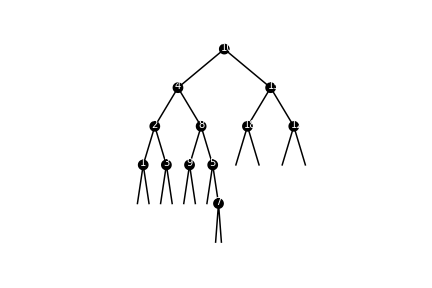
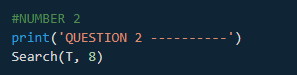
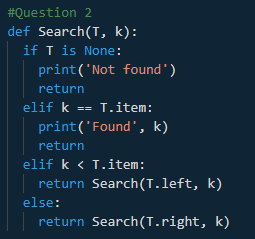
Lab 3

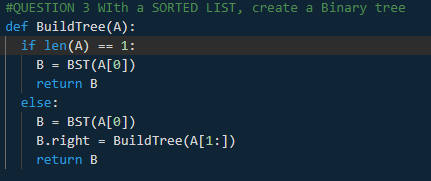
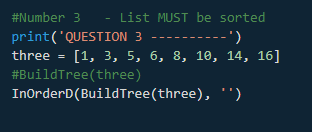
For this lab we are dealing with Binary Search Trees. We need to create a graph of the binary search tree, and an iterative search function. We need to create a method that makes a BST from an array, make a sorted list from a BST, and print the value at a certain depth. For my lab, all that is unfinished is question 1.

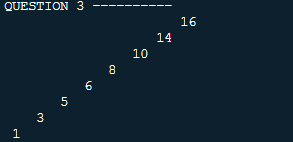
First off graphing the binary tree was most troublesome, as it requires you to input the logic of the binary tree into a graph. The trouble I had was getting the graph to work properly in displaying the numbers, getting it to form properly, and getting the order of the BST correct. As of now, the errors in my code are that the tree more resembles the tree from Lab 1-3, so that it has extra branches that are not needed. My traversal of the BST is off, as whenever the tree traverses right, the next Left/Right values get swapped. And lastly the Numbers in the tree are not formatted as the lab intended it. I filled the circles black with white text, mainly because I couldn’t get a full white circle with the black text within it, as the branches kept overlapping the circles.



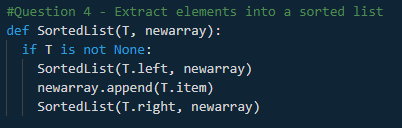
 For the second question, if I’m correct, iterative means that it processes everything node by node, as it traverses to its desired value. So, I developed a search function (Although I’m not too sure it is how the lab wanted it.) that checks the value of each node to see if it’s the value k. If it is, I return that it is found. Otherwise I travel left of the node if its less that the node value, and right if it is more. Also, if it traverses to a node not in the tree, then the value is said to be not found. Run time is O(n)

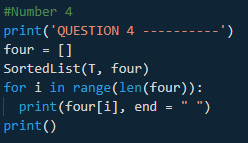
 

 For the third question, we are to build a balanced BST only given a sorted list as an input. Initially, this was confusing, as I wasn’t sure how to properly traverse the array while creating the tree. However, because the list is already sorted from least to greatest, the BST will only have right children. And so, I use recursion to add to new tree B as I split the array down through each recursive statement. Run time is O(n)

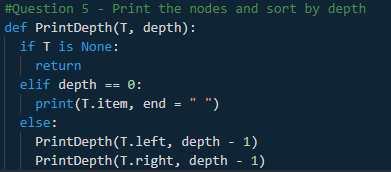


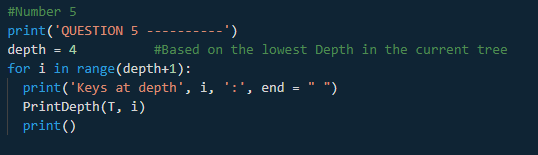
For problem 4, we need to extract the elements into a sorted list. This is similar to how you print in order, except instead of printing the node value, I append it into an array. Running time is O(n)

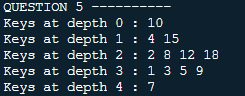


Problem 5, we need to print all the elements sorted by depth. So, it needs to read all the elements in the tree, while determining what depth it is located at. What I created is a method that prints all the elements in a depth, and I read that method through a for statement, so that it comes out reading the BST sorted by depth. The problem at the moment with how I created it, is that the user has to manually edit the depth to fit the maximum depth of the tree, otherwise there will either be missing values, or extra outputs (Showing as: Keys at depth : ) The running time for this is O(n^2)







In conclusion, what I mostly learned/realized is to read directions carefully, as it would have saved me a lot of time when working on question three. As for BST I know the logic of how they operate and how to traverse them. What I need to practice is how to deal with graphs and the logic of fitting a BST within it. I need to be able to get my graph working correctly, and the tree working within it correctly too.

APPENDIX:

#Patrick Brannan - Last Edited 4/5

#Only Question 1 is Unfinished.

import numpy as np

import matplotlib.pyplot as plt

import math

#BST CODE

class BST(object):

# Constructor

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

self.item = item

self.left = left

self.right = right

def Insert(T,newItem):

if T == None:

T = BST(newItem)

elif T.item > newItem:

T.left = Insert(T.left,newItem)

else:

T.right = Insert(T.right,newItem)

return T

def Delete(T,del\_item):

if T is not None:

if del\_item < T.item:

T.left = Delete(T.left,del\_item)

elif del\_item > T.item:

T.right = Delete(T.right,del\_item)

else: # del\_item == T.item

if T.left is None and T.right is None: # T is a leaf, just remove it

T = None

elif T.left is None: # T has one child, replace it by existing child

T = T.right

elif T.right is None:

T = T.left

else: # T has two chldren. Replace T by its successor, delete successor

m = Smallest(T.right)

T.item = m.item

T.right = Delete(T.right,m.item)

return T

def InOrder(T):

# Prints items in BST in ascending order

if T is not None:

InOrder(T.left)

print(T.item,end = ' ')

InOrder(T.right)

def InOrderD(T,space):

# Prints items and structure of BST

if T is not None:

InOrderD(T.right,space+' ')

print(space,T.item)

InOrderD(T.left,space+' ')

def Largest(T):

if T.right is None:

return T

else:

return Largest(T.right)

#Question 2

def Search(T, k):

if T is None:

print('Not found')

return

elif k == T.item:

print('Found', k)

return

elif k < T.item:

return Search(T.left, k)

else:

return Search(T.right, k)

#QUESTION 3 WIth a SORTED LIST, create a Binary tree

def BuildTree(A):

if A[0] == A[-1]:

B = BST(A[0])

return B

else:

B = BST(A[0])

B.right = BuildTree(A[1:])

return B

if T == None:

T = BST(newItem)

elif T.item > newItem:

T.left = Insert(T.left,newItem)

else:

T.right = Insert(T.right,newItem)

return T

#Question 4 - Extract elements into a sorted list

def SortedList(T, newarray):

if T is not None:

SortedList(T.left, newarray)

newarray.append(T.item)

SortedList(T.right, newarray)

#Question 5 - Print the nodes and sort by depth

def PrintDepth(T, depth):

if T is None:

return

elif depth == 0:

print(T.item, end = " ")

else:

PrintDepth(T.left, depth - 1)

PrintDepth(T.right, depth - 1)

#GRAPHING CODE ---- ALSO NUMBER 1 - UNFINISHED

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\_circle(ax, center, radius):

x,y = circle(center,radius)

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

ax.fill(x, y, "k")

def insert\_text(x,y, text):

plt.text(x-20 , y-10, text, color='w')

#MAKE TREE SHAPES

def tree(ax,n,a, b, c, x, T):

if n>0:

if T is not None:

p = np.array([a, b, c])

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

#Make on left

x = x\*.5

bl = a

cl = [a[0] + x, a[1]-250]

al = [a[0] - x, a[1]-250]

tree(ax, n-1, al, bl, cl, x, T.left)

draw\_circle(ax, b, 30)

insert\_text(b[0], b[1], T.item)

#Make on right

br = c

ar = [c[0] + x, c[1]-250]

cr = [c[0] - x, c[1]-250]

tree(ax, n-1, ar, br, cr, x, T.right)

#SETUP

plt.close("all")

# Code to test the functions above

T = None

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

for a in A:

T = Insert(T,a)

#NUMBER 1 - UNFINISHED

a, b, c = [700, 750], [1000, 1000], [1300, 750]

x = 300

fig, ax = plt.subplots()

n = 5 #layers of trees

tree(ax, n, a, b, c, x, T)

ax.set\_aspect(1.0)

ax.axis('off')

plt.show()

fig.savefig('BST.png')

#NUMBER 2

print('QUESTION 2 ----------')

Search(T, 8)

#Number 3 - List MUST be sorted

print('QUESTION 3 ----------')

three = [1, 3, 5, 6, 8, 10, 14, 16]

#BuildTree(three)

InOrderD(BuildTree(three), '')

#Number 4

print('QUESTION 4 ----------')

four = []

SortedList(T, four)

for i in range(len(four)):

print(four[i], end = " ")

print()

#Number 5

print('QUESTION 5 ----------')

depth = 4 #Based on the lowest Depth in the current tree

for i in range(depth+1):

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

PrintDepth(T, i)

print()

-------**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. -Patrick Brannan**