Jakob Smith

CSC 310

1.)

from \_queue import Empty  
from priority\_queue\_base import PriorityQueueBase  
from positional\_list import PositionalList  
  
"""The following classes (above) were taken from in class examples"""  
class SortedPriorityQueue(PriorityQueueBase): # base class defines \_Item  
 *"""A min-oriented priority queue implemented with a sorted list."""* def \_\_init\_\_(self):  
 *"""Create a new empty Priority Queue."""* self.data = PositionalList()  
  
 def \_\_len\_\_(self):  
 *"""Return the number of items in the priority queue."""* return len(self.data)  
  
 """The add function uses insertion sort to add a key-value pair to the sorted priority queue"""  
  
 def add(self, key, value):  
 *"""Add a key-value pair."""* newest = self.\_Item(key, value)  
 walk = self.data.last()  
 while walk is not None and newest < walk.element():  
 walk = self.data.before(walk)  
 if walk is None:  
 self.data.add\_first(newest)  
 else:  
 self.data.add\_after(walk, newest)  
  
 def min(self):  
 *"""Return but do not remove (k,v) tuple with minimum key.  
  
 Raise Empty exception if empty.  
 """* p = self.data.first()  
 item = p.element()  
 return item.\_key, item.\_value  
  
 def remove\_min(self):  
 if self.is\_empty():  
 print("priority Queue is empty")  
 return  
 item = self.data.delete(self.data.first())  
 return item.\_value  
  
 def \_\_iter\_\_(self):  
 *"""Generate iteration of the map's keys."""* for item in self.data:  
 yield item # yield the KEY  
  
  
test = SortedPriorityQueue()  
test.add(1, 3)  
test.add(4, 2)  
test.add(2, 1)  
test.add(3, 4)  
  
for i in test:  
 print(i, end=' ')

2.)

import math  
  
  
class Node:  
 def \_\_init\_\_(self, key):  
 self.left = None  
 self.right = None  
 self.val = key  
  
  
class Traversal:  
  
 inOrderList = []  
 preOrderList = []  
  
 def inOrder(self, root):  
 # escape if no root  
 if root is None:  
 return  
 else:  
 # traverse left nodes  
 x = (self.inOrder(self, root.left))  
 # add to list if the input was not 'null'  
 if x is not None:  
 self.inOrderList.append(x)  
 # add to list if the root is not 'null'  
 y = root.val  
 if y is not None:  
 self.inOrderList.append(y)  
 # traverse right nodes  
 z = (self.inOrder(self, root.right))  
 if z is not None:  
 self.inOrderList.append(z)  
  
 def preOrder(self, root):  
 # escape if root is none  
 if root is None:  
 return  
 y = root.val  
 # if the root is not None, add it to the list  
 if y is not None:  
 self.preOrderList.append(y)  
 # visit left nodes  
 if root.left is not None:  
 self.preOrder(self, root.left)  
 # visit right nodes  
 if root.right is not None:  
 self.preOrder(self, root.right)  
  
  
root = []  
placeHolder = 0  
user = 0  
print("Type \"Done\" to finish \n")  
  
while user is not 'Done':  
 user = input("Enter an integer: ")  
 if user == "Done":  
 break  
 if user == "null":  
 root.append(Node(None))  
 else:  
 root.append(Node(user))  
  
max = len(root) - math.pow(2, (math.log2(len(root) + 1) - 1))  
  
while placeHolder < max:  
 root[placeHolder].left = root[2 \* placeHolder + 1]  
 root[placeHolder].right = root[2 \* placeHolder + 2]  
 placeHolder += 1  
  
test = Traversal  
test.preOrder(test, root[0])  
test.inOrder(test, root[0])  
print("Inorder:")  
print(test.inOrderList)  
print("Preorder:")  
print(test.preOrderList)