Jon Munoz

Lab 2

CS2302

Professor: Olac Fuentes

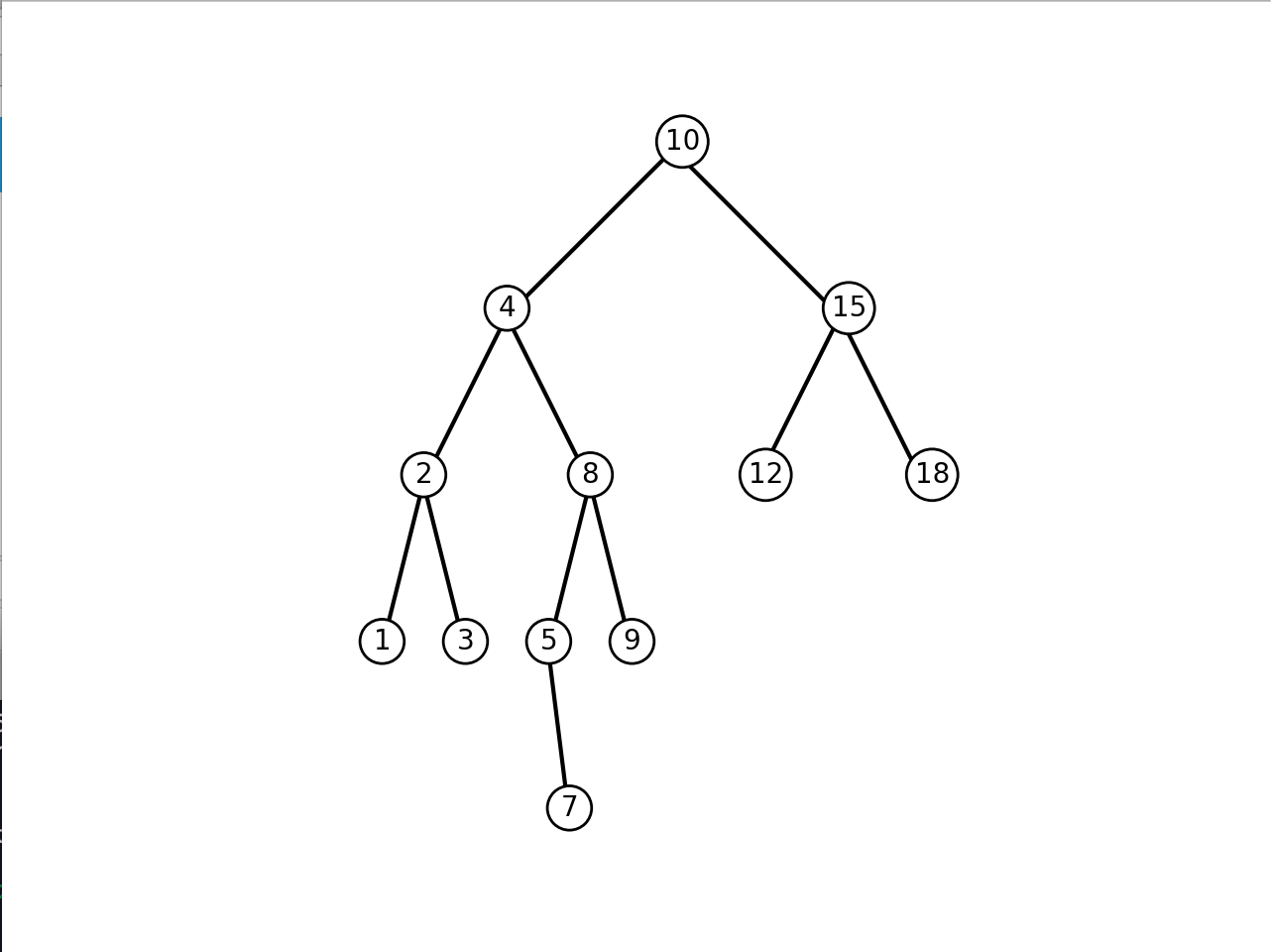
**INTRODUCTION**

For this lab we were working with binary search trees. We had five questions we had to work on those being: drawing the tree, implementing an iterative search method, converting a tree to a list, converting a list to a tree and print the elements in the tree by their depth in the tree.

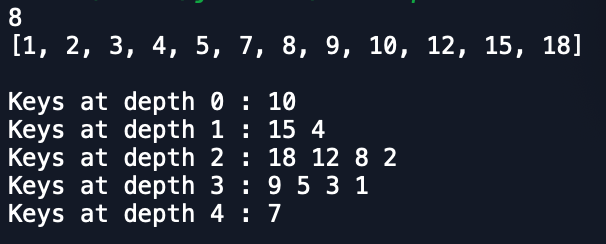
**PROPOSED SOLUTION DESIGN AND IMPLEMENTATION**

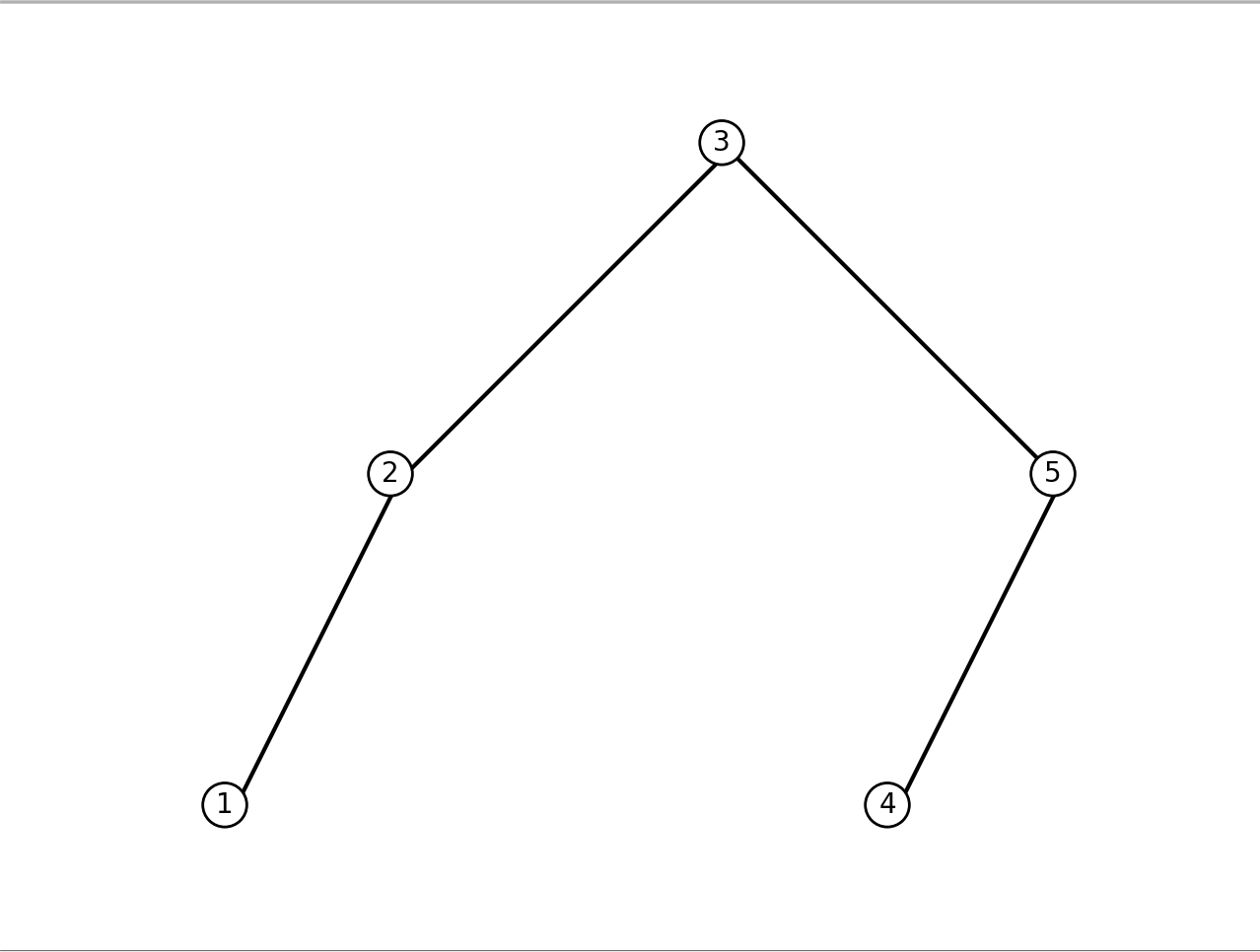
For these questions I had some difficulty due to the fact that I would over complicate some of them, in particular question 5, but I was able to solve them eventually with a plt.text with a bbox call. With the drawing tree method, I was having trouble due to the fact that I could not figure out how to draw the circles with the numbers in them. The iterative search method was not difficult since all I had to do was traverse down the tree while comparing the item in the tree with the key and choosing my path based on the result of the comparison. Making the list out of the tree was also giving me some issues at first. I was originally trying to just make two list, one being the left subtree and the other being the right subtree, but once I tried returning the list at the end I was just getting an empty list. I eventually made three list: the left subtree list, the root list, and right subtree list. At the end I concatenated these three lists to give me the final list and this worked out. With the list to tree I made the root of the tree the middle point of the list. I then set the first half if the list to the left subtree and the rest of the list as the right subtree. This one did not really give me that much trouble. With the printing depths I made two separate methods. One method printed items at a given depth and then my second method I made a for loop to call the given depth method for as long as the tree was not None.

**EXPERIMENTAL RESULTS**

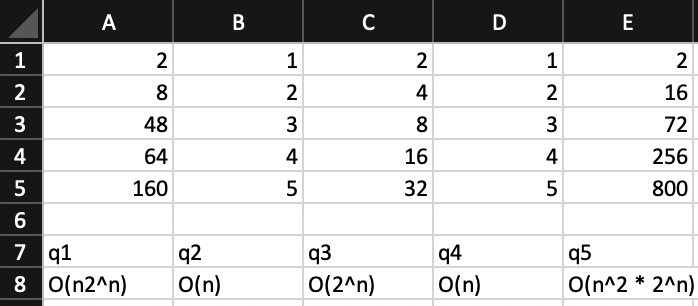
For my print tree method this is the result that I got. The tree does come out very similar to how its supposed to be but some of the circles are a bit off and every time I tried fixing the x coordinates of the circles that were not aligned then the ones that were originally aligned would get messed up. So this was the closest that I could get but it is still very similar.

For the search method I tried using 8 as my key. Since 8 was in the tree once the method finds it then it will print the key. Writing the report I see now that If the key is not found nothing will happen. I should have put a case where the key is not found return -1 or something else. With the tree to list method I created the printed list below and as you can see it worked out just fine with the method returning the desired output. The method printing all the items at each depth is also working. As seen below each depth has all of its items printed together with the indication of their depths which is what I was trying to do with the method.



For the list to tree I used a small input of 5 items to make a tree. When I called the method and drew it I got the desired outcome which is pictured below.

Below I have also included the running times for each of my methods. The y axis is the input sizes starting at 1 and going to 5 and the x axis are the question numbers starting with 1 to 5:



**CONCLUSION**

At the end of this assignment I learned how to work with binary search trees. Once I understood how to traverse the tree and stop when desired I was able to easily complete the assignment. Even though I knew some things about this data structure before the class I was nowhere close to where I am now.

**APPENDIX**

**SOURCE CODE**

﻿#Jon Munoz

#CS2302 Data Structures

#Lab 3

#Instructor:Olac Fuentes

#TA:Anindita Nath

#Last Modified 3/8/19

import numpy as np

import matplotlib.pyplot as plt

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 inOrder(T):

if T == None:

return

else:

inOrder(T.left)

print(T.item,end = ' ')

inOrder(T.right)

def smallest(T):

if T is None:

return None

t = T

while t.left is not None:

t = t.left

return t

def smallestR(T):

if T.left is None:

return T

return smallest(T.left)

def largestR(T): #can do something like t = T and use t in the rest

if T.right is None:

return T

return largestR(T.right)

def find(T, key):

if T is None or T.item == key:

return T

if T.item < key:

find(T.right, key)

return find(T.left,key)

#method counts the total number of nodes in given tree

def numNodes(T):

if T == None:#if T is none then you want to return 0 since there are no nodes at this point

return 0

else:

count = 1

if T.right != None:#if T.right is not None then you want to continue down that tree and continually add 1 until T is None

count += numNodes(T.right)

if T.left != None:#if T.left is not None then you want to continue down that tree and continually add 1 until T is None

count += numNodes(T.left)

return count#return count (the total number of nodes)

#gets the height of given tree

def getHeight(T):

if T == None:#if T is none then then you want to return 0 since there is no more depth at this point

return 0

else:

lHeight = getHeight(T.left)#lHeight gets the height of the left

rHeight = getHeight(T.right)#rHeight gets the height of the right

if lHeight > rHeight:#compare lHeight with rHeight and if lHeight is greater then you want to return lHeight plus 1 since lHeight went down further than rHeight

return lHeight + 1

else:

return rHeight + 1#else rHeight was bigger so you return that

#Search method but done iteritively

def findIterative(T, key):

while T.item != key:#as long as the item at the current node is not the key then you keep traversing

if key < T.item:#if the key is less than the current item then you want to traverse the left subtree

T = T.left

elif key > T.item:#if the key is less than the current item then you want to traverse the right subtree

T = T.right

return T

#method creates a sorted list from a binary tree

def TtoL(T):

if T == None:

return []#return an empty list if T is equal to none since there is nothing in the node

elif T != None:

rootList = [T.item]#rootList is the list comprised of the root of the tree(or node)

smallList = TtoL(T.left)#recursively calls the method with the left subtree and stores what it returns into smallList

largeList = TtoL(T.right)#recursively calls the method with the right subtree and stores what it returns into largeList

return smallList + rootList + largeList#return the concatenation of rootList, smallList, and moreList

#method creates a binary tree out of a sorted list

def LtoT(L):

if(len(L)==0):

return None

elif len(L) > 0:#since the lenght must be greater than 0 to have contents this is the constraint

mid =len(L)//2#mid is assigned to the length of the list divided by 2

T = BST(L[mid])#puts the middle element of the list into the node of the tree

T.left = LtoT(L[:mid])#recursively calls the LtoT method with the second half of the list

T.right = LtoT(L[mid + 1:])#recursively calls the LtoT method with the first half of the list

return T

#This method prints all the items at a given depth

def printAtDepth(T, d):

if T == None:#base case for when T is None

return

if d == 0:#if d is 0 then you are at your desired depth and proceed to print the elements

print(T.item, end = ' ')#prints the items on the given depth

else:

printAtDepth(T.right, d - 1)#traverses down the right tree to the given depth

printAtDepth(T.left, d - 1)#traverses down the right tree to the given depth

#this method uses printAtDepth to print each depths items

def printAtDepth2(T, n):

if T == None:

return

if n != -1:#-1 since the depth can be 0

for i in range(n):

print()#this print statement seperates each of the below print statement

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

printAtDepth(T, i)#recursive call to the printAtDepth method

#lines 131 - 144 are required to draw the tree shape

def draw\_triangle(ax,n,p,w):

if n>0:

i1 = [1,0]

q = p\*w + p[i1]\*(1-w)

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

draw\_triangle(ax,n-1,q,w)

plt.close("all")

orig\_size = 1000

fig, ax = plt.subplots()

ax.set\_aspect(1.0)

ax.axis('off')

plt.show()

fig.savefig('tree.png')

#this method draws the BST

def draw\_tree(T, x, y, height, deltaX):

if T is not None:#continue with the below lines while T is not None since you dont want to make unnessecary lines

plt.text(x - .5, y + height, T.item, bbox = {"boxstyle": "circle", "facecolor": "white", "edgecolor": "black"})#this line drawsa circle at the desired point and draws the T item in said circle

if T.left is not None:#continue to the methods that draw the lines and the recursive call as long as T.left is not None

p = np.array([[x-deltaX,y],[x, y+height]])

draw\_triangle(ax, 1, p,.9)

draw\_tree(T.left , x-deltaX, y-height, height, deltaX/2)

if T.right is not None:#continue to the methods that draw the lines and the recursive call as long as T.right is not None

p = np.array([[x, y+height],[x+deltaX,y]])

draw\_triangle(ax, 1, p,.9)

draw\_tree(T.right , x+deltaX, y-height, height, deltaX/2)

#lines 170 - 173 create the BST

T = None

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

for a in A:

T = Insert(T,a)

#call to draw\_tree which is question 1

#draw\_tree(T, 10, 10, 10, 10)

#method call to findIterative which is question 2

print(findIterative(T, 8).item)

#method call to LtoT which is question 3

L = [1,2,3,4,5]#sample list

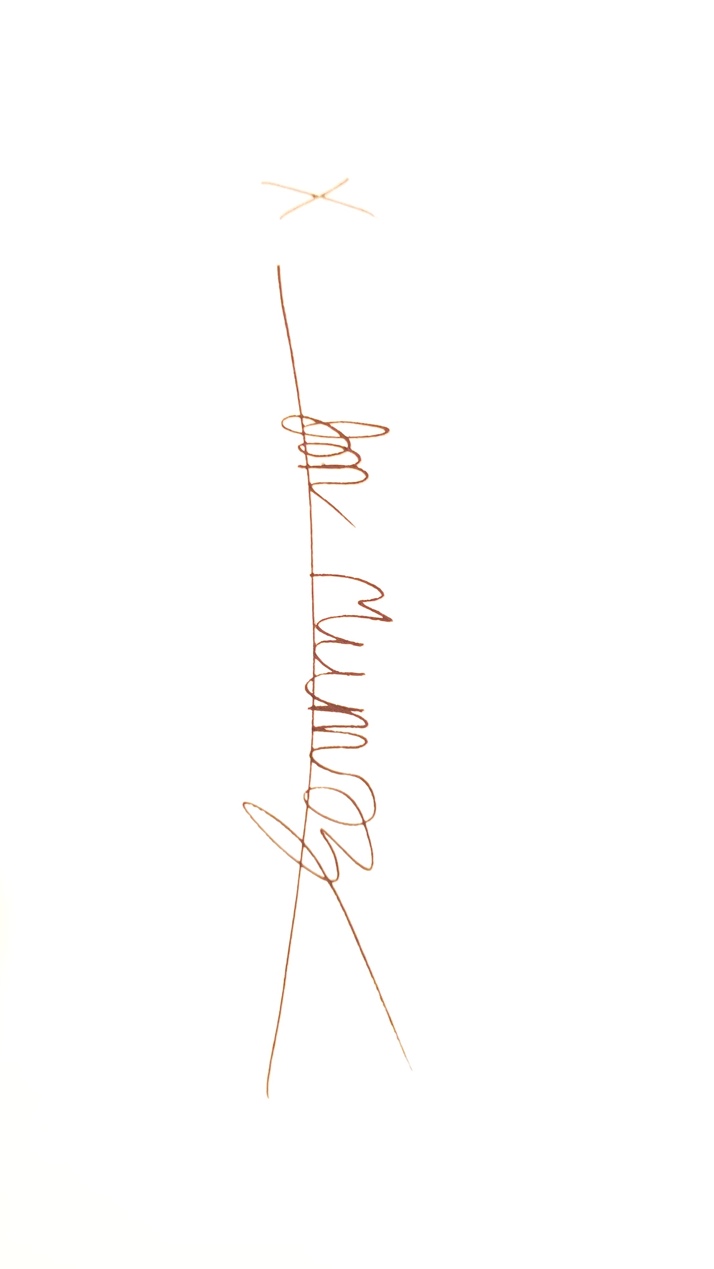
NEWTREE = LtoT(L)

draw\_tree(NEWTREE, 10, 10, 10, 10) #prints the tree as proof method works

#method call to TtoL which is question 4

NEWLIST = TtoL(T)

print(TtoL(T)) #prints the list to prove method works

#method call to printAtDepth2 which is question 5

printAtDepth2(T, 5)

“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 provide inappropriate assistance to any student in the class.”