**Introduction**

The purpose of this assignment was to understand and manipulate Binary Search Trees for the full comprehension of the data structure. The lab entailed the plotting of the tree on a graph, then produce a balanced binary tree from a sorted list. The next step requires an iterative version of the Search function. The next step necessitates the production of a sorted list from a binary tree. Finally, the last step asks that the depth of each tree is printed form left to right.

**Proposed solution design and implementation**

Beginning with the draw\_trees() function, the best method for implementation found starts with a base case checking if the node T is none, if it is, it will return to the caller function. In the case that T is not none, it will plot a circle, and the current item of the node within said circle. Once the previous steps are completed, there are if statements that check if the right and left nodes exist, if they do exist, they will draw the branches of each respective branch, should their nodes exist, otherwise returning to the caller function if no nodes exist.

Secondly, the assignment calls for the creation of an iterative Search() function. The function would require a loop, which is why a while loop was used. The loop checks if a node T exists within the tree, if such a node exists, it continues to check if the item is the element of T, if not, it checks the branches. When checking the branches, the while loop checks if the sought-after item is lesser than or greater than the current node’s item. In the instance that the key is greater than the current node’s element, the loop moves to the right node, should it exist, if the key lesser, it will move to the left node, should it exist. The function will return the item of the node when the key is found, else in the case that the key is not within the tree, the loop will break and return None.

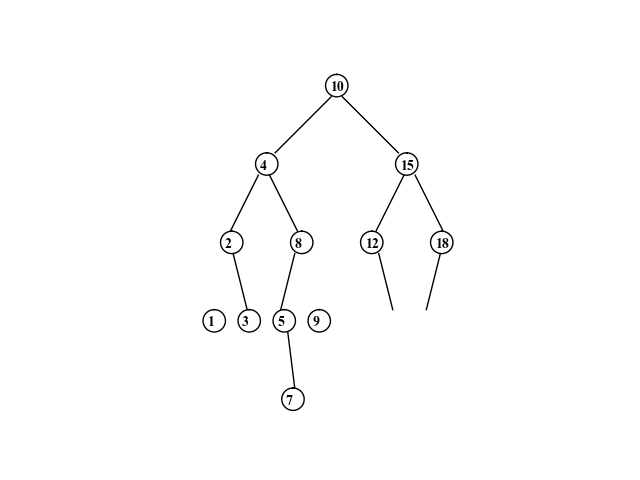
The next objective requests the creation of a balanced binary tree from a sorted list. Using recursion, the first line of the function BalTree() checks if the list exists, it checks that the length is not less than or equal to 0. Given a list of a length greater than 0, the function halves the list’s length and saves that under the variable name “root”. The name “root” was given since the middle element of the list will become the root of the tree. The left subtree consists of every element from index 0 to index root of the list, then the right subtree consists of every element from index root+1 to the end of the list. This is done recursively until the subsequent list’s length is equal to 0.

The following objective requires that a given tree’s elements be imported to a sorted list. The function SortedList() returns a list of length equal to the number of elements in the tree, but returns the list in Post-Order not in In-Order. The attempt of In-Order insertion was brought to a halt since the lack of preparation meant there was a lack of time to produce a proper solution for this method.

The final task to complete asks for the printing of the elements of the tree ordered by depth. This one was not done completely correct, as the function atDepth() prints “Keys at depth d:” (where d is an arbitrary depth) along every element of the tree in its own line. The function should be printing tree-height number of lines along with each element of each respective depth

**Experimental results**

The produced tree is similar to the desired tree given through the lab assignment, the only difference being the branches of 12 and 18, respectively, should be the left and right branches of 2 and 8, respectively.

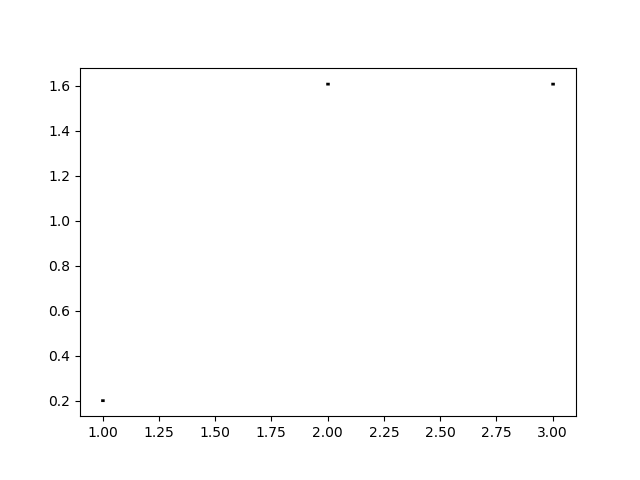


The figure should take O(log n) time to complete

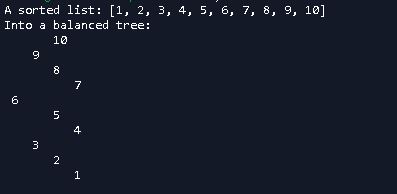
The Search() function returns the item if found or None if it is not found, outside of the function, there are if and else statements that print “Not found” if no element in the tree exists equal to the key, and “found” if the key is found within the tree.



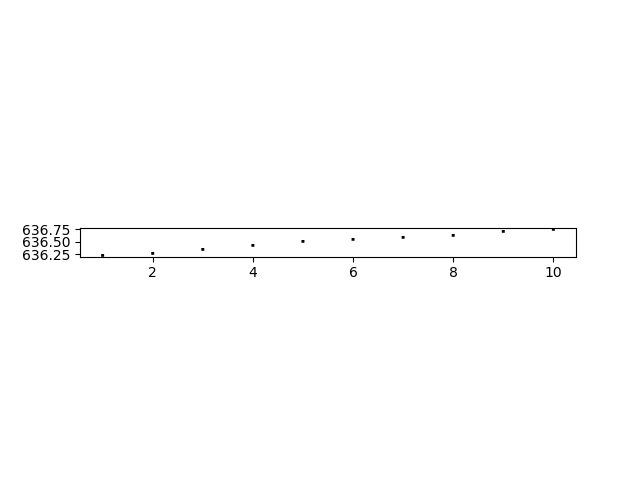
The running time should come out to O(log n); this is reflective from the diagram



The BalTree() function takes a sorted list and returns a balanced tree, such that no branch is 1 greater than or 1 less than its sibling branch’s length.



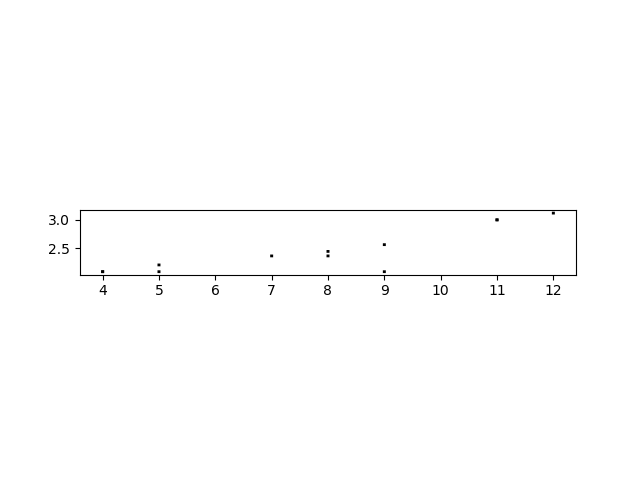
The running time of the BalTree() function should be linear, O(n), this is shown in the following graph.



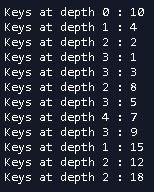
The SortedList() function returns a Post-Order sorted list, it should return an In-Order sorted list.



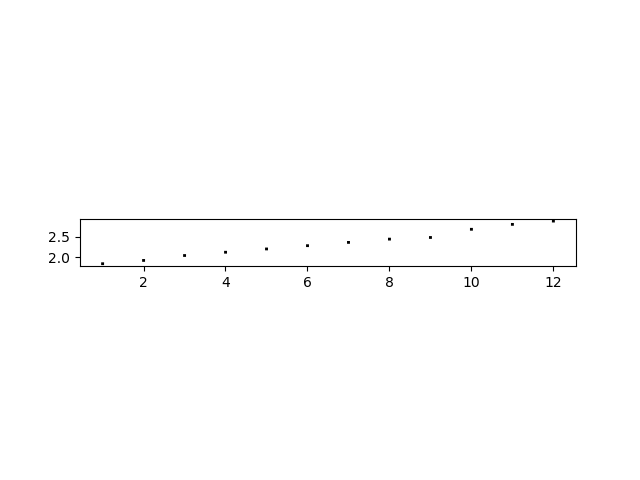
The running time should be, and is, linear O(n) as shown in the diagram below.



The atDepth() function prints “Key at depth d:” (Where d is the given depth relative to the given key) for each and every element in the tree, it should print tree height number of lines with each key printed with every other key in their depth.



The running time comes out to be O(n), this is shown below.



**Conclusions**

Upon completion of this assignment, multiple ideas can be grasped: Take time for each step, do not over-complicate each step, and lastly, persist with finding a solution. Having to take on what seems to be a difficult task was stressful, yet it was plausible to accomplish successfully with patience and study, which was understood nearing completion. The most intimidating aspect of the lab was the plotting of the tree on a graph, causing over-complicated “solutions” to be introduced. Meeting these dead ends and acknowledging what was thought to be the best answers, it was realized that there was a lack of patience, and most of the functions were convoluted, in fact many did not meet the desired goals expected. Having completed this lab, more studying can be done, and more time can be made to approach these problems with greater ease; this will be greatly considered for future assignments.

**Appendix**

"""

Course: CS 2302

Author: Jacob Montenegro

Lab: 3

Instructor: Dr. Olac Fuentes

T.A.: Anindita Nath, Maliheh Zargaran

Date of Last Modification: 3/8/2019

Program's Purpose: Get aqcuainted with manipulating Binary Search Trees

"""

import matplotlib.pyplot as plt

import numpy as np

import math

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(BST):

if BST is not None:

InOrder(BST.left)

print(BST.item,end=' ')

InOrder(BST.right)

def PrintTree(BST,space):

if BST is not None:

PrintTree(BST.right,space+' ')

print(space,BST.item)

PrintTree(BST.left,space+' ')

def Height(BST):

if BST is None:

return -1

LHeight = Height(BST.left)

RHeight = Height(BST.right)

if LHeight > RHeight:

return LHeight + 1

else:

return RHeight + 1

def NumOfNodes(BST):

if BST is None:

return 0

return 1 + NumOfNodes(BST.left) + NumOfNodes(BST.right)

#Creates a balanced tree from a sorted list.

def BalTree(L):

#Base-Case: checks if there are any elements in L, returns if empty

if len(L)<=0:

return

#Finds the middle element

root = (len(L))//2

#makes middle element the head of the tree

B = BST(L[root])

#The left sub-tree is then created with each element before the middle element

B.left = BalTree(L[:root])

#The right sub-tree is then created with each element after the middle element

B.right = BalTree(L[root+1:])

#returns the completed balanced-tree

return B

#Iterative version of Search function

def Search(T,c):

while T is not None:

#If T is None, the loop will break and return None, thus, c is not found.

if T is None:

break

#Returns item if it's in T

if c == T.item:

return T.item

if T is not None:

#Searches right sub-tree if c is greater

if c > T.item:

T=T.right

if T is not None:

#Searches left sub-tree if c is lesser

if c < T.item:

T=T.left

return None

#This functon prints each element at each depth

def atDepth(T,d):

#Checks if a tree exists

if T is None:

return -1

print('Keys at depth',d,':',T.item,end=' ')

print()

d+=1

atDepth(T.left,d)

atDepth(T.right,d)

#This method draws a binary tree, it has an x,y, change in x (dx) and cahnge in y (dy), as well as a k.

#In this function, cx and cy are present, they serve the purpose of shifting each of the circles created,

#As well as shifting each element of the tree to their respective circles.

def draw\_trees(ax,x,y,dx,dy,k,T,center,r,cx,cy):

#This checks to see if a given tree has an element

if T is not None:

#This part focuses on producing the circles as well as the items

q,t = circle(center,r)

ax.plot(q+cx,t+dy-4-cy,linewidth=1,color='k')

we = T.item

font = {'family':'Times New Roman','color':'black','size':10}

ax.text(center[0]-2.5+cx, center[1]+dy-cy-6,we, fontdict=font)

if T.right is not None:

#The ax.plot line prints a branch, the second ax.plot line prints a reflection of this ax.plot line.

ax.plot(x\*dx+2,y,linewidth=1,color='k')

draw\_trees(ax,x+k\*dx,y-dy,dx/2,dy,k\*4,T.right,center,r,cx+25\*dx,cy+28)

if T.left is not None:

ax.plot(-x\*dx-2,y,linewidth=1,color='k')

#k is added since I noticed that the drawing kept shifting inward by multiples of 2, 4, 16, etc.

#I realized that in order to shift left and right, I would need to continuously add and subtract dx by multiples of 4, which is why I am using k.

draw\_trees(ax,x-k\*dx,y-dy,dx/2,dy,k\*4,T.left,center,r,cx-25\*dx,cy+28)

#This function was a given by Dr. Fuentes for the first lab.

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

#This is an attempt to create a sorted list from a BST, this function should return a sorted list, instead it returns the left, right, then middle element of a subtree

def SortedList(T,L):

#Base-Case, returns the list

if T is None:

return L

#Iterates through left subtree(s)

if T.left is not None:

L[:-1] = SortedList(T.left,L[1:])

#adds current node to end of list (This is because the last nodes to be added would be the root/right nodes)

L[-1] = T.item

#Iterates through the right subtree(s)

if T.right is not None:

L[:-1] = SortedList(T.right,L[1:])

#returns list

return L

T = None

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

for a in A:

T = Insert(T,a)

plt.close("all")

p = np.array([[0,20],[20,0]])

fig, ax = plt.subplots()

draw\_trees(ax,p[0],p[1],1,28,52,T,[0,0],4,0,0)

ax.set\_aspect(1.0)

ax.axis('off')

plt.show()

#fig.savefig('tree.png')

W = None

L = [1,2,3,4,5,6,7,8,9,10]

print('A sorted list:',L)

print('Into a balanced tree:')

W = BalTree(L)

PrintTree(W,'')

print()

d = 20

w = Search(T,d)

if w is None:

print(d,': Not found')

else:

print(d,': found')

print()

atDepth(T,0)

print()

Q = [None] \* NumOfNodes(T)

B = SortedList(T,Q)

print('Sorted List:',B)

I certify that this project is entirely my own work. I wrote, debugged, and tested the code 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. Signed, Jacob M. Montenegro