**Lab 3 Report**

**Introduction**

The task is to manipulate binary trees in order to print at depth, iterative find,extract to array, and build a bst to a Sorted list.

**Solution**

To iterative find while t.item is not none you compare k to t.item if k is less go t.left if k is greater go t.right and if it is equal return k

To print at depth first you print T.item at depth 0 and at the same time each time you go to left and right add one to depth and print again unless it is none

Given a sorted array you take the middle element as the rood and then add the left and right element.

Given a bst to create a sorted array extract in order transversal format.

Conclusion

To search for an item compare the elements left if k Is less than item and right if k is greater.

Most operations can be preformed in big O log n form

**Academic Certification statement**

I certify that the attached written or electronically transmitted material is my own work. I further certify that (a) I have formally cited or otherwise fully acknowledged the quotations, ideas, and wording used here from other sources, whether published or unpublished, in written or electronic form, (b) I have engaged in no falsification or misrepresentation of data or experience in this submission, (c) I have disclosed any collaboration not specifically authorized by the instructor.

I understand that the relative autonomy of the legal profession carries with it special obligations of self-regulation as outlined in the Columbia Law School Bulletin and that any violation of this certification will subject me to discipline, including possible suspension or expulsion by the Law School, declination to certify for admission to the Bar, and sharing of information about discipline with Bar admissions committees. I further understand that the Law School is required to notify Bar admissions committees if a student has been the subject of disciplinary proceedings, regardless of the outcome of those proceedings.

**Appendix**

#Justin Ruiloba

#4/4/19

#lab 3

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 treetosortedlist(T,A,i):

if T == None or i>len(A):

return A

treetosortedlist(T.left,A,i);

print(A[i],i)

A[i]=T.item

i+1

treetosortedlist(T.right,A,i)

def sortedlist(T,A):

T=None

for i in range (len(A)):

if i==A[len(A)//2]:

T= A[len(A)//2]

elif T.item<A[i]:

T.right = A[i]

elif T.item>A[i]:

T.left= A[i]

def printD(T,D):

if T.item is None:

return D

print("D at depth",D,T.item)

def searchi(T,k):

while T.item is not None:

if T.item == k:

return T.item

if T.item<k:

T=T.right

if T.item>k:

T=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)

InOrderD(T,'')

print()

print(searchi(T,40))

print(printD(T,0))