Samuel Chong

Olac Fuentes

CS 2302

MW 10:30-11:50

Lab 3

3/12/19

The purpose of this lab was to learn how to use binary search trees. We needed to draw a binary tree, search a tree iteratively, build a binary search tree using a sorted list as an input, create a sorted list using a binary search tree as an input and print by depth of a tree.

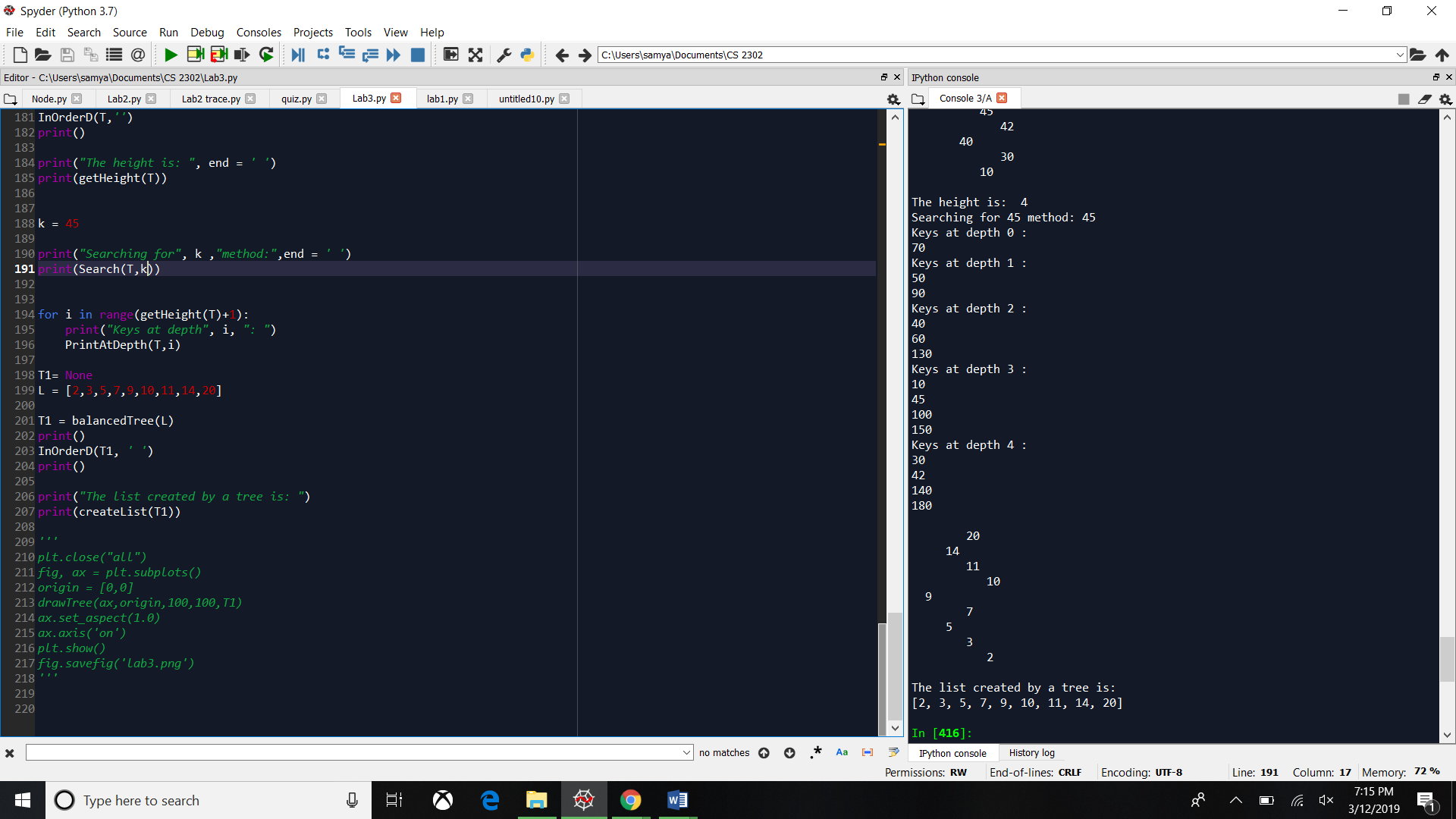
Draw Binary Search Tree:

Unfortunately, I was unable to complete this method, my approach to drawing this tree was to reuse code from lab 1 but I could not figure out how to draw the circles and add the numbers as inputs and make them appear in the drawing.

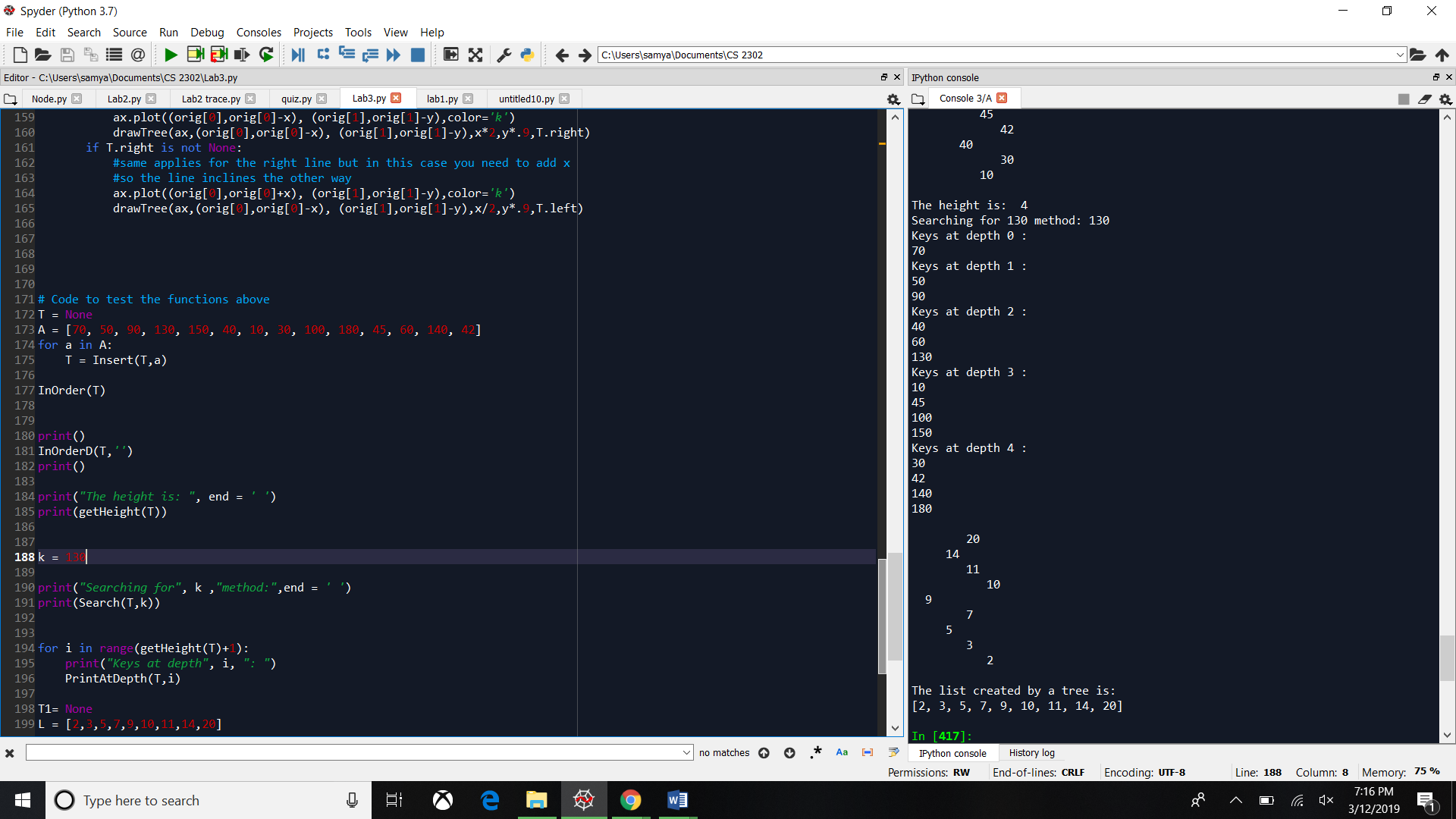
Iterative Search:

This method was the easiest to make. If the tree was empty, then it return nothing. For it to work iteratively I used a temporary variable of it to iterate, while the new variable is not empty then it would go inside the loop. If the key or the number you were looking for was the same as the variable’s item, then it would return that item, but if the key was smaller than the item, then it would search in the left part of the tree and if it was bigger it would check the right side of the tree.

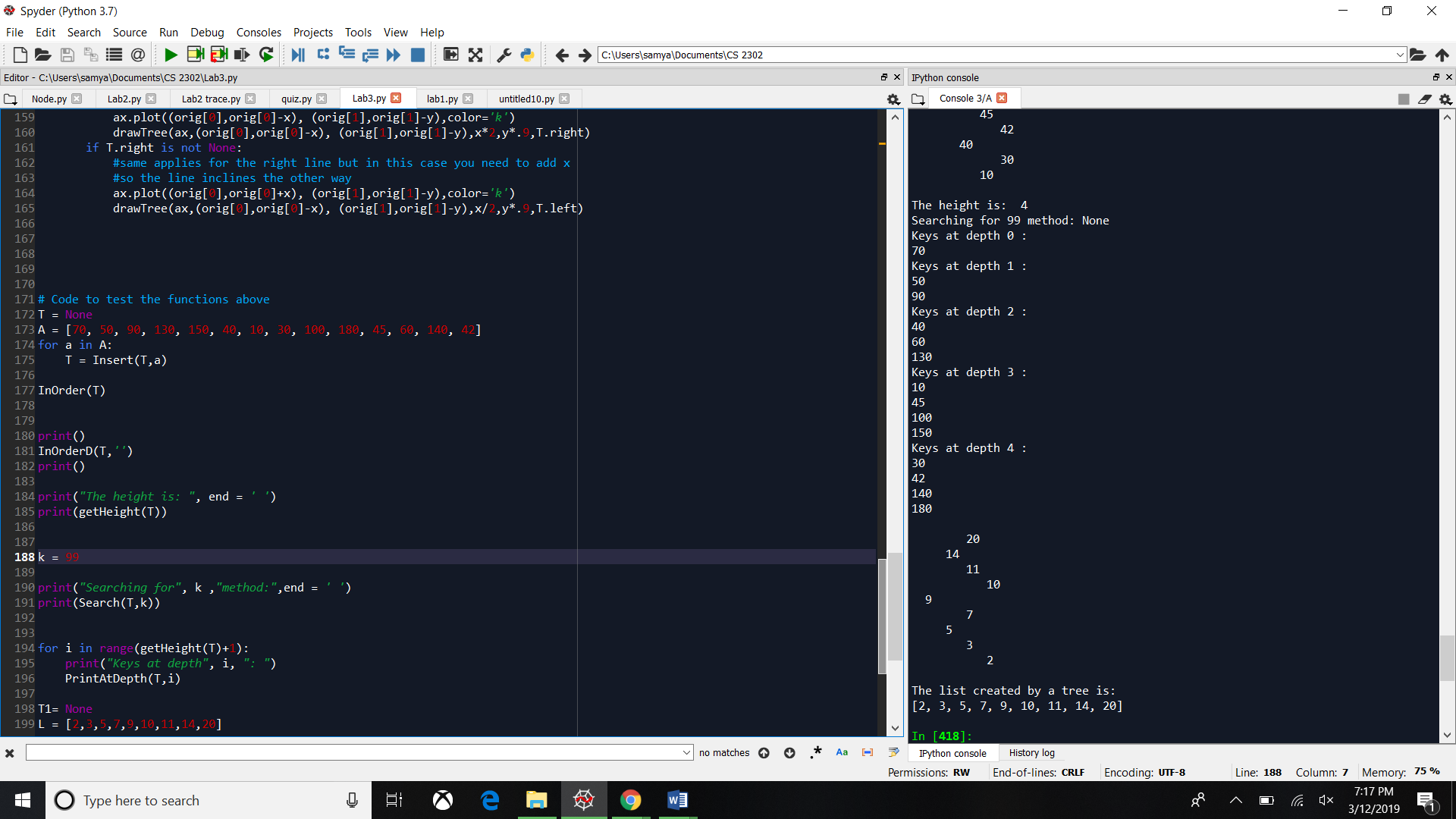
Looking for k=45:



Looking for k=130:



Looking for k=99:

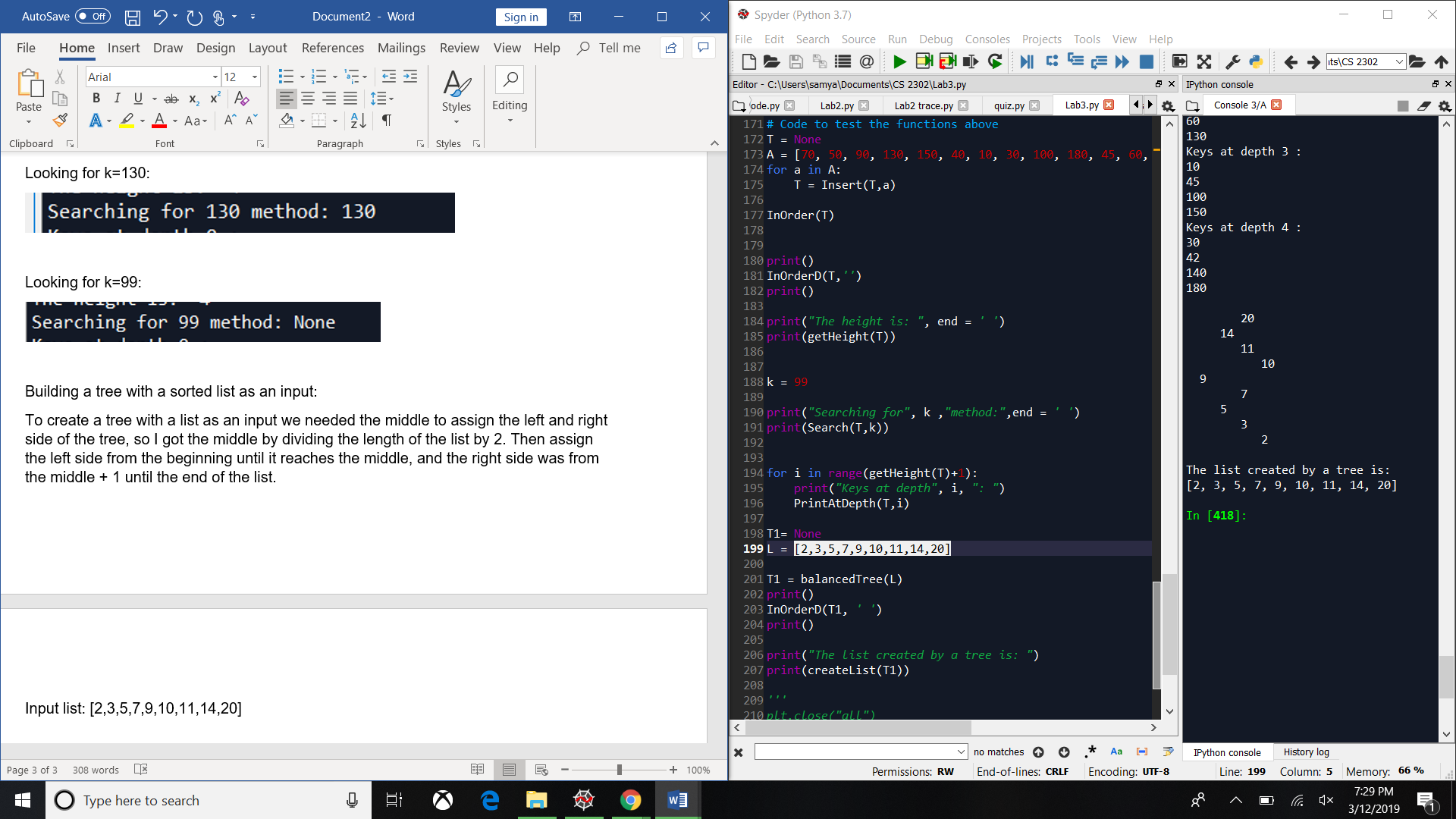


Building a tree with a sorted list as an input:

To create a tree with a list as an input we needed the middle to assign the left and right side of the tree, so I got the middle by dividing the length of the list by 2. Then assign the left side from the beginning until it reaches the middle, and the right side was from the middle + 1 until the end of the list.

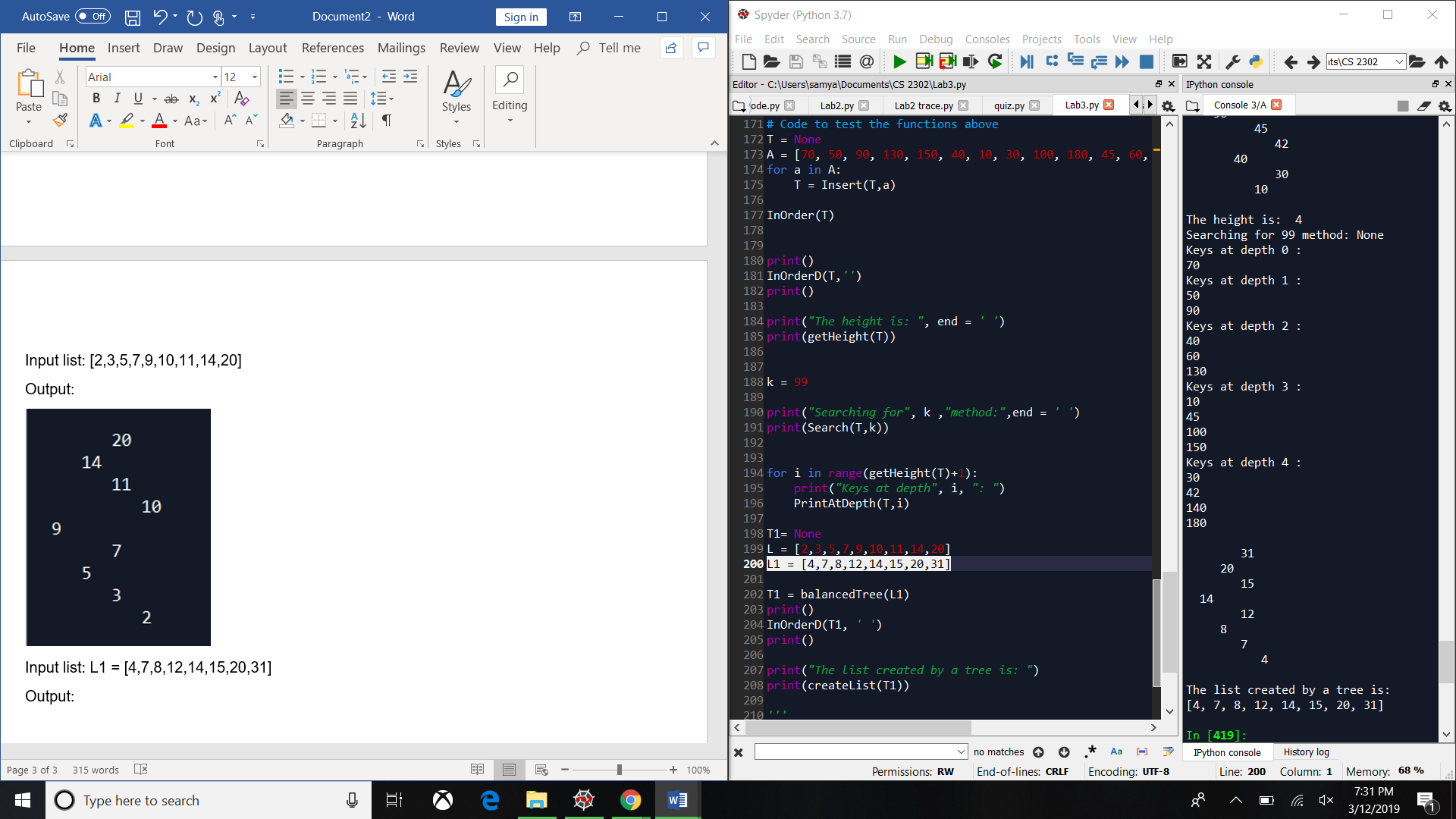
Input list: [2,3,5,7,9,10,11,14,20]

Output:



Input list: L1 = [4,7,8,12,14,15,20,31]

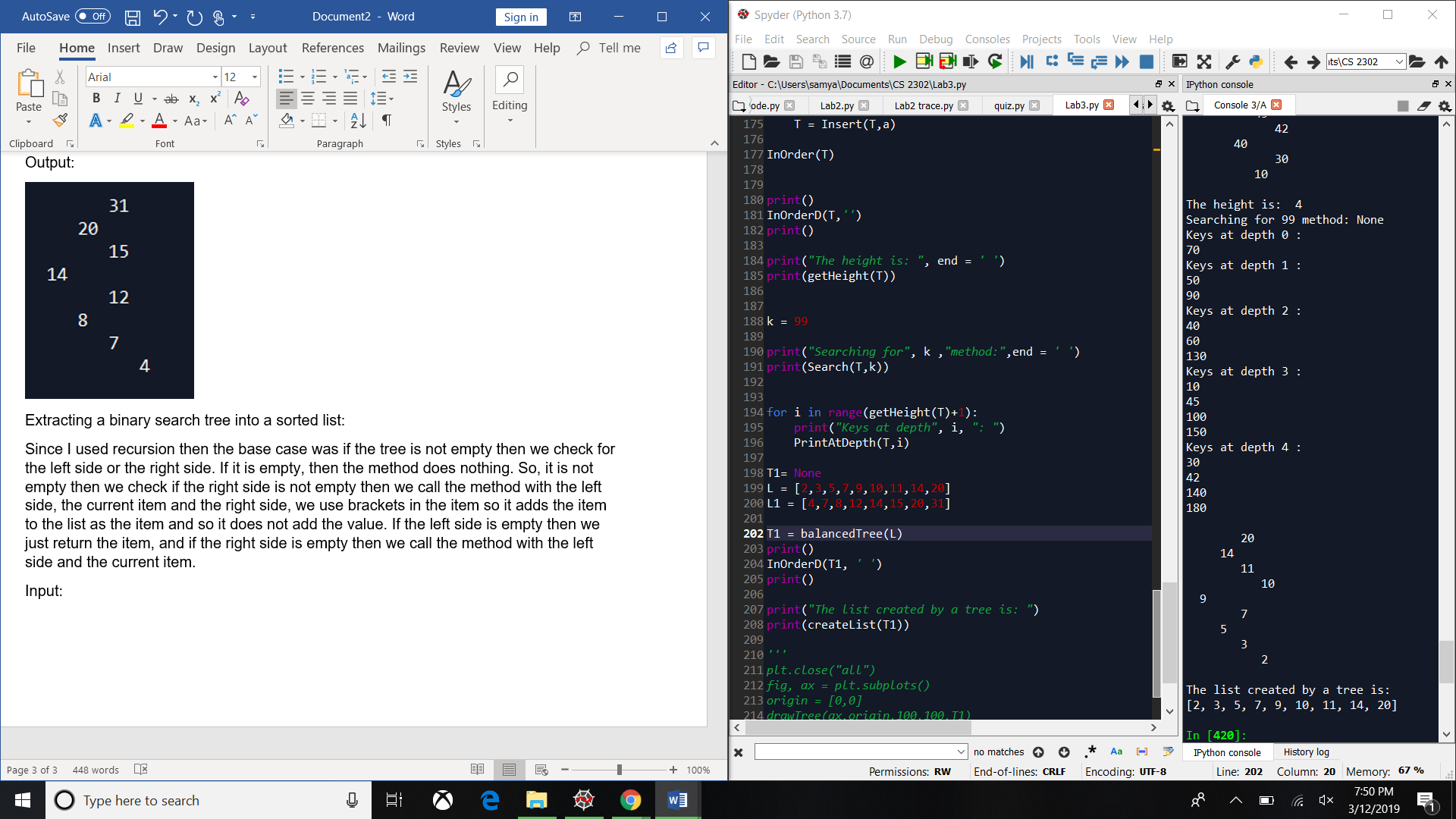
Output:



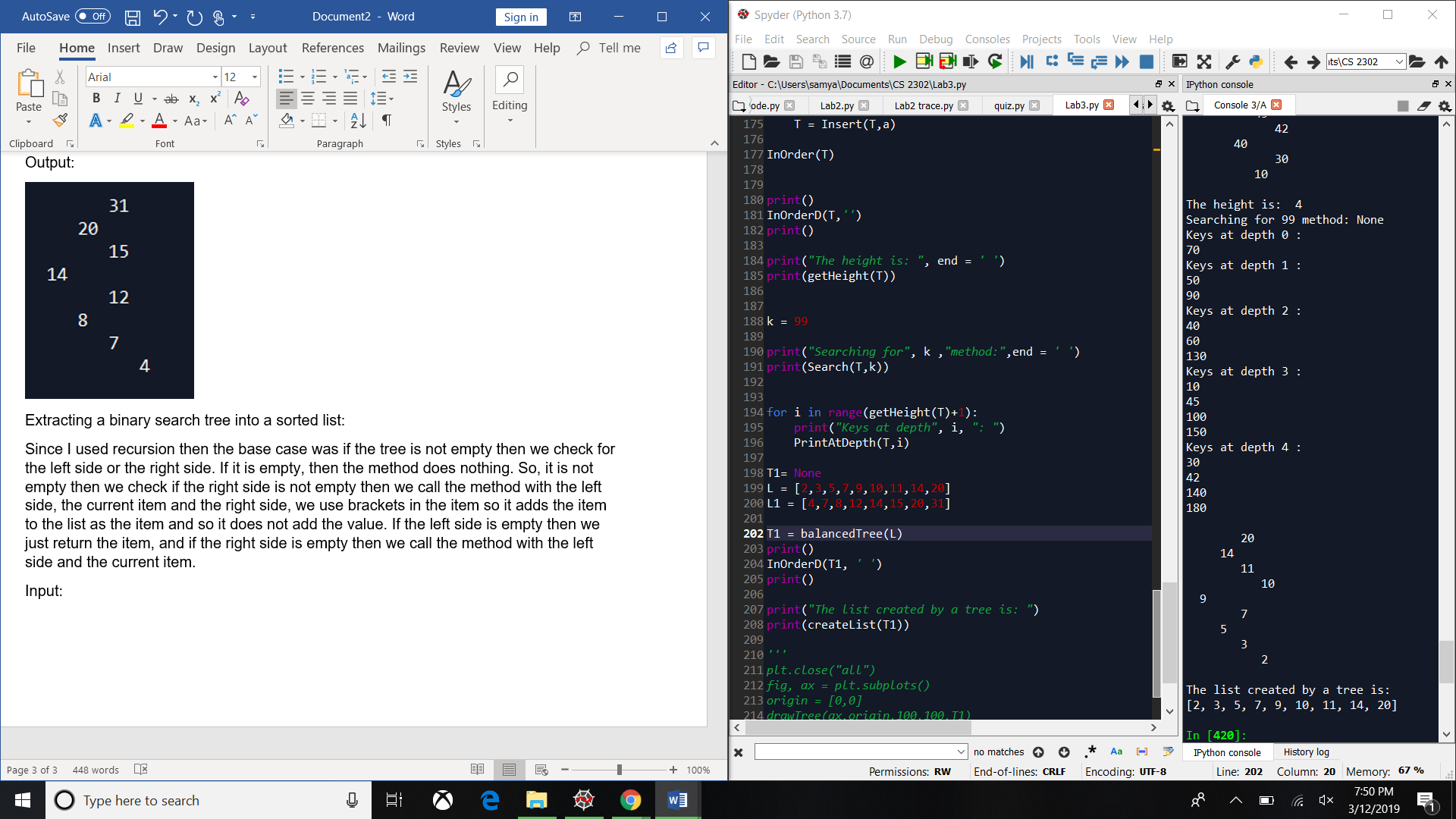
Extracting a binary search tree into a sorted list:

Since I used recursion then the base case was if the tree is not empty then we check for the left side or the right side. If it is empty, then the method does nothing. So, it is not empty then we check if the right side is not empty then we call the method with the left side, the current item and the right side, we use brackets in the item so it adds the item to the list as the item and so it does not add the value. If the left side is empty then we just return the item, and if the right side is empty then we call the method with the left side and the current item.

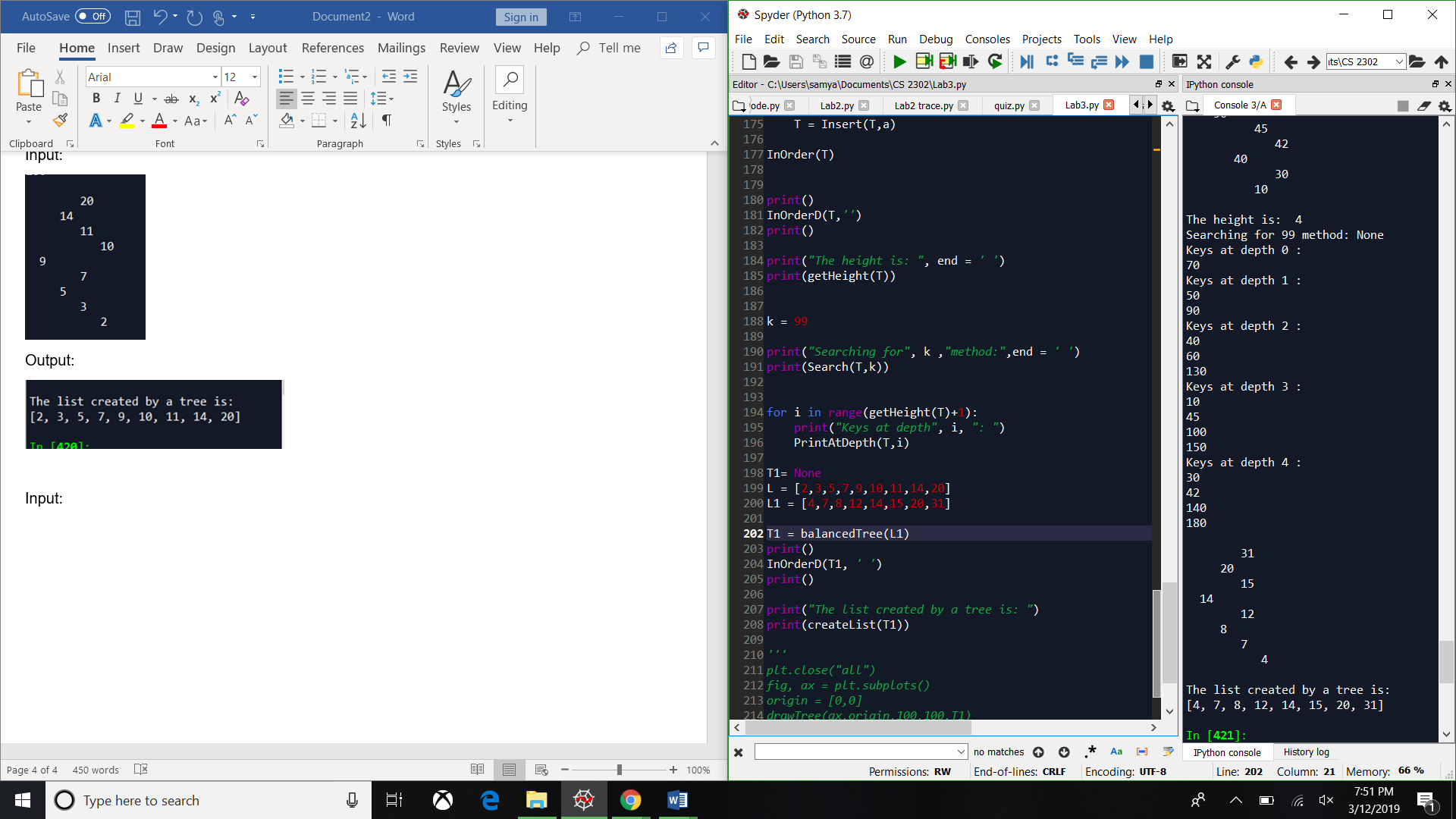
Input:



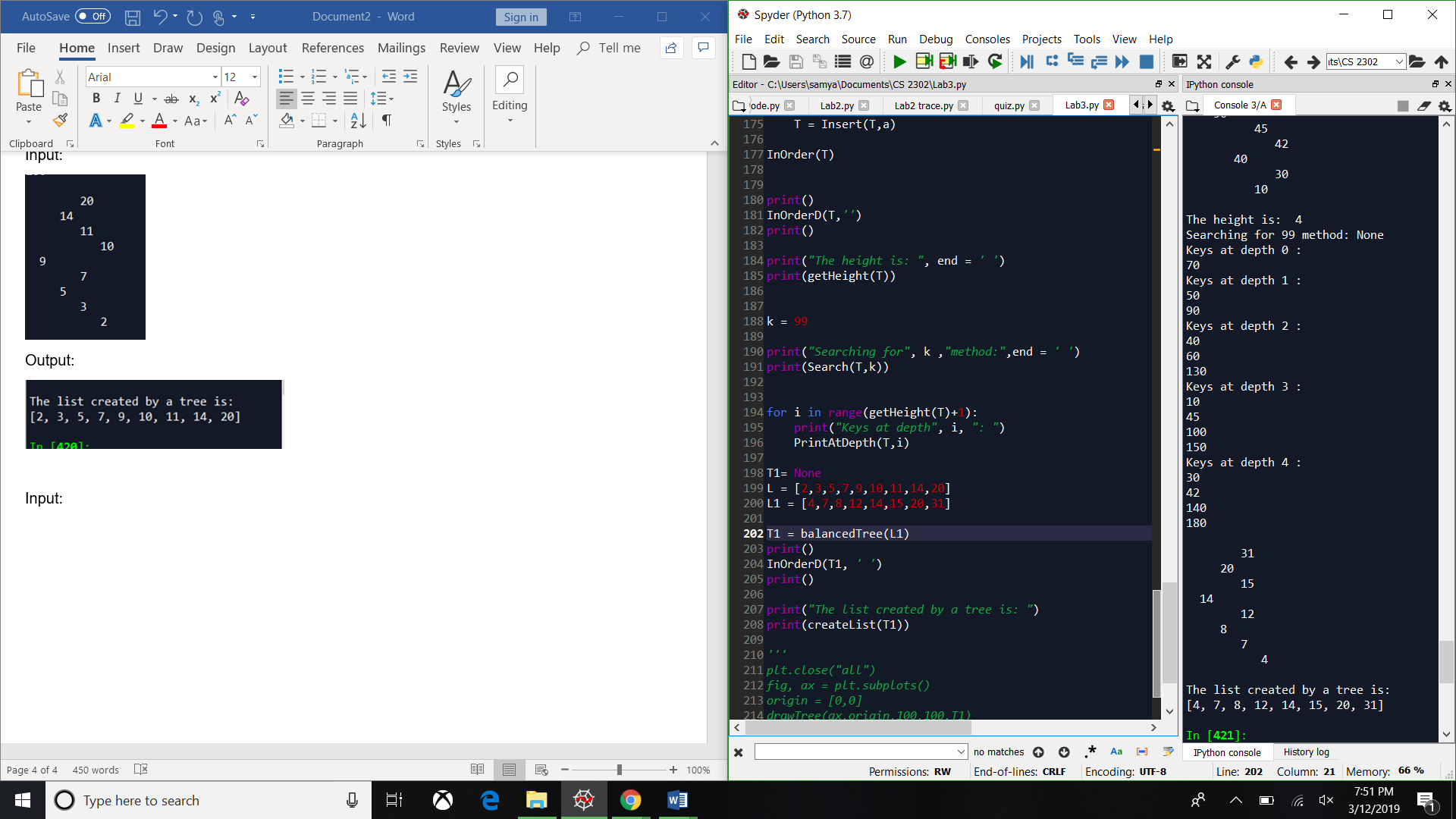
Output:



Input:



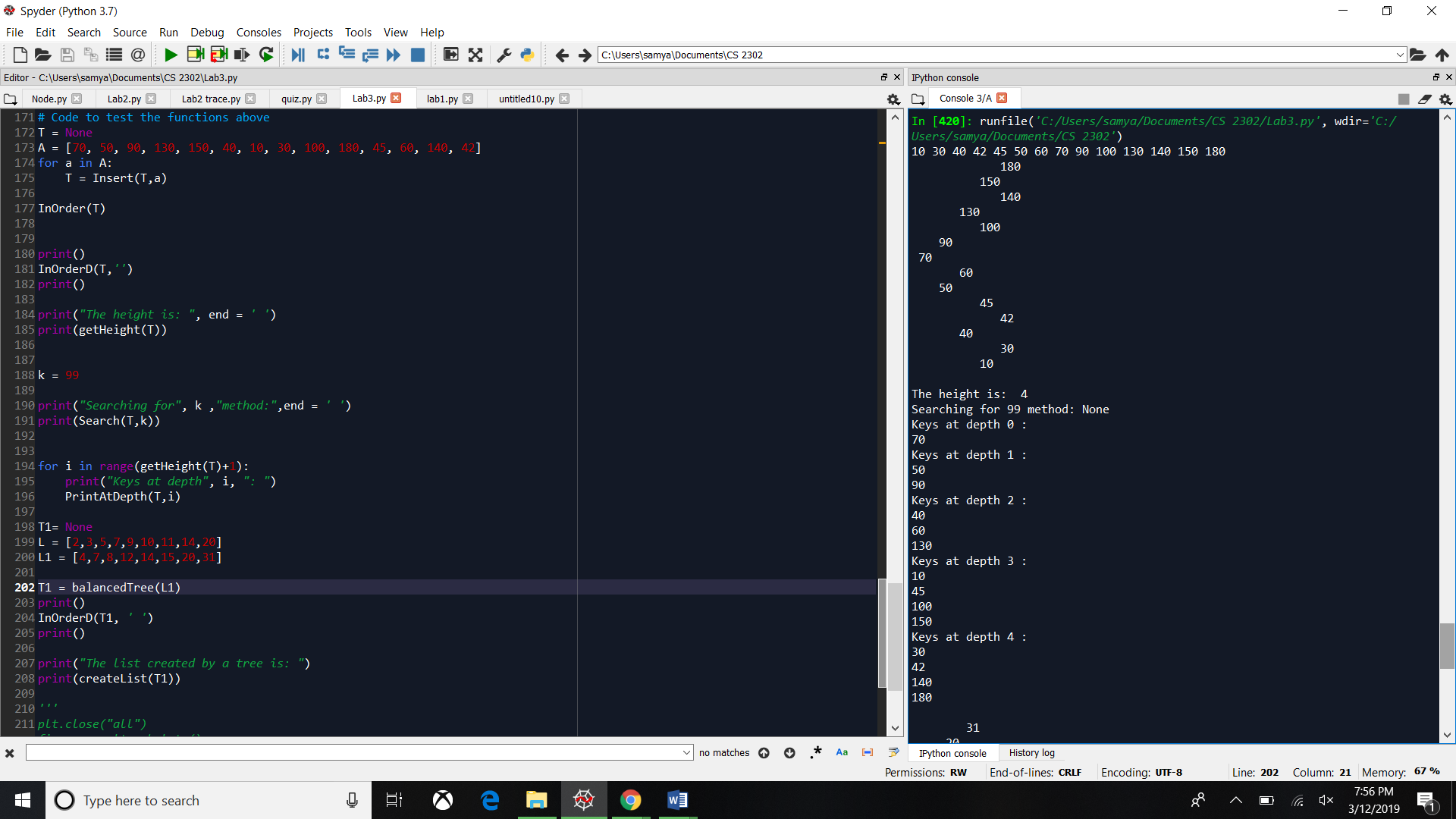
Output:



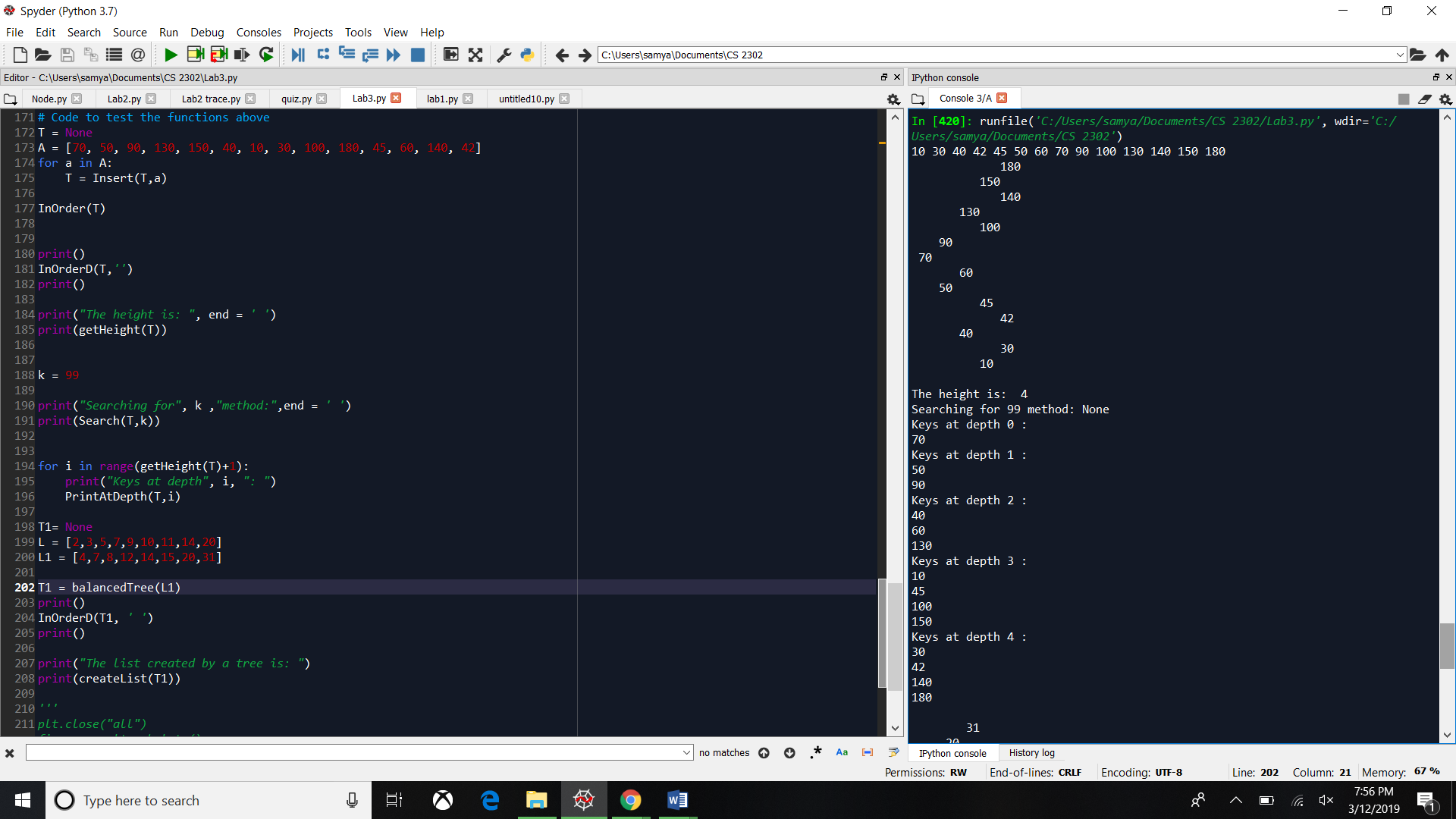
Print by depth:

If the tree is empty then the method simply returns none, if the depth is equal to 0 it means it reached the root, then it returns the item that is in that depth. If it is not any of those cases then we call the method twice, one with the left side subtracting 1 to the depth and the other with right side and also subtracting 1 to the depth. For the method to print by each depth, I called the method in a for loop and use the i to show at what depth it is printing.

Input:



Output:



Code:

"""

@author: Samuel Chong

Professor Olac Fuentes

Lab 3

3/11/2019

TAs:Anindita Nath, Maliheh Zargaran

In this lab we programmed how to search a tree iteratively,

build a tree with a list as an input, create a list with a tree

as an input,print by depth in a tree and draw a tree

"""

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 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 SmallestL(T):

# Returns smallest item in BST. Returns None if T is None

if T is None:

return None

while T.left is not None:

T = T.left

return T

def Smallest(T):

# Returns smallest item in BST. Error if T is None

if T.left is None:

return T

else:

return Smallest(T.left)

def Largest(T):

if T.right is None:

return T

else:

return Largest(T.right)

def Find(T,k):

# Returns the address of k in BST, or None if k is not in the tree

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

return T

if T.item<k:

return Find(T.right,k)

return Find(T.left,k)

def FindAndPrint(T,k):

f = Find(T,k)

if f is not None:

print(f.item,'found')

else:

print(k,'not found')

def Search(T,k):

if T is None: #if the tree is empty then return none

return None

current = T

while current is not None: #while the tree still has elements

if k == current.item: #if the key is found in the current item, return the item

return current.item

elif k < current.item: #if k is less than the current item then go left

current = current.left

else: #if k is greater than the current item then go right

current = current.right

return None

def PrintAtDepth(T,d):

if T is None: #if the tree is empty

return None

if d == 0: #if the depth is 0, print the item

print(T.item)

else: #else subtract 1 from the depth and go left and right

PrintAtDepth(T.left,d-1)

PrintAtDepth(T.right,d-1)

def getHeight(T):

if T is None: #if it is empty return 0

return 0

count = 0

current = T

while current is not None: #while the tree is not empty

count = count + 1 #add 1 everytime it iterates

if T.left is not None: # if the left side is not None then go left

current = current.left

elif T.right is not None:#if the right side is not None then go right

current = current.right

return count #return the counter

def balancedTree(L):

if not L:#if there is nothing return None

return None

mid = len(L)//2 #designates the left and right side

T = BST(L[mid]) #construct the tree

T.left = balancedTree(L[:mid]) #recursive call for the left side

T.right = balancedTree(L[mid+1:]) #recursive call for the right side

return T

def createList(T):

if T is not None: #if the Tree is not empty

if T.right is not None: #if the right side is not empty then make a recursive call with the

#left side, the item and the right side

return createList(T.left) + [T.item] + createList(T.right)

if T.left is None: #if the left side is None then return the item

return [T.item]

else: #if the right is None

return createList(T.left) + [T.item]

def drawTree(ax,orig,x,y,T):

while T is not None:

if T.left is not None:

#to draw a line to the left then we must subtract x to our original 'x'

#coordinate and subtract y to out original 'y' in order for the line

#to go down or else it would be a straight line

ax.plot((orig[0],orig[0]-x), (orig[1],orig[1]-y),color='k')

drawTree(ax,(orig[0],orig[0]-x), (orig[1],orig[1]-y),x\*2,y\*.9,T.right)

if T.right is not None:

#same applies for the right line but in this case you need to add x

#so the line inclines the other way

ax.plot((orig[0],orig[0]+x), (orig[1],orig[1]-y),color='k')

drawTree(ax,(orig[0],orig[0]-x), (orig[1],orig[1]-y),x/2,y\*.9,T.left)

# Code to test the functions above

T = None

A = [70, 50, 90, 130, 150, 40, 10, 30, 100, 180, 45, 60, 140, 42]

for a in A:

T = Insert(T,a)

InOrder(T)

print()

InOrderD(T,'')

print()

print("The height is: ", end = ' ')

print(getHeight(T))

k = 99

print("Searching for", k ,"method:",end = ' ')

print(Search(T,k))

for i in range(getHeight(T)+1):

print("Keys at depth", i, ": ")

PrintAtDepth(T,i)

T1= None

L = [2,3,5,7,9,10,11,14,20]

L1 = [4,7,8,12,14,15,20,31]

T1 = balancedTree(L1)

print()

InOrderD(T1, ' ')

print()

print("The list created by a tree is: ")

print(createList(T1))

'''

plt.close("all")

fig, ax = plt.subplots()

origin = [0,0]

drawTree(ax,origin,100,100,T1)

ax.set\_aspect(1.0)

ax.axis('on')

plt.show()

fig.savefig('lab3.png')

'''

I, Samuel Chong, sign the academic honesty certification. This is my work and only my work. No external help was used for this lab. Also, this report was made by me and no collaboration was made.