#jeff davidson

#homework 4

# insertion sort program using pq

class priorityQueue():

def \_\_init\_\_(self):

self.queue = []

def isEmpty(self): # check if empty

return len(self.queue) == 0

def insert(self,element): # insert element from sequence into pq

if self.isEmpty(): # check if empty to add first element into pq

self.queue.append(element)

else: #if not empty compare values to place element in right index

for i in range(0, len(self.queue)):

if self.queue[i] > element: # if new element is less than find the spot it belongs at

temp = self.queue[i]

self.queue[i] = element

self.queue.insert(i+1,temp)

break

if self.queue[len(self.queue)-1] < element: # if new element is larger than last element add it at the end

self.queue.append(element)

print(self.queue) #print to show that it is inserting as a sorted list

def delete(self): # delete from pq and add to sequence

temp = self.queue[0]

del(self.queue[0])

return temp

#start pq and sequence

queue = priorityQueue()

sequence = [7,4,8,2,5,3,9]

print(sequence)

n = len(sequence)

# insert into pq

for x in range(0,n):

temp = sequence[0] #remove from sequence

del(sequence[0])

queue.insert(temp)

#show sequence is empty

print(sequence)

#insert back into sequence

for x in range(0,n):

sequence.append(queue.delete())

#print finished sorted

print(sequence)

#queue.printQueue()

------------------------------------------------------------------------------------------------------------------------------------------

#jeff davidson

#tree traversal inorder and preorder

class newNode: #creates a new node to be used to create a tree

def \_\_init\_\_(self, x):

self.val = x

self.left = None

self.right = None

# recursive function to tell where a node is suppose to be pretty much builds the tree to be traversed

def traversed(a, i, root):

if i < len(a): #if i is greater than length of entered array

temp = newNode(arr[i]) # store current value in temp

root = temp # set root of new node to the value of temp

root.left = traversed(a, (2 \* i) + 1, root.left) #recure to set value of left node

root.right = traversed(a, (2 \* i) + 2, root.right) #recure to set value of right node

return root

# function to traverse the tree inorder

def inOrder(root):

if root != None: #base case

inOrder(root.left) #recure to the left of the root

if root.val != "null": # check root before print

print(root.val, end=" ") #prints value

inOrder(root.right) #recure to the right of the root

#function to traverse tree in pre order

def preOrder(root):

if root != None: #base case

if root.val != "null": # check to make sure root is not null

print(root.val, end=" ") #print root value

preOrder(root.left) # recure left of root

preOrder(root.right) # recure right of root

#main

arr = [1,"null",2,"null","null",3] #array with nulls as place holder i couldnt figure out how to do it without them but you said in an email this was fine so i left it as is

n = len(arr) #length of entered array

root = None # set root to none

root = traversed(arr,0,root) #set root to what is returned from the function that builds the tree

#print traverse in these orders

inOrder(root)

print() #new line

preOrder(root)