Luis Renteria

ID 88740232

CS 2302

Dr. Fuentes

Lab Report 3

**Introduction**

The obective of our third lab assignment was to better understand binary trees by creating diferent methods to manipulate the trees. Our tasks included printing a diagram of a binary tree, using iteration to search through the tree, building a balanced tree through the use of a list, extracting the elements from the tree into a sorted list, and printing all items in a tree by depth.

Side Note: Much of my for work this lab is incomplete, and some completely missing. In this report I will do my best to explain what I was able to understand from the parts of the assignment that are incomplete.

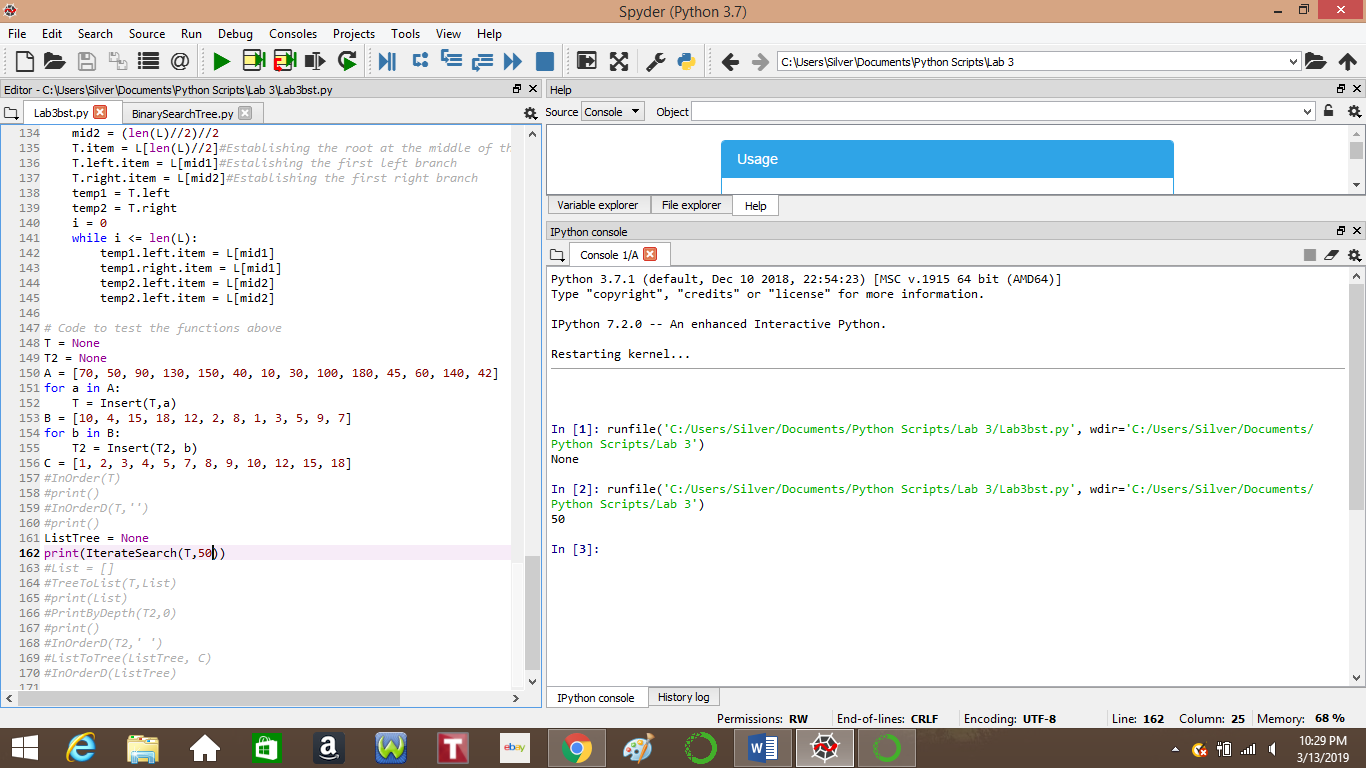
**Binary Tree Diagram**

The first part of this lab, was displaying a binary tree represented by bubbles containing each element and lines drawn to other bubbles to represent the left and right branches. This diagram was to be made using the pyplot functions in order to draw functions and fill them with the correct data, as well as draw the lines connecting each bubble.

**Iterative Search**

The second part of our assignment was to create a search function using iteration. Professor Fuentes had already introduced us to a search function using recursion, so using that as a base I created the iterative version. First, I set a temporary variable to traverse the tree. Using temp, I made a while loop that continued as long as temp was not null. Three if statements within the loop compared the seached value, k, to the item of the current branch. If the item was greater than the current branch, temp was sent to the right, if k was smaller, temp was sent to the left. Once k was found within the tree, k was returned. If k was not found, None was returned.

This screenshot shows the IterateSearch method being called with tree T and searching number 50. The method returns 50, which signifies that the number has been found in the tree



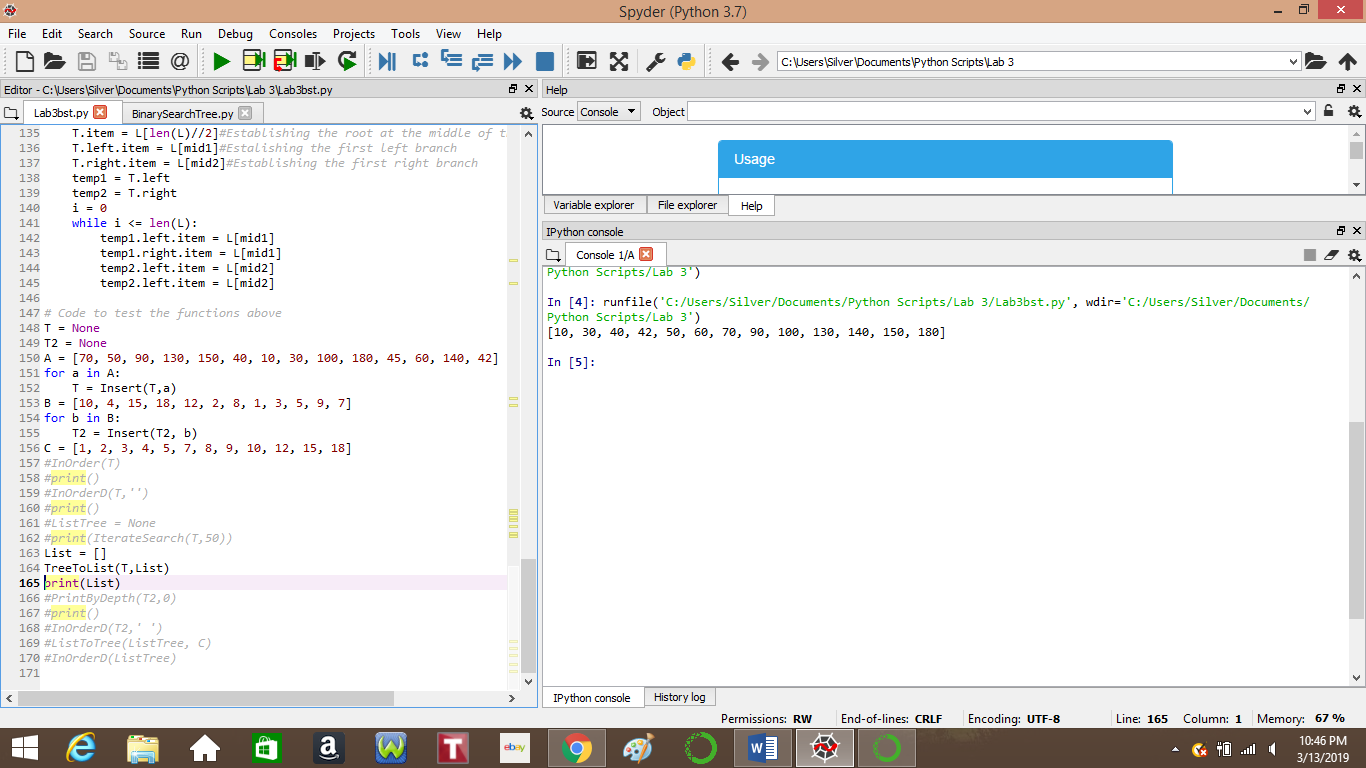
**Building A Tree**

For part 3, we were requird to build a balanced tree using a sorted list. I was not able to complete this part of the lab, but I will attempt to explain my thought process and how I approached the problem. First, I decided to write out a list of 10 elements ranging from 1-10. From there , I realized that in order to get a balanced tree, the root would have to be the element in the middle of the list, to ensure even distribution. I also realized that for every left and right child, the index of the element would have to be between the index of the root and the index of the last or first element, depending on which side was being constructed. This method would result in a balanced tree, but I ultimately had no clue on how to execute such a thing using code.

**Tree To List**

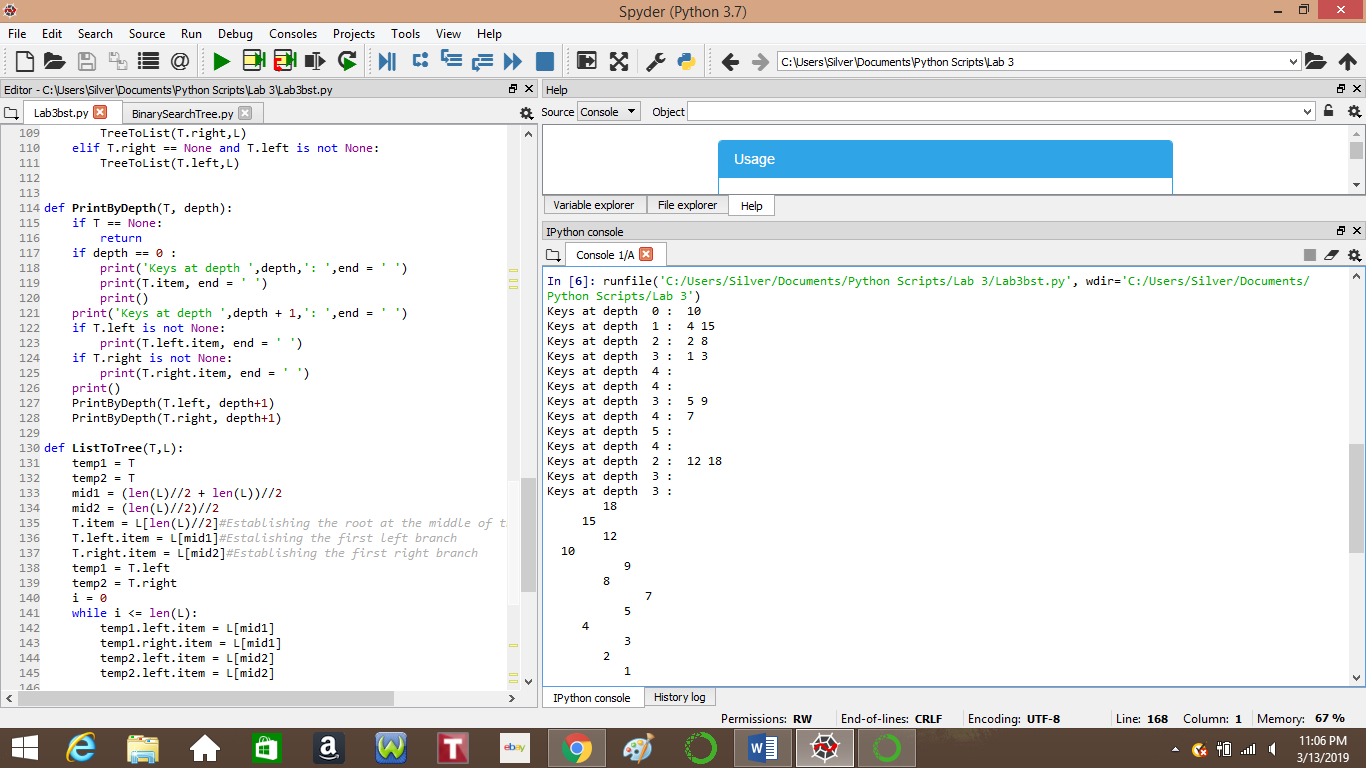
The fourth part of our lab was creating a sorted list from a binary tree. Once again I drew a binary tree in order to understand where to start. I understood that the very first element in the list should be the leaf farthest left and that the very last element in the list should be the leaf furthest right. From there, I considered every case of branched, such as the left side being null while the right side was not, or neither being null. I was able to understand that if an element was a leaf, it should be appended to the list, and if a node has both a right and left, the left should be appended first, the node itself, then the right. Using this understanding, I was able to use various if statements to make recursive calls based on whether or not each node had left or right branches.

The picture below shows tree T being used to create an ascending list of elements.



**Printing By Depth**

The final part of this lab was creating a method that printed the elements in each depth of a tree. I approached this by having a variable, depth, keep track of the depth at each recursive call. In order to print all elements, one recursive call was made for each branch. If a branch was not null, then the item would be printed. Although this method does print out every element in the tree, it does not do so in an orderly manner. In order to demonstrate what I mean, I have included the picture below that also includes the tree itself.



**Conclusion**

Although I struggled and was unsuccessful in completing the lab, I do now have a better understanding of traversing a binary search tree. I feel I do need more practice with this particular data structure, but I am hoping that more examples will lead to a better understanding of how to efficiently use a binary search tree.

**Academic Statement:**

I hereby certify that all work presented is my own.

Signed Luis Renteria

**Appendix**

The following is the code that was used in this lab, as taken from the class website.

# Code to implement a binary search tree

# Programmed by Olac Fuentes

# Last modified February 27, 2019

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')

# 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)