Lab 9

Exercise 1

```
class Node:
    def __init__(self,initdata):
        self.data = initdata
        self.next = None

    def getData(self):
        return self.data

    def getNext(self):
        return self.next

    def setData(self,newdata):
        self.data = newdata

    def setNext(self,newnext):
        self.next = newnext

class LStack:
    def __init__(self):
        self.head = None

    def push(self,item):
        temp = Node(item)
        temp.setNext(self.head)
        self.head = temp

    def size(self):
        current = self.head
        count = 0
        while current != None:
            count = count + 1
            current = current.getNext()
        return count

    def peek(self):
        return self.head.getData()

    def isEmpty(self):
        return self.head == None
```

```
    def pop(self):
        current = self.head
        current = current.getNext()
        self.head = current

linkedStack = LStack()
linkedStack.push(1)
linkedStack.push(6435)
linkedStack.push(35)
linkedStack.push(4)
linkedStack.push(3534)

print(linkedStack.size())
print(linkedStack.peek())
linkedStack.pop()
print(linkedStack.isEmpty())

print(linkedStack.size())
print(linkedStack.peek())
linkedStack.pop()
print(linkedStack.isEmpty())

print(linkedStack.size())
print(linkedStack.peek())
linkedStack.pop()
print(linkedStack.isEmpty())

print(linkedStack.size())
print(linkedStack.peek())
linkedStack.pop()
print(linkedStack.isEmpty())

print(linkedStack.size())
print(linkedStack.peek())
linkedStack.pop()
print(linkedStack.isEmpty())
```

I tested with the commands above. I put the same commands in for the regular Stack class, except the class was not called LStack, and was an actual stack. The outputs were the same for both:

5
3534
False
4
4
False
3
35
False
2
6435
False
1
1
True

Exercise 2

```python
class Node:
    def __init__(self,initdata):
        self.data = initdata
        self.next = None

    def getData(self):
        return self.data

    def getNext(self):
        return self.next

    def setData(self,newdata):
        self.data = newdata

    def setNext(self,newnext):
        self.next = newnext

class LQueue:
    def __init__(self):
        self.head = None

    def isEmpty(self):
        return self.head == None

    def size(self):
        current = self.head
        count = 0
        while current != None:
            count = count + 1
            current = current.getNext()
        return count

    def dequeue(self):
        current = self.head
        temp = current
        popped = temp.getData()
        current = current.getNext()
        self.head = current
        return popped

    def enqueue(self,item):
        current = self.head
        temp = Node(item)
        if current == None:
            temp = Node(item)
```

```
        temp.setNext(self.head)
        self.head = temp
    else:
        while current != None:
            previous = current
            current = current.getNext()
        previous.setNext(temp)


q = LQueue()
print(q.isEmpty())
q.enqueue(23)
q.enqueue(4)
q.enqueue(2435)
q.enqueue(1234)
print(q.isEmpty())
print(q.dequeue())
print(q.size())
print(q.dequeue())
print(q.isEmpty())
```

Similarly to exercise one, I tested the same commands with the queue class. The outputs were the same:

True
False
23
3
4
False

Exercise 3

```
class Node:
    def __init__(self,initdata):
        self.data = initdata
        self.next = None

    def getData(self):
        return self.data

    def getNext(self):
        return self.next

    def setData(self,newdata):
        self.data = newdata

    def setNext(self,newnext):
        self.next = newnext

class LDequeue:
    def __init__(self):
        self.head = None

    def isEmpty(self):
        return self.head == None

    def size(self):
        current = self.head
        count = 0
        while current != None:
            count = count + 1
            current = current.getNext()
        return count

    def addRear(self, item):
        current = self.head
        temp = Node(item)
        if current == None:
            temp = Node(item)
            temp.setNext(self.head)
            self.head = temp
        else:
            while current != None:
                previous = current
                current = current.getNext()
            previous.setNext(temp)
```

```python
    def addFront(self, item):
        temp = Node(item)
        temp.setNext(self.head)
        self.head = temp

    def removeFront(self):
        current = self.head
        temp = current
        popped = temp.getData()
        current = current.getNext()
        self.head = current
        return popped

    def removeRear(self):
        current = self.head
        previous = None
        while True:
            if current.getNext() == None:
                break
            else:
                previous = current
                current = current.getNext()

        removed = current.getData()

        if previous == None:
            self.head = None
        else:
            previous.setNext(current.getNext())

        return removed



q = LDequeue()

q.addFront(34)
q.addFront(4)
q.addRear(9)
print(q.removeFront())
q.addFront(23)
print(q.removeRear())
print(q.removeFront())
print(q.removeRear())
print(q.size())
print(q.isEmpty())
```

Once again, I tested with the same commands in the Dequeue:

4
9
23
34
0
True

Exercise 4

```
import time
import unorderedlist

print("The list will be 5 items long, with the items 1, 3, 2, 5, 4. These are the results for each.")

'''This is the initializing test.'''
#For regular list
print("\n")
bc = time.time()
list = [1, 3, 2, 5, 4]
ad = time.time()
ft = ad - bc
print ("The time it took to initialize your list was " + str(ft))

#For linked list
bc = time.time()
llist = unorderedlist.UnorderedList()
llist.add(4)
llist.add(5)
llist.add(2)
llist.add(3)
llist.add(1)
ad = time.time()
ft = ad - bc
print ("The time it took to initialize your linked list was " + str(ft))

'''This is the test for adding an item to the beginning of the list, since linked list cannot add to the end without a special function.'''

#Test for regular list.
print("\n")
bc = time.time()
i = 1
item = 6
previous = list[0]
list[0] = item
while i < len(list):
    current = list[i]
    list[i] = previous
    previous = current
    i += 1
if i <= len(list):
    list.append(previous)
ad = time.time()
ft = ad - bc
```

```python
print("The time it took to add an item to the begginning of the list, without losing any data was " + str(ft))

#Test for linked list
bc = time.time()
llist.add(item)
ad = time.time()
ft = ad - bc
print("The time it took to add an item to the begginning of the linked list, without losing any data was " + str(ft))

'''This is a test to see how long it takes to look for an item in the list.'''

value = 3
#Test for regular list
print("\n")
bc = time.time()
found = False
for i in range(len(list)):
    if list[i] == value:
        found = True
        break
print(found)
ad = time.time()
ft = ad - bc
print("The time it took to find this item in a list was " + str(ft))

#Test for linked list
print("\n")
bc = time.time()
print(llist.search(3))
ad = time.time()
ft = ad - bc
print("The time it took to find this item in a linked list was " + str(ft))

'''This is the test for removing an item, with the assumption that the item is there.'''
value = 3
print("\n")
#Test for list
bc = time.time()
i = 0
while True: #I did a while true statement, because in a linked list if the item is not there, it will search until it is out of range.
    if list[i] == value:
        list.pop(i)
        break
    else:
        i += 1
ad = time.time()
ft = ad + bc
```

```
print("The time it took to remove this item in a list was " + str(ft))

#Test for a linked list
bc = time.time()
llist.remove(3)
ad = time.time()
ft = ad - bc
print("The time it took to remove this item in a linked list was " + str(ft))
```

The outputs for this experiment were rather interesting to me. I tried to make the test as fair as possible, and make the two data types' functions as similar to one another without changing the actual data type itself. Here are the outputs:

The list will be 5 items long, with the items 1, 3, 2, 5, 4. These are the results for each.

The time it took to initialize your list was 9.5367431640625e-07
The time it took to initialize your linked list was 5.245208740234375e-06

The time it took to add an item to the begginning of the list, without losing any data was 3.814697265625e-06
The time it took to add an item to the begginning of the linked list, without losing any data was 0.0

True
The time it took to find this item in a list was 1.0967254638671875e-05

True
The time it took to find this item in a linked list was 1.9073486328125e-06

The time it took to remove this item in a list was 3296080590.9537134
The time it took to remove this item in a linked list was 9.5367431640625e-07

In almost every situation, the link list did far better than a regular list did. The only time it did not do better, the difference was marginal at best.