CS 2302 Data Structures

Fall 2019

Lab Report #3

Due: October 4th, 2019

Professor: Olac Fuentes

TA: Anindita Nath

Introduction

For this lab we were tasked with creating a linked list that is always sorted. We then were tasked with creating several methods to manipulate the list such as adding an element to the list or checking for any duplicate numbers in the list.

Proposed Solution Design and Implementation

Creating the Sorted List:

I started the program by creating a Node object that contains the data of the node as well as the position of the next node. After that I created a new class called sorted list that would hold all the methods as well as an initiate method that starts the list.

1. **Print(self)**
   For this method I would set a temporary variable "T" to be the head of the linked list. Then was long as T was not None the method would use a while loop to print every element of the list until T equaled None.

2. **Insert(self, i)**
   For this method I first started by setting variable "H" as the head of the linked list, then created a node with the integer i. Next the program would check if the list was empty. If it was the node created would be set to the head and tail of the list. If it was not empty the program would check if the new node element was smaller then the head and if so, would place the new node as the head of the list. Otherwise it would just use a while loop to find where it is less then the next value of the list and insert is making sure to set the previous nodes next to itself and its node to the original next.

3. **Delete(self, i)**
   For this method I start out by checking if the list is empty because if it is there is no need to do anything. After checking if the list is not empty the method then use a while loop to move through the list to the i^{th} position and remove that node. If there is no i^{th} position the program just returns.

4. **Merge(self, M)**
   For this method I First checked if M.head was None in case we were trying to merge with an empty list. After that I checked if self.head was None and if so would just set the head and tail of M to be the head and tail of the list. If neither of those cases run then the

program uses a while loop to sort through M and using the insert method it would add each element in its appropriate spot in the list.

5. **IndexOf(self, i)**
   For this method I checked if the value of I was 0 and if so, would return the data of the head. If not I would use a while loop to go through the list to find and return the elemnt at index i. I would return -1 if there is no element i.

6. **Clear(self)**
   For this method I reset the head.next of the list to none. Then I set the head and tail of the list to none ensuring that the list was completely reset

7. **Min(self)**
   For this method since the list was already sorted, I just returned the value of the head.

8. **Max(self)**
   For this method since the list was sorted, I just returned the value of the tail.

9. **HasDuplicates(self)**
   For this method I first checked if the head of the list was empty and if so, returned false as there are technically no duplicates in the list. Then I created a list with the data of the head as well as a node value with the index of head.next. After that I would use a while loop and for loop to iterate through the list. While going through the list it would check if the temp nodes data is within the list of values already stored and if there is a matching value it would return true. Otherwise it would end after it finished iterating through the list.

10. **Select(self, k)**
    For this method I first created two base cases, to see if the value of k was negative or if it was 0 and if so, would return the value of the head node. Otherwise It use a while loop it go through the list to return the k^{th} smallest element as well as returning math.inf if k was larger then the length of the list.

Experimental Results

L1.AppendList([9,8,7,6,5,4,3,2,1])

L1.Print()

L1.Insert(0)

L1.Delete(0)

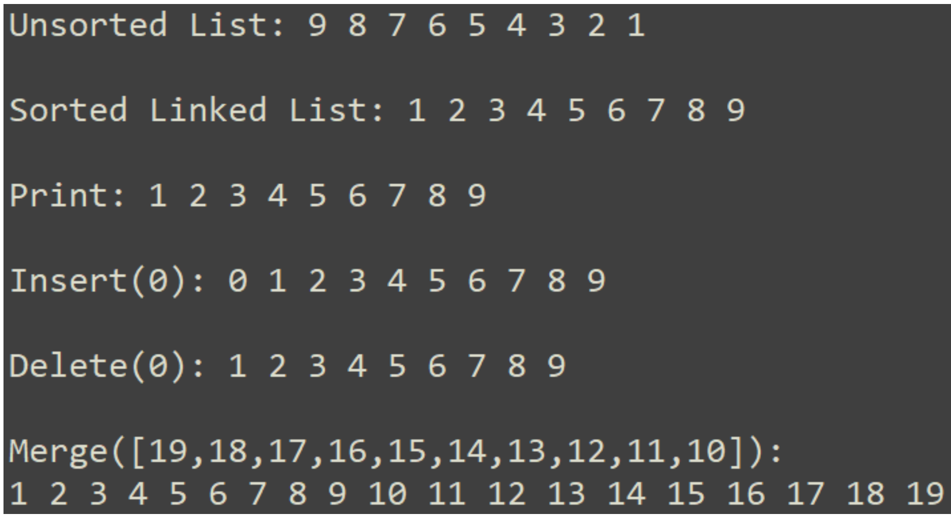
L2 = SortedList()

L3 = SortedList()

L2.AppendList([9,8,7,6,5,4,3,2,1])

L3.AppendList([19,18,17,16,15,14,13,12,11,10])

L2.Merge(L3)



```
Unsorted List: 9 8 7 6 5 4 3 2 1

Sorted Linked List: 1 2 3 4 5 6 7 8 9

Print: 1 2 3 4 5 6 7 8 9

Insert(0): 0 1 2 3 4 5 6 7 8 9

Delete(0): 1 2 3 4 5 6 7 8 9

Merge([19,18,17,16,15,14,13,12,11,10]):
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
```

L1.IndexOf(0)

L1.Clear()

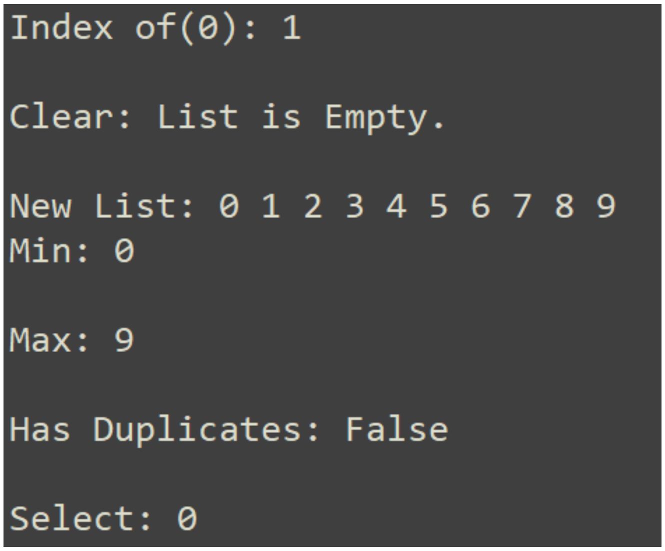
L1.Append([9,8,7,6,5,4,3,2,1,0])

L1.Min()

L1.Max()

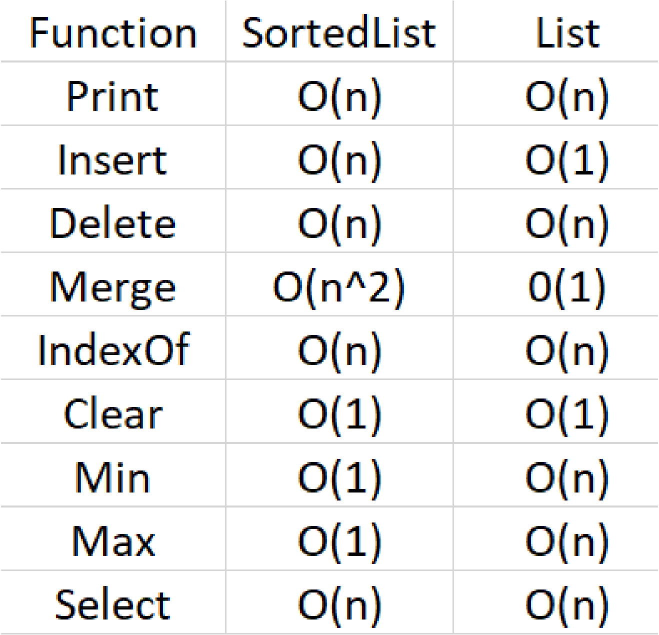
L1.HasDuplicates()

L1.Select(0)



```
Index of(0): 1

Clear: List is Empty.

New List: 0 1 2 3 4 5 6 7 8 9
Min: 0

Max: 9

Has Duplicates: False

Select: 0
```

Data



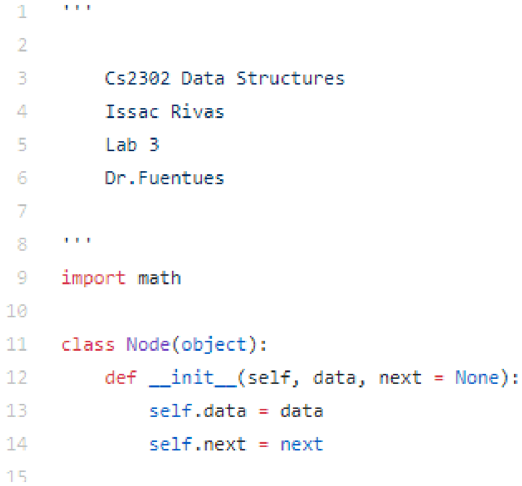
Function	SortedList	List
Print	O(n)	O(n)
Insert	O(n)	O(1)
Delete	O(n)	O(n)
Merge	O(n^2)	0(1)
IndexOf	O(n)	O(n)
Clear	O(1)	O(1)
Min	O(1)	O(n)
Max	O(1)	O(n)
Select	O(n)	O(n)

The data helps show how some tasks end up taking longer then a normal list suck as in the case of merge sort(O(n^2)) were each item needs to be place in ascending order. While other tasks like Min and Max can be solved with a run time of O(1) because the list is already sorted.

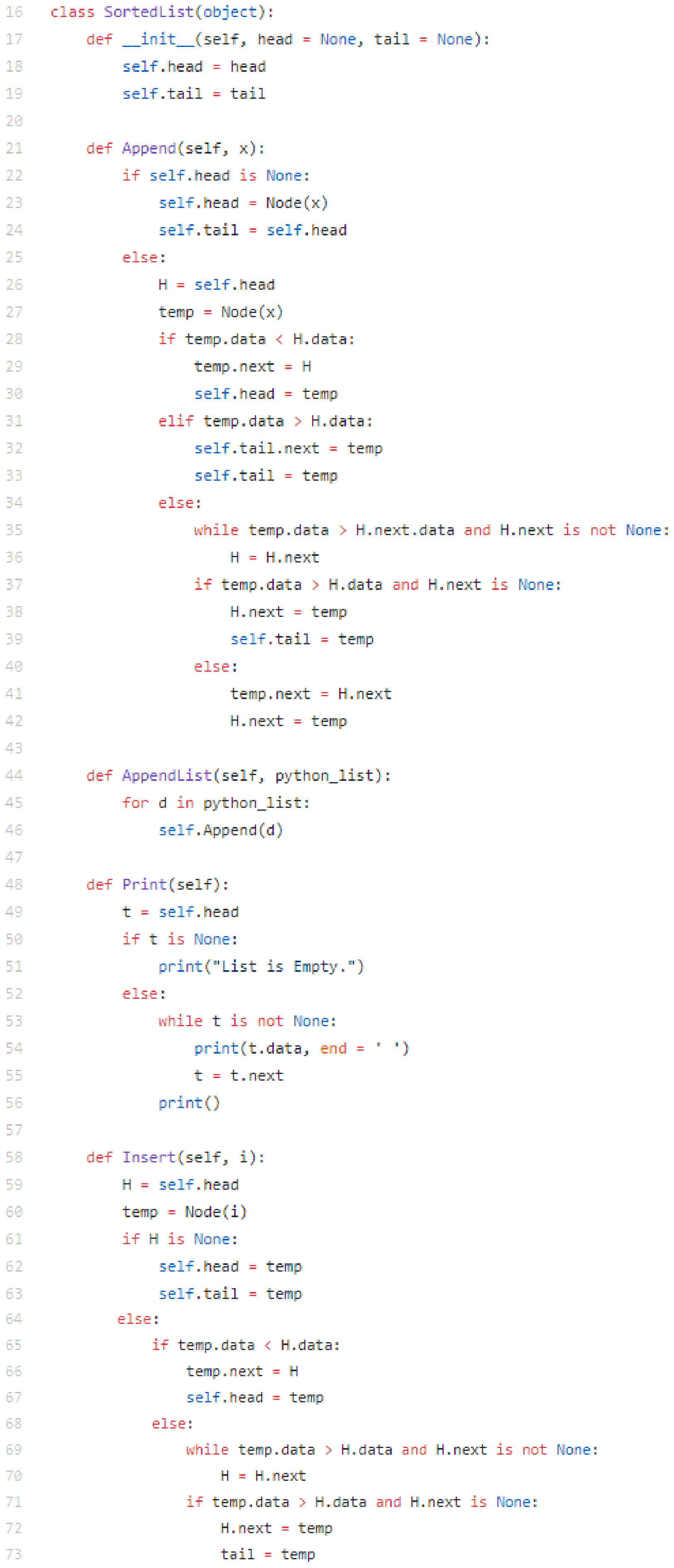
Conclusion

In conclusion through completing this lab I was able to see how a sorted linked list was able to have better run times in certain tasks then an unsorted linked list as well as how I can create methods that manipulate these linked lists such as if I wanted to create one for descending order.

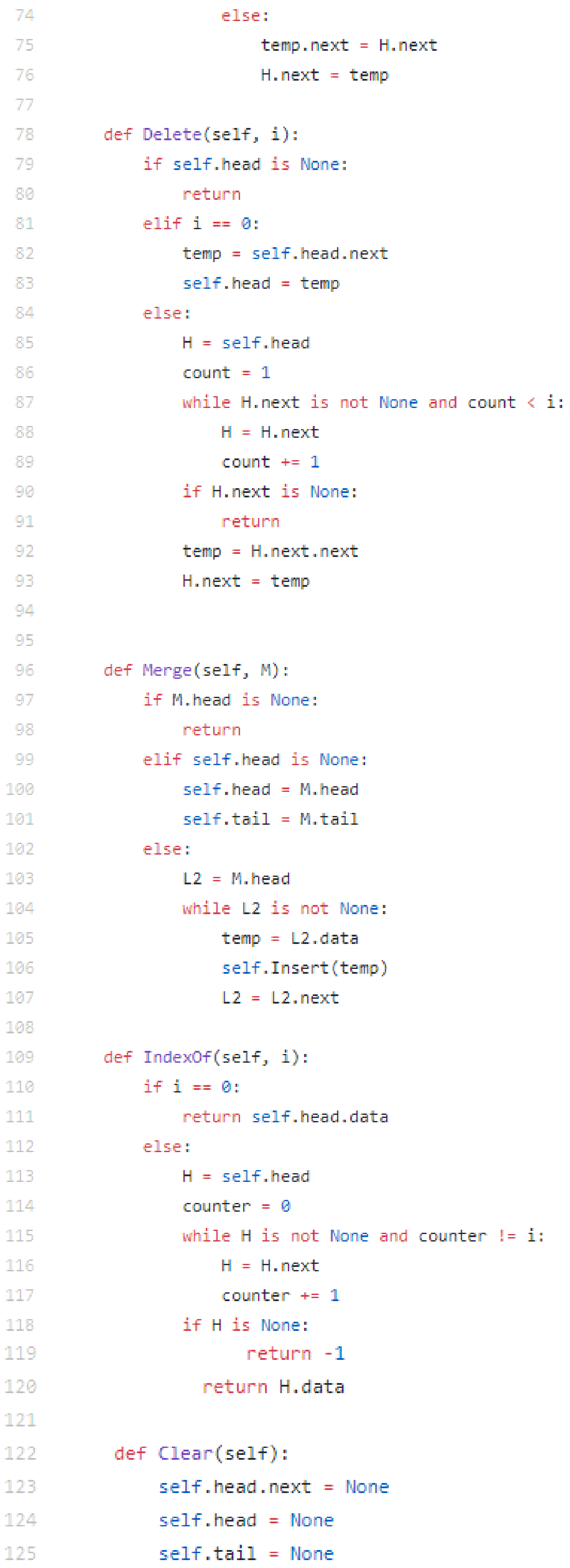
Appendix



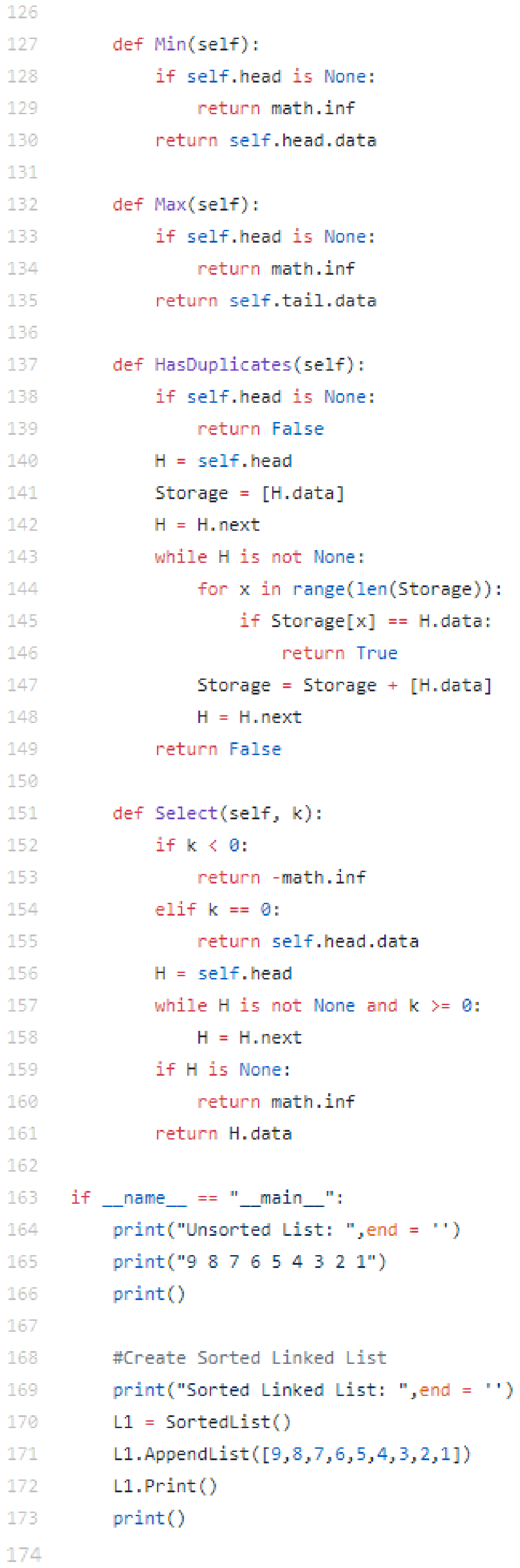
```
'''

    Cs2302 Data Structures
    Issac Rivas
    Lab 3
    Dr.Fuentues

'''
import math

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

```



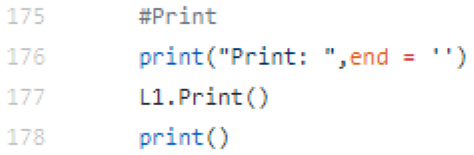
```python
class SortedList(object):
    def __init__(self, head = None, tail = None):
        self.head = head
        self.tail = tail

    def Append(self, x):
        if self.head is None:
            self.head = Node(x)
            self.tail = self.head
        else:
            H = self.head
            temp = Node(x)
            if temp.data < H.data:
                temp.next = H
                self.head = temp
            elif temp.data > H.data:
                self.tail.next = temp
                self.tail = temp
            else:
                while temp.data > H.next.data and H.next is not None:
                    H = H.next
                if temp.data > H.data and H.next is None:
                    H.next = temp
                    self.tail = temp
                else:
                    temp.next = H.next
                    H.next = temp

    def AppendList(self, python_list):
        for d in python_list:
            self.Append(d)

    def Print(self):
        t = self.head
        if t is None:
            print("List is Empty.")
        else:
            while t is not None:
                print(t.data, end = ' ')
                t = t.next
            print()

    def Insert(self, i):
        H = self.head
        temp = Node(i)
        if H is None:
            self.head = temp
            self.tail = temp
        else:
            if temp.data < H.data:
                temp.next = H
                self.head = temp
            else:
                while temp.data > H.data and H.next is not None:
                    H = H.next
                if temp.data > H.data and H.next is None:
                    H.next = temp
                    tail = temp
```



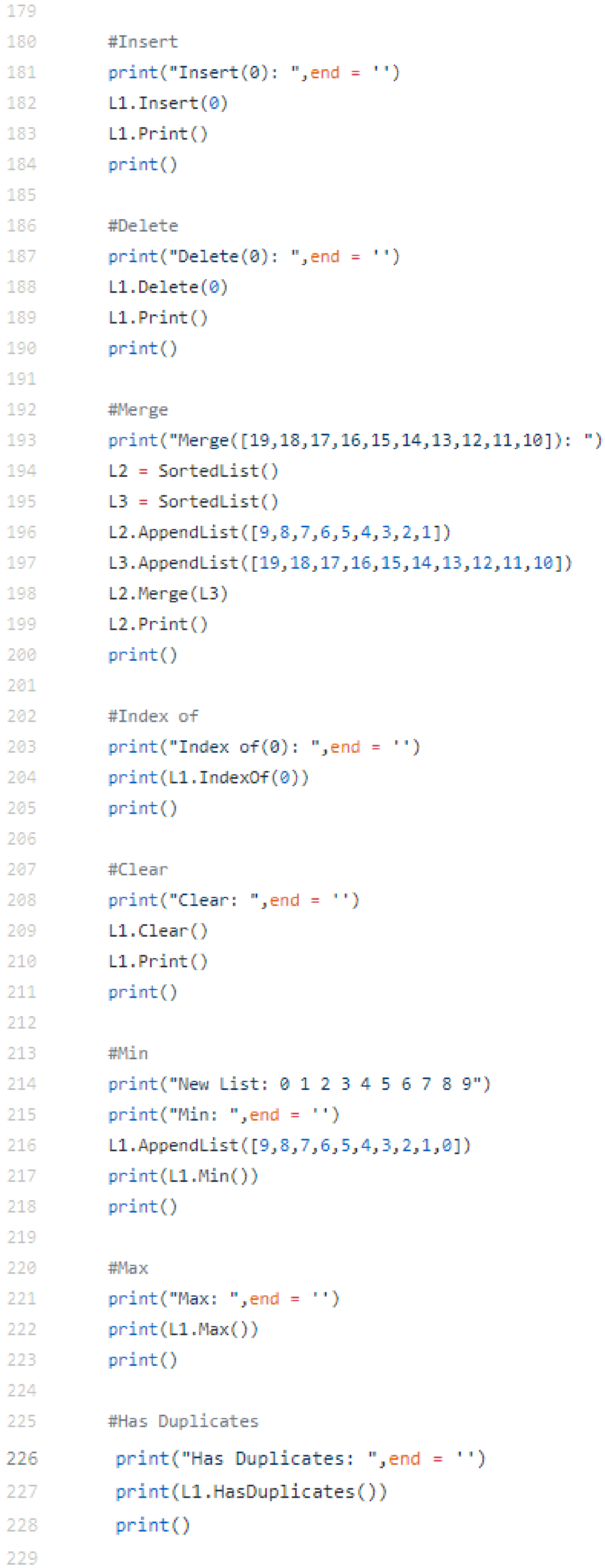
```python
                else:
                    temp.next = H.next
                    H.next = temp

    def Delete(self, i):
        if self.head is None:
            return
        elif i == 0:
            temp = self.head.next
            self.head = temp
        else:
            H = self.head
            count = 1
            while H.next is not None and count < i:
                H = H.next
                count += 1
            if H.next is None:
                return
            temp = H.next.next
            H.next = temp


    def Merge(self, M):
        if M.head is None:
            return
        elif self.head is None:
            self.head = M.head
            self.tail = M.tail
        else:
            L2 = M.head
            while L2 is not None:
                temp = L2.data
                self.Insert(temp)
                L2 = L2.next

    def IndexOf(self, i):
        if i == 0:
            return self.head.data
        else:
            H = self.head
            counter = 0
            while H is not None and counter != i:
                H = H.next
                counter += 1
            if H is None:
                return -1
            return H.data

    def Clear(self):
        self.head.next = None
        self.head = None
        self.tail = None
```



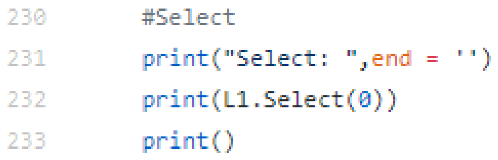
```python

    def Min(self):
        if self.head is None:
            return math.inf
        return self.head.data

    def Max(self):
        if self.head is None:
            return math.inf
        return self.tail.data

    def HasDuplicates(self):
        if self.head is None:
            return False
        H = self.head
        Storage = [H.data]
        H = H.next
        while H is not None:
            for x in range(len(Storage)):
                if Storage[x] == H.data:
                    return True
            Storage = Storage + [H.data]
            H = H.next
        return False

    def Select(self, k):
        if k < 0:
            return -math.inf
        elif k == 0:
            return self.head.data
        H = self.head
        while H is not None and k >= 0:
            H = H.next
        if H is None:
            return math.inf
        return H.data

if __name__ == "__main__":
    print("Unsorted List: ",end = '')
    print("9 8 7 6 5 4 3 2 1")
    print()

    #Create Sorted Linked List
    print("Sorted Linked List: ",end = '')
    L1 = SortedList()
    L1.AppendList([9,8,7,6,5,4,3,2,1])
    L1.Print()
    print()

```



```python
    #Print
    print("Print: ",end = '')
    L1.Print()
    print()
```



```
    #Insert
    print("Insert(0): ",end = '')
    L1.Insert(0)
    L1.Print()
    print()

    #Delete
    print("Delete(0): ",end = '')
    L1.Delete(0)
    L1.Print()
    print()

    #Merge
    print("Merge([19,18,17,16,15,14,13,12,11,10]): ")
    L2 = SortedList()
    L3 = SortedList()
    L2.AppendList([9,8,7,6,5,4,3,2,1])
    L3.AppendList([19,18,17,16,15,14,13,12,11,10])
    L2.Merge(L3)
    L2.Print()
    print()

    #Index of
    print("Index of(0): ",end = '')
    print(L1.IndexOf(0))
    print()

    #Clear
    print("Clear: ",end = '')
    L1.Clear()
    L1.Print()
    print()

    #Min
    print("New List: 0 1 2 3 4 5 6 7 8 9")
    print("Min: ",end = '')
    L1.AppendList([9,8,7,6,5,4,3,2,1,0])
    print(L1.Min())
    print()

    #Max
    print("Max: ",end = '')
    print(L1.Max())
    print()

    #Has Duplicates
    print("Has Duplicates: ",end = '')
    print(L1.HasDuplicates())
    print()
```



```
#Select
print("Select: ",end = '')
print(L1.Select(0))
print()
```

I Issac Rivas, certify that this project is entirely my own work. I wrote, debugged, and tested the code being presented, preformed the experiments, and wrote the report. I also certify that I did not share my code or report provided inappropriate assistance to any student in the class.