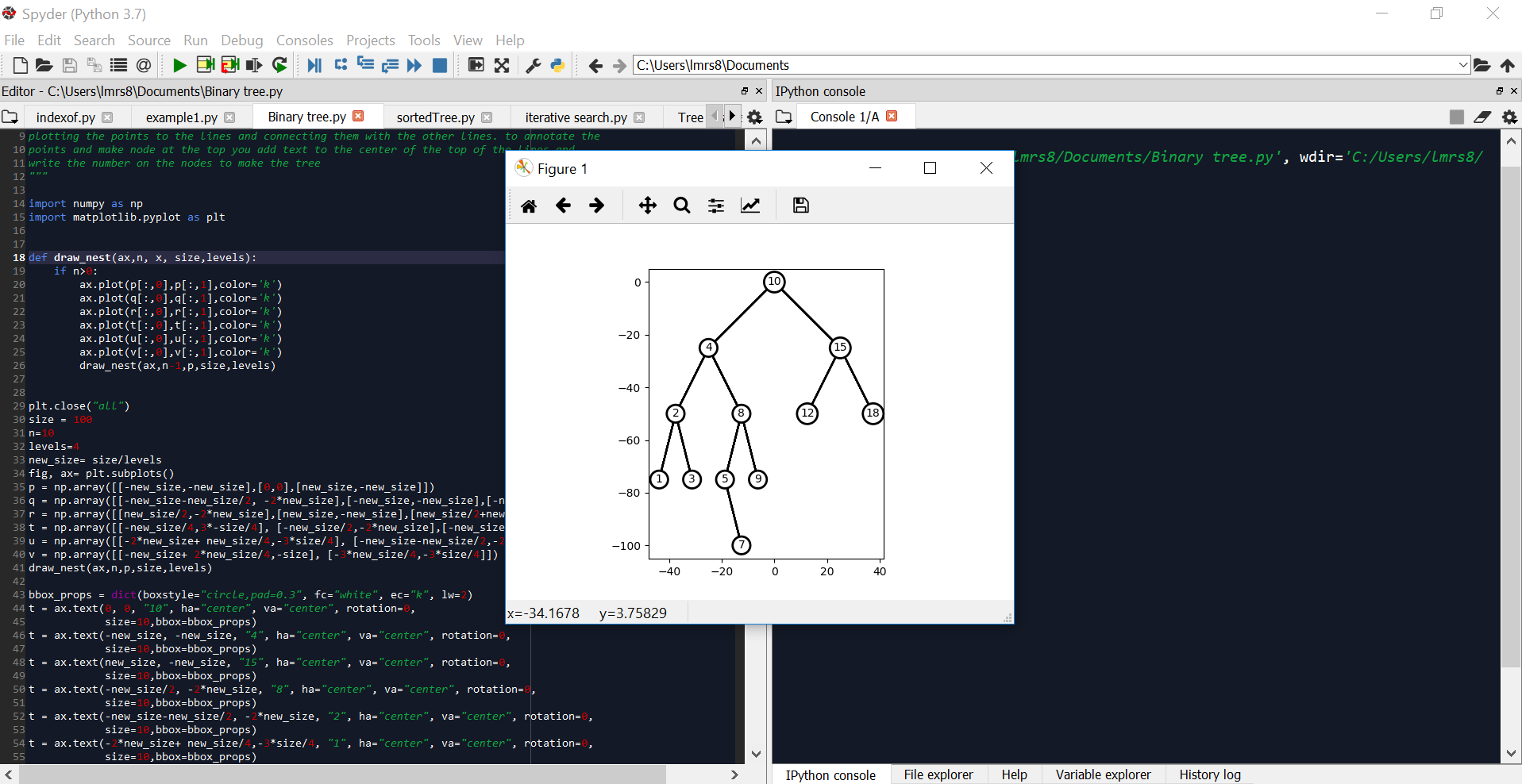
For the third part for building a binary search tree through a sorted array I made two methods called sortedArray and preOrder. The sorted array takes the array and splits it in two then assigns the middle node as the root of the tree. It then separately sorts the left and right half of the tree assigning them to different nodes of the tree. The preorder method then prints out the tree in preorder which would print it from root, left, right.

For extracting an array from the binary tree, I made a method called TreeToArray. It would take the tree and check the nodes from the left side first and put it in to the array first. Then it would check the root and put it in the middle.Lastly it would check the right side of the array and place it in order into the array.

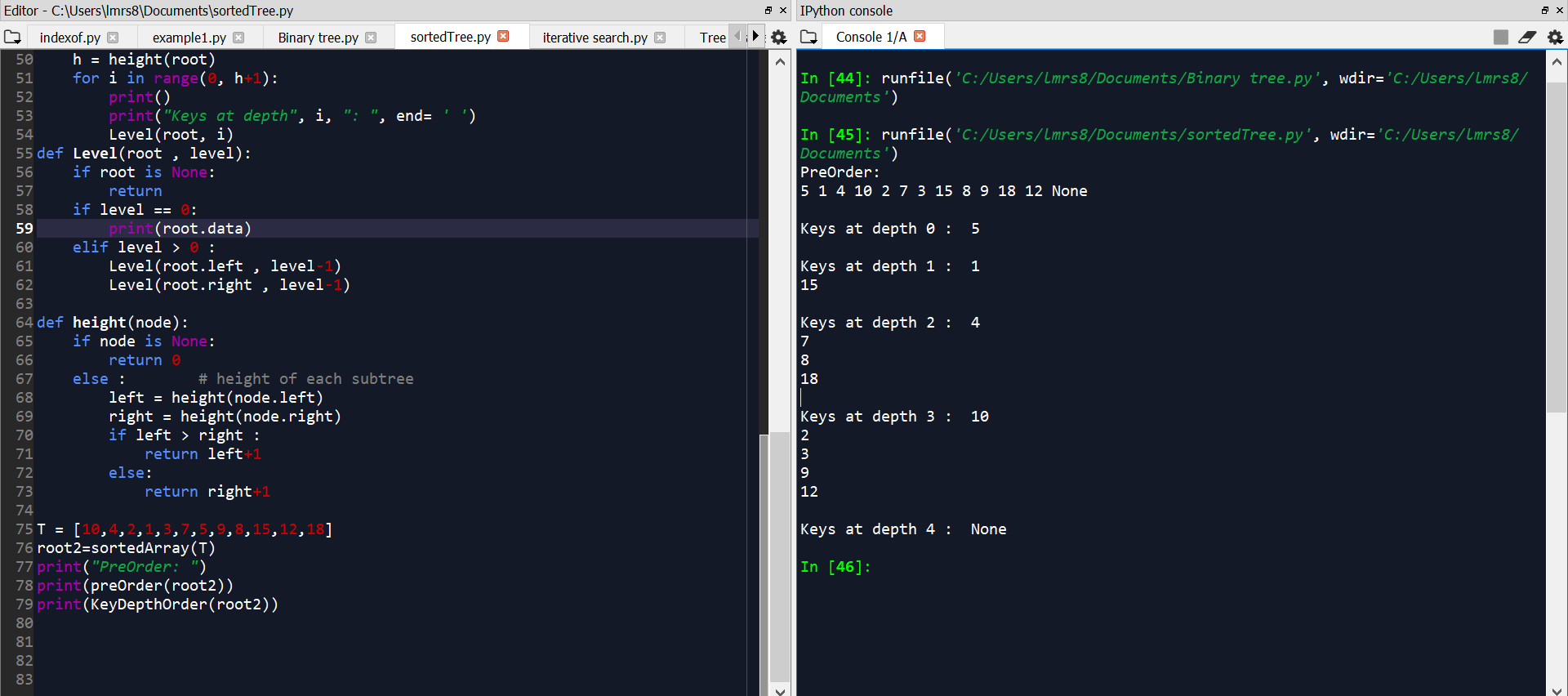
For printing the elements in a binary tree ordered by depth I made three methods called Levels, height, and KeyDepthOrder. Levels sorts out the elements in every depth by identifying the root then sorting the rest of the elements. The height method calculates the depth of every element. And KeyDepthOrder prints out the elements in order of depth beginning at depth 0.

• Experimental results

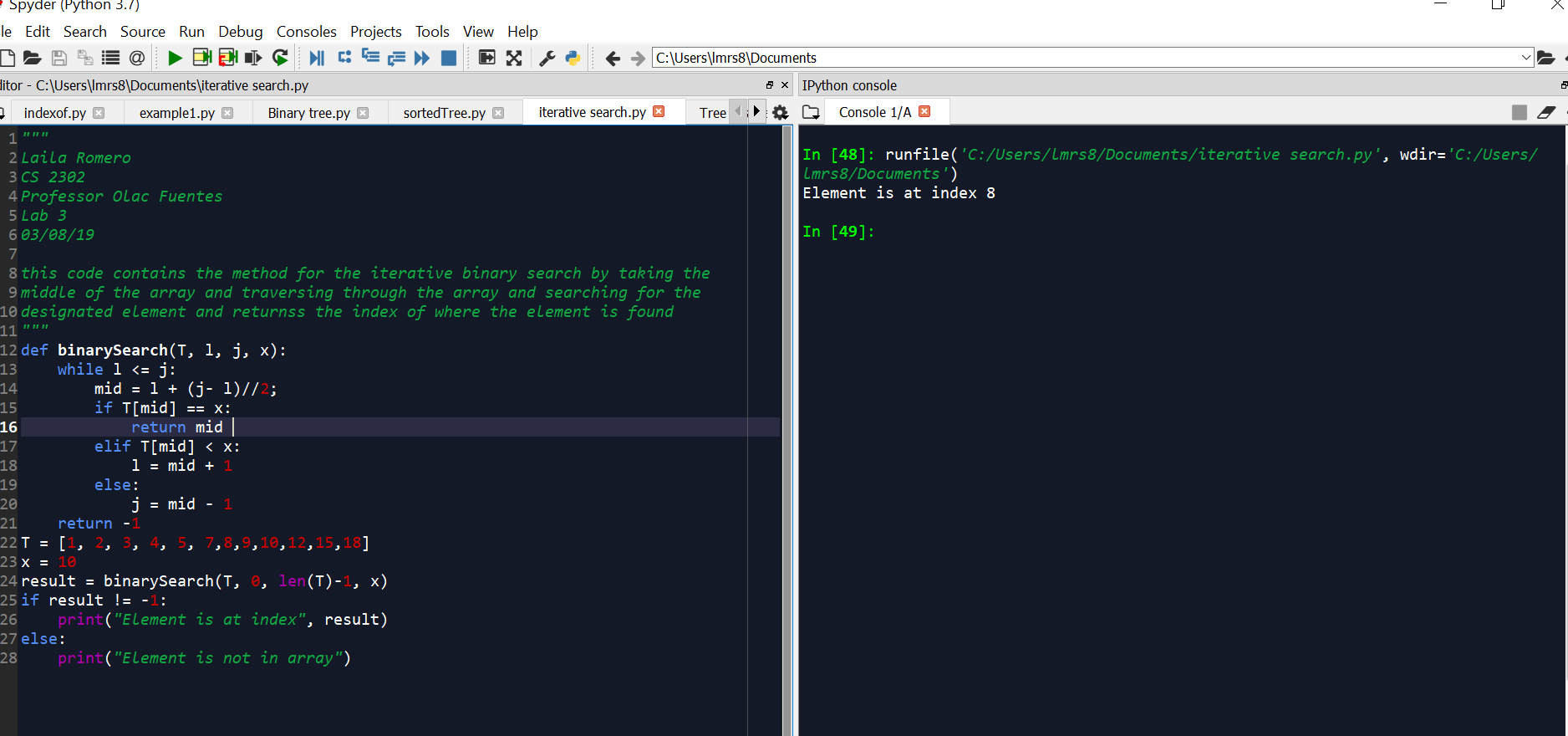
For the graphics part I was able to get the binary search tree to appear exactly as the lab requested as seen in the screenshot below.



For building a balanced binary search tree from a sorted array I had to work through a lot of trial and error to get the tree to print balanced. After multiple attempts I was able to get the results I wanted. When it came to extracting an array from a binary search tree I was also met with difficulties. Although I was able to come up with a method that I best believed worked but was met with errors so I had to comment it out. I understood the concept behind extracting the elements from the binary tree but unfortunately was not able to get past the errors my method made. For the keys at depth I was able to find the depth of the binary tree and take each element and print them with their corresponding depths. My method would also provide an extra depth with elements of none to make sure the tree was completely traversed. Below are the results of the binary search and the keys at depth.



For the iterative search, I was able to return the index of the element without running in to any problems by using the method of splitting the array. The results can be seen below in the screenshot.



• Conclusion

In conclusion, I learned how to maneuver through binary search trees. I learned how to use iterative versions of the binary search method instead of recursion and how to access individual nodes. I also learned how to annotate graphics on matplotlib. I also learned how to convert arrays into binary search trees and vice versa.

• Appendix – Source codes

class Node:

def \_\_init\_\_(self, data):

self.data = data

self.left = None

self.right = None

class BST(object):

# Constructor

def \_\_init\_\_(self, item, left=None, right=None):

self.item = item

self.left = left

self.right = right

#def TreeToArray(node, root):

# arr = []

# if node is None:

# return None

#TreeToArray(root.leftChild())

# arr[index++] = root.item

#TreeToArray(root.rightChild())

def sortedArray(T):

if not T:

return None

mid = (len(T)) // 2

root = Node(T[mid])

root.left = sortedArray(T[:mid])

root.right = sortedArray(T[mid+1:])

return root

def preOrder(node):

if not node:

return

print( node.data, end=' ')

preOrder(node.left)

preOrder(node.right)

def KeyDepthOrder(root):

h = height(root)

for i in range(0, h+1):

print()

print("Keys at depth", i, ": ", end= ' ')

Level(root, i)

def Level(root , level):

if root is None:

return

if level == 0:

print(root.data)

elif level > 0 :

Level(root.left , level-1)

Level(root.right , level-1)

def height(node):

if node is None:

return 0

else : # height of each subtree

left = height(node.left)

right = height(node.right)

if left > right :

return left+1

else:

return right+1

T = [10,4,2,1,3,7,5,9,8,15,12,18]

root2=sortedArray(T)

print("PreOrder: ")

print(preOrder(root2))

print(KeyDepthOrder(root2))

def binarySearch(T, l, j, x):

while l <= j:

mid = l + (j- l)//2;

if T[mid] == x:

return mid

elif T[mid] < x:

l = mid + 1

else:

j = mid - 1

return -1

T = [1, 2, 3, 4, 5, 7,8,9,10,12,15,18]

x = 10

result = binarySearch(T, 0, len(T)-1, x)

if result != -1:

print("Element is at index", result)

else:

print("Element is not in array")

import numpy as np

import matplotlib.pyplot as plt

def draw\_nest(ax,n, x, size,levels):

if n>0:

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

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

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

ax.plot(t[:,0],t[:,1],color='k')

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

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

draw\_nest(ax,n-1,p,size,levels)

plt.close("all")

size = 100

n=10

levels=4

new\_size= size/levels

fig, ax= plt.subplots()

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

q = np.array([[-new\_size-new\_size/2, -2\*new\_size],[-new\_size,-new\_size],[-new\_size/2,-2\*new\_size]])

r = np.array([[new\_size/2,-2\*new\_size],[new\_size,-new\_size],[new\_size/2+new\_size,-2\*new\_size]])

t = np.array([[-new\_size/4,3\*-size/4], [-new\_size/2,-2\*new\_size],[-new\_size+new\_size/4,-3\*size/4]]) #5

u = np.array([[-2\*new\_size+ new\_size/4,-3\*size/4], [-new\_size-new\_size/2,-2\*new\_size], [-new\_size-new\_size/4,-3\*size/4]])

v = np.array([[-new\_size+ 2\*new\_size/4,-size], [-3\*new\_size/4,-3\*size/4]])

draw\_nest(ax,n,p,size,levels)

bbox\_props = dict(boxstyle="circle,pad=0.3", fc="white", ec="k", lw=2)

t = ax.text(0, 0, "10", ha="center", va="center", rotation=0,

size=10,bbox=bbox\_props)

t = ax.text(-new\_size, -new\_size, "4", ha="center", va="center", rotation=0,

size=10,bbox=bbox\_props)

t = ax.text(new\_size, -new\_size, "15", ha="center", va="center", rotation=0,

size=10,bbox=bbox\_props)

t = ax.text(-new\_size/2, -2\*new\_size, "8", ha="center", va="center", rotation=0,

size=10,bbox=bbox\_props)

t = ax.text(-new\_size-new\_size/2, -2\*new\_size, "2", ha="center", va="center", rotation=0,

size=10,bbox=bbox\_props)

t = ax.text(-2\*new\_size+ new\_size/4,-3\*size/4, "1", ha="center", va="center", rotation=0,

size=10,bbox=bbox\_props)

t = ax.text(-new\_size-new\_size/4,-3\*size/4, "3", ha="center", va="center", rotation=0,

size=10,bbox=bbox\_props)

t = ax.text(-new\_size/4,-3\*size/4, "9", ha="center", va="center", rotation=0,

size=10,bbox=bbox\_props)

t = ax.text(-new\_size+new\_size/4,-3\*size/4, "5", ha="center", va="center", rotation=0,

size=10,bbox=bbox\_props)

t = ax.text(new\_size/2,-2\*new\_size, "12", ha="center", va="center", rotation=0,

size=10,bbox=bbox\_props)

t = ax.text(new\_size/2+new\_size,-2\*new\_size, "18", ha="center", va="center", rotation=0,

size=10,bbox=bbox\_props)

t = ax.text(-new\_size+ 2\*new\_size/4,-size, "7", ha="center", va="center", rotation=0,

size=10,bbox=bbox\_props)

bb = t.get\_bbox\_patch()

bb.set\_boxstyle("circle", pad=0.3)

ax.set\_aspect(1.0)

ax.axis('on')

plt.show()

#fig.savefig('squares.png')