################################################# Question 1 #################################################  
# example list queue for #1  
# 3, 1, 2  
# front....rear  
class MyCircularDeque():  
 # constructor  
 def \_\_init\_\_(self, usize):  
 self.queue = [] # data items  
 # maximum size  
 self.mSize = usize  
 # current size  
 self.size = 0  
 # front index  
 self.f = 0  
 # rear index  
 self.r = 0  
  
 # inserting at the front  
 def insertFront(self, num):  
 # if the queue is full  
 if len(self.queue) == self.mSize:  
 print("False, the Queue is full")  
 # if the queue isn't full  
 elif self.size == 0:  
 self.queue.insert(self.f, num)  
 self.size += 1  
 self.r += 1  
 print("True")  
 else:  
 # the queue isnt full and isnt empty either  
 self.queue.insert(self.f % self.size, num)  
 self.size += 1  
 print("True")  
  
 # inserting at the end  
 def insertLast(self, num):  
 # full queue  
 if len(self.queue) == self.mSize:  
 print("False, the Queue is full")  
 else:  
 # insert and increase the size and r  
 self.queue.insert(self.r % self.mSize, num)  
 self.size += 1  
 self.r += 1  
 print("True")  
  
 def deleteFront(self):  
 # if there are items in the queue, delete the front one and decrement  
 if self.size > 0:  
 self.queue.\_\_delitem\_\_(self.f % self.mSize)  
 self.size -= 1  
 else:  
 # empty queue statement  
 print("False, the Queue is empty")  
  
 def deleteLast(self):  
 # if there are items in the queue, delete the last one and decrement  
 if self.size > 0:  
 self.queue.\_\_delitem\_\_(self.r - 1)  
 self.size -= 1  
 self.r -= 1  
 else:  
 # empty queue statement  
 print("False, the Queue is empty")  
  
 # get the front item  
 def getFront(self):  
 if len(self.queue) > 0:  
 print(self.queue[self.f])  
 else:  
 # no elements  
 print("-1")  
  
 # get the rear element  
 def getRear(self):  
 if len(self.queue) > 0:  
 print(self.queue[self.r - 1])  
 else:  
 # no elements  
 print("-1")  
  
 # return true or false for empty status  
 def isEmpty(self):  
 print(0 == len(self.queue))  
  
 # return true or false for full status  
 def isFull(self):  
 print(len(self.queue) == self.mSize)  
  
  
################################################# Question 2 #################################################  
# question two class  
# The linked list is derived from the Linked List in the Powerpoint  
class Q21:  
 # initialize queue  
 def \_\_init\_\_(self):  
 self.head = None  
 self.tail = None  
 self.size = 0  
  
 # node class  
 class \_Node:  
 def \_\_init\_\_(self, element, next):  
 self.selement = element  
 self.snext = next  
  
 # only need enqueue for this class,  
 def enqueue(self, num):  
 newnum = self.\_Node(num, None)  
 # if empty, the list size will be 0, and the node will be the head  
 if self.size == 0:  
 self.head = newnum  
 # The new element is put at the end of the list and the tail and size are updated  
 else:  
 self.tail.snext = newnum  
 self.tail = newnum  
 self.size += 1  
  
  
class Q22:  
 # initialize queue  
 def \_\_init\_\_(self):  
 self.head = None  
 self.tail = None  
 self.size = 0  
  
 # node class  
 class \_Node:  
 def \_\_init\_\_(self, element, next):  
 self.selement = element  
 self.snext = next  
  
 # only need enqueue for this class,  
 def enqueue(self, num):  
 newnum = self.\_Node(num, None)  
 # if empty, the list size will be 0, and the node will be the head  
 if self.size == 0:  
 self.head = newnum  
 # The new element is put at the end of the list and the tail and size are updated  
 else:  
 self.tail.snext = newnum  
 self.tail = newnum  
 self.size += 1  
  
  
class Q23:  
 # initialize queue  
 def \_\_init\_\_(self):  
 self.head = None  
 self.tail = None  
 self.size = 0  
  
 # node class  
 class \_Node:  
 def \_\_init\_\_(self, element, next):  
 self.selement = element  
 self.snext = next  
  
 # only need enqueue for this class,  
 def enqueue(self, num):  
 newnum = self.\_Node(num, None)  
 # if empty, the list size will be 0, and the node will be the head  
 if self.size == 0:  
 self.head = newnum  
 # The new element is put at the end of the list and the tail and size are updated  
 else:  
 self.tail.snext = newnum  
 self.tail = newnum  
 self.size += 1  
  
  
################################################# Question 3 #################################################  
# this class is derived from the linked queue given in the notes for our class. This class still has the limitation of the Q2 class, which is the elements in the linked lists will start to overlap  
class Question3:  
 # Node Class  
 class \_Node:  
 def \_\_init\_\_(self, element, next):  
 self.selement = element  
 self.snext = next  
  
 # Queue Setup and Operations  
 def \_\_init\_\_(self):  
 self.\_head = None  
 self.\_tail = None  
 # amount of queue elements  
 self.size = 0  
  
 # return the size of the Queue  
 def length(self):  
 print(self.size)  
  
 # checks if it is empty and returns true or false  
 def is\_empty(self):  
 return (self.size == 0)  
  
 def first(self):  
 if self.is\_empty():  
 print("Queue is Empty")  
 return  
 # returns element at the front of the queue  
 print(self.\_head.selement)  
 return  
  
 # returns and removes the first element in the queue  
 def dequeue(self):  
 # started with empty queue  
 if self.is\_empty():  
 print("Queue is Empty")  
 return  
 # save the answer and update the head and size  
 answer = self.\_head.selement  
 self.\_head = self.\_head.snext  
 self.size -= 1  
 # if the dequeue created an empty queue, update the tail to none  
 if self.is\_empty():  
 self.\_tail = None  
 # returns the answer  
 print(answer)  
  
 # add an element to the back of the queue  
 def enqueue(self, num):  
 newnum = self.\_Node(num, None)  
 if self.size == 0:  
 self.\_head = newnum  
 else:  
 self.\_tail.snext = newnum  
 self.\_tail = newnum  
 self.size += 1  
  
 # search through the list to see if the key is inside it  
 def search(self, key):  
 found = False  
 check = self.\_head  
 for x in range(self.size):  
 if (check.selement != key):  
 check = check.snext  
 elif (check.selement == key):  
 found = True  
 print(found)  
  
  
################################################# Question 1 #################################################  
# testing the code for Question 1  
print("############# Question One #############")  
test = MyCircularDeque(3)  
test.insertLast(1)  
test.insertLast(2)  
test.insertLast(3)  
test.insertFront(4)  
test.getRear()  
test.isFull()  
test.deleteLast()  
test.isFull()  
test.insertFront(4)  
test.isEmpty()  
test.getFront()  
test.deleteFront()  
test.getFront()  
test.deleteFront()  
test.getFront()  
test.deleteFront()  
test.getFront()  
  
################################################# Question 2 #################################################  
print()  
# Testing for Question 2  
print("############# Question Two #############")  
#initialize and create list 1  
l1 = Q21  
l1.\_\_init\_\_(l1)  
l1.enqueue(l1, 1)  
l1.enqueue(l1, 2)  
l1.enqueue(l1, 3)  
place1 = l1.head  
size1 = l1.size  
  
#initialize and create list 2  
l2 = Q22  
l2.\_\_init\_\_(l2)  
l2.enqueue(l2, 4)  
l2.enqueue(l2, 5)  
l2.enqueue(l2, 6)  
l2.enqueue(l2, 7)  
place2 = l2.head  
size2 = l2.size  
  
#print list 1 and list 2  
print("List 1:")  
for x in range(size1):  
 print(place1.selement, end=' ')  
 place1 = place1.snext  
print()  
  
print("List 2:")  
for x in range(l2.size):  
 print(place2.selement, end=' ')  
 place2 = place2.snext  
print()  
  
#save the heads so the lists can be walked through  
p1 = l1.head  
p2 = l2.head  
#create and initialize the new list the two lists will be sorted into  
merged = Q23  
merged.\_\_init\_\_(merged)  
#index variables to walk through each list and allow inequal lists  
index1 = 0  
index2 = 0  
#walk through the list the sum of the size of the two lists  
for x in range(size2 + size1):  
 # print("round", x, " p1 is:", p1.selement, " p2 is:", p2.selement) # Testing Portion  
 #if both lists have elements, compare them and add to merged  
 if (index1 < size1 & index2 < size2):  
 if (p1.selement <= p2.selement):  
 merged.enqueue(merged, p1.selement)  
 p1 = p1.snext  
 index1 += 1  
 else:  
 merged.enqueue(merged, p2.selement)  
 p2 = p2.snext  
 index2 += 1  
 #if list 1 is larger  
 elif (index1 < size1):  
 merged.enqueue(merged, p1.selement)  
 p1 = p1.snext  
 index1 += 1  
 #if list 2 is larger  
 elif (index2 < size2):  
 merged.enqueue(merged, p2.selement)  
 p2 = p2.snext  
 index2 += 1  
#print the merged list  
print("Merged List:")  
m = merged.head  
for x in range(size1 + size2):  
 print(m.selement, end=' ')  
 m = m.snext  
print()  
  
################################################# Question 3 #################################################  
print()  
print("############### Question 3 Testing ###############")  
Q3 = Question3()  
Q3.enqueue(5)  
Q3.enqueue(3)  
Q3.first() # prints 5  
Q3.length() # prints 2  
Q3.dequeue() # prints 5  
print(Q3.is\_empty()) # prints False  
Q3.enqueue(7)  
Q3.enqueue(9)  
Q3.search(7) # prints True  
Q3.search(0) # prints False  
Q3.enqueue(4)  
Q3.length() # prints 4  
Q3.dequeue() # prints 3