CS2302 Data Structures

Lab Report No. 3

Author: Bryan Ramos
Due: October 4th, 2019
Professor: Olac Fuentes
TA: Anindita Nath

Introduction

The purpose of this lab was to provide additional practice using linked lists which we have seen during class lectures and the past quizzes, the only difference is that in this case the element of this list will always be sorted in ascending over at all times and contain a series of functions that allow different operations to be performed on the list. In addition, there was a chance for additional practice with algorithm time complexity where we had to determine the time complexity for the functions in the Sorted *List* class and compare those runtimes to the regular *List* class we saw during the course.

Proposed Solution Design & Implementation

Before tackling the design of the assigned functions, first I imported common code from the List class (i.e. constructor) to create a linked list data structure. I kept the name of my Sorted List data structure, as *List*, the same name as the one from our deliberate practice exercise. The instructions required that the list elements should always be sorted in ascending order, so I brainstormed several methods to make sure that condition was met. My method which I called *Append* has several checks to determine where the element belongs in conjunction with the existing elements so that everything stays in ascending order and inserts the integer in a node where it belongs.

I created a List and populated it with random integer values that I could manipulate with each of my operations and use the output in the console to debug and test. After that, I created a menu that shows each of the assigned functions as a menu option and implemented a switch using if-else-if conditions to execute each of the assigned operations separately in the console.

With that out of the way, I began working the assigned functions in the order they were assigned:

1. **Print:** This function needed to print the contents of the list. The print function accepts the list as a parameter and I used a loop to traverse the loop starting from the head all the way to the last element (tail), printing the data at each node.
2. **Insert:** Elements need to be inserted in ascending order. The insert function requires two parameters, the list and the integer to be added to the list. Because I had already designed the method Append when creating my lists for debugging and testing, I went ahead and called that function within this one.
3. **Delete:** The instructions state that this function should remove integer *i* from the list and if the integer *i* is not in the list it should do nothing. The way I approached writing this function was by using a try and except block. First, I have two variables, *previous* which is initially null and used to keep track of the previous element found before current. Since there is no previous element before the head it

is initially null, and *t* which is the current node and initially is the head of the linked list. Next, my try-except block which traverses the list and if the element is not found returns None, the function stops there, thereby fulfilling the requirements. If it is found that element is removed, that is, that node's reference is removed from the list.

> For example, say that we are traversing the list and the current node the traversal is at is somewhere in the middle of the list and it happens to be the node storing the data we want to remove, then we remove the node by pointing the previous element's (stored in *previous*) reference to the element after the current, if there is an element after current.

4. **Merge:** This function should insert all the elements from a second list, say list M, into another list, say L, both of which should be of the same type, Sorted *List*. This method was simple to complete. First, the method accepts two parameters, the original list and the list to merge the elements from (M). Next, starting from the head of list M, I traversed list M, adding each element to the original list by utilizing the *Append* method again that I created earlier.
5. **IndexOf:** This function should return the index of *i* in the list, or -1 if *i* is not found in the list. The head is found at index 0, for example. Here, I created a variable called *index* to keep track of the index during the traversal of the list. While traversing, if the data at the current node matches the integer whose index we are looking for, the index is returned. Otherwise, the index is incremented by one and the traversal goes to the next node. If the element is not found, that is, the list was traversed completely, -1 is returned. While writing this report, I also thought of other ways this method could be implemented since this lab assignment deals with sorted lists.

   > For example, say we have a list [0, 1, 4, 5, 6, 7]. In an alternate solution for the method, say we are looking for the index of 3. When searching for the index of this element, while traversing to the list we will reach 1, then the next current node in the traversal would be 4, but since we know the list is sorted, if 4 is reached after 1, it's evident the element is not in the list so -1 should be returned at that moment.

6. **Clear:** With this function, I took a simple approach and based my solution on what I've learned in the past when implementing lists in other courses. I set the head reference to null. I know that in Java I can set the head to null and Java's built-in garbage collector will deal with the removed refences. When I have some time, I want to do some research to see if Python has a built-in garbage collector.
7. **Min:** When I wrote the solution for this method, I initially set the min value to the max integer value in Python and traversed the list setting the min value to anything

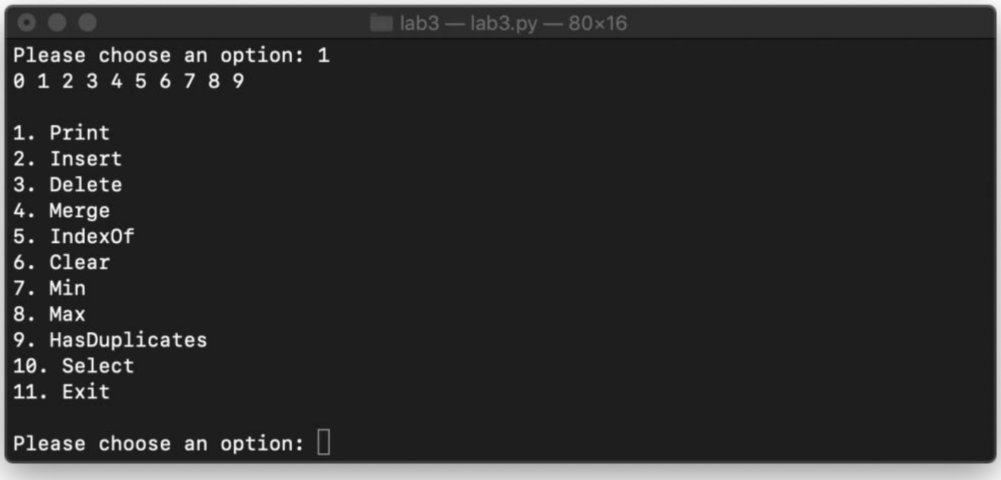
smaller than the value currently stored in min, but then I realized, we are dealing with a sorted list, therefore, the head of the list will be the smallest element in the list, always. When writing my report, I realized I did not have to traverse the list to solve this method and it would have had a O(1) runtime.

8. **Max:** Similar situation in my case as *Min*. My solution initially set max to the min integer value in Python and traversed the list setting the max value to anything larger than the value currently stored in min, and when writing the report again realized traversing the list was not necessary, it could have been done in constant time by returning the tail of the list, because it is the last element in a sorted list, it is the largest element in the list.
9. **HasDuplicates**: The instructions stated that this method should return a Boolean - if the list has duplicate elements, true should be returned, otherwise false. First, I considered the edge case of the list being empty, if so, an empty list does not have duplicate elements. Next, I traversed the list. Because the list is always sorted, in order to see if there are duplicate elements, the element in the list after the current one should be different, not the same. If the next element in the list is the same as the current element in the list, then there are duplicates and true will be returned. Otherwise, false will be returned.
10. **Select**: Finally, I had to tackle the last method assigned, which was like what was done with lab 2 where the kth smallest element in the list is returned or math.inf if the list has less than k-1 elements. The structure of this method is very similar to that of the *IndexOf* method. First for my edge case, if the list is empty, math.inf is returned. Otherwise, I traverse the list, if the index matches the value of k, the node data at the current node is returned, otherwise the index is incremented, and the traversal moves to the next node until the element is found as long as k is in bounds. If k is not in bounds, that is, k is greater than the last index of the list, math.inf or None is returned.

Experimental Results

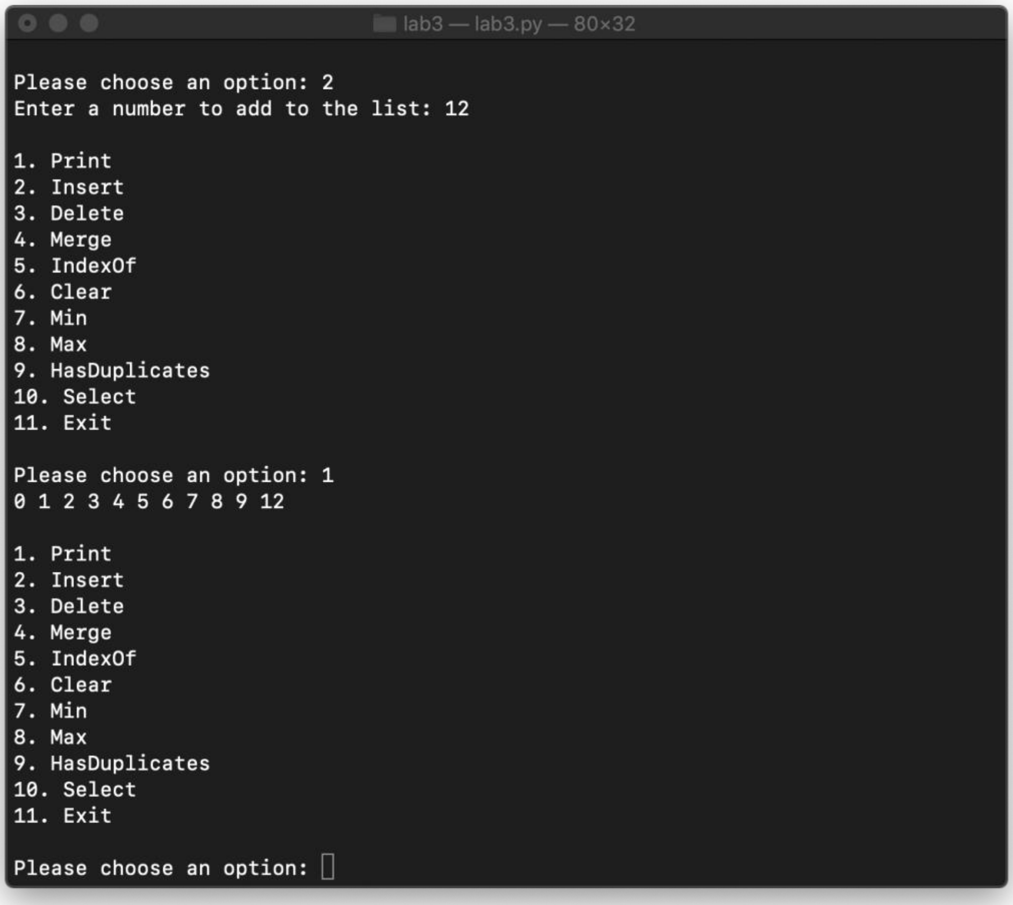
Here is the output in the console for a list called L containing values 0-9 and the list to merge M that contain random integer values from 0-12 and each operation is shown.

Print:



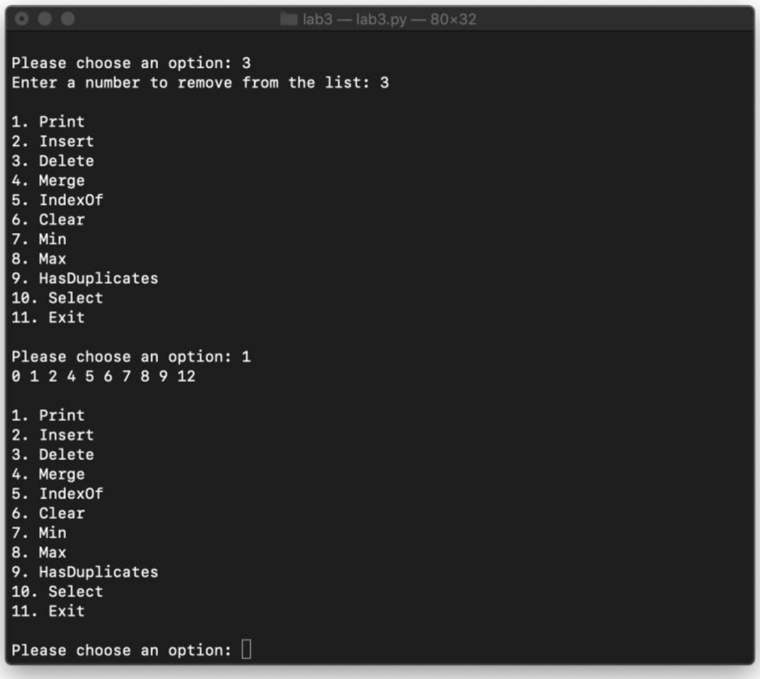
```
Please choose an option: 1
0 1 2 3 4 5 6 7 8 9

1. Print
2. Insert
3. Delete
4. Merge
5. IndexOf
6. Clear
7. Min
8. Max
9. HasDuplicates
10. Select
11. Exit

Please choose an option: 
```

Insert 12:



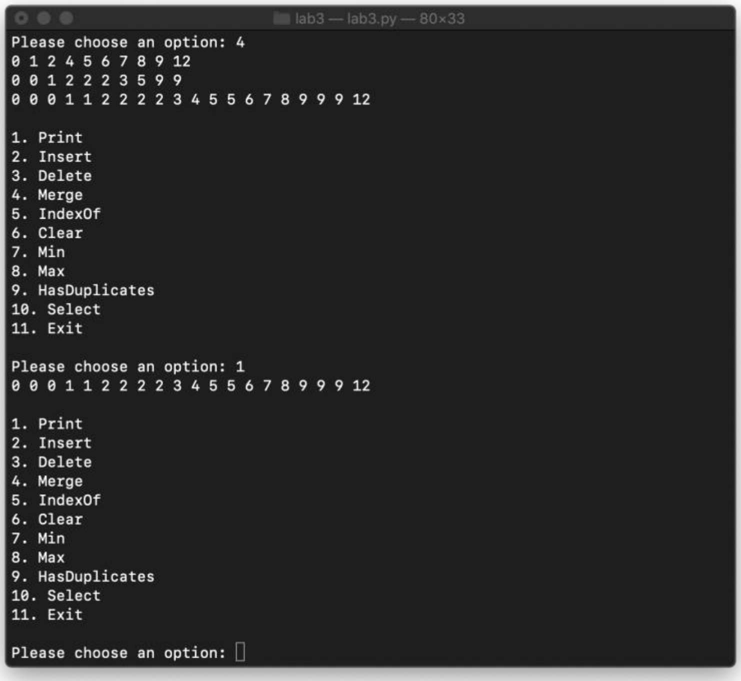
```

Please choose an option: 2
Enter a number to add to the list: 12

1. Print
2. Insert
3. Delete
4. Merge
5. IndexOf
6. Clear
7. Min
8. Max
9. HasDuplicates
10. Select
11. Exit

Please choose an option: 1
0 1 2 3 4 5 6 7 8 9 12

1. Print
2. Insert
3. Delete
4. Merge
5. IndexOf
6. Clear
7. Min
8. Max
9. HasDuplicates
10. Select
11. Exit

Please choose an option: 
```

Delete 3:



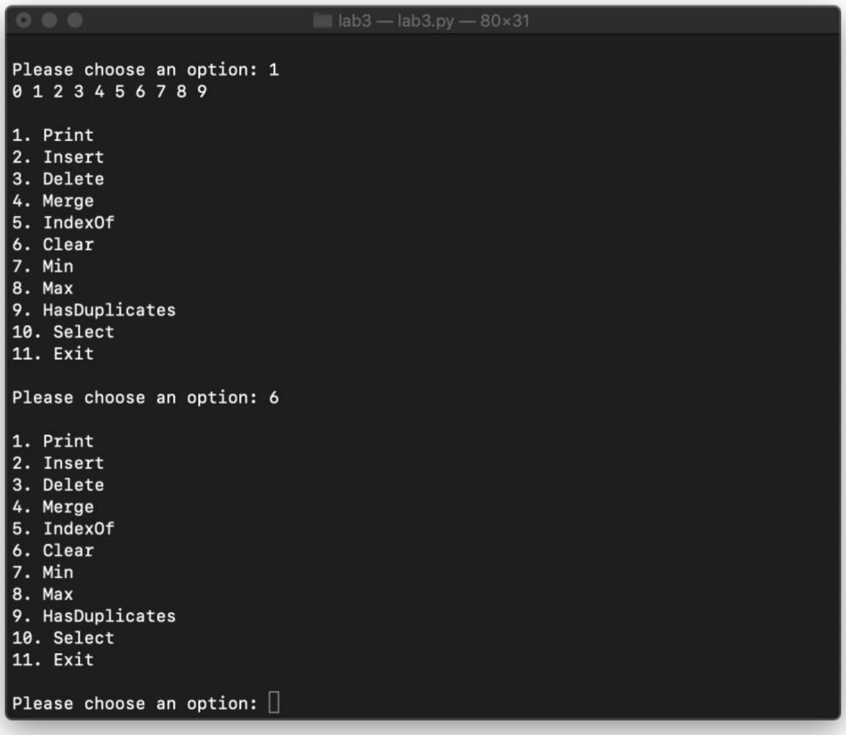
```
lab3 — lab3.py — 80×32

Please choose an option: 3
Enter a number to remove from the list: 3

1. Print
2. Insert
3. Delete
4. Merge
5. IndexOf
6. Clear
7. Min
8. Max
9. HasDuplicates
10. Select
11. Exit

Please choose an option: 1
0 1 2 4 5 6 7 8 9 12

1. Print
2. Insert
3. Delete
4. Merge
5. IndexOf
6. Clear
7. Min
8. Max
9. HasDuplicates
10. Select
11. Exit

Please choose an option: 
```

Merge M into L:



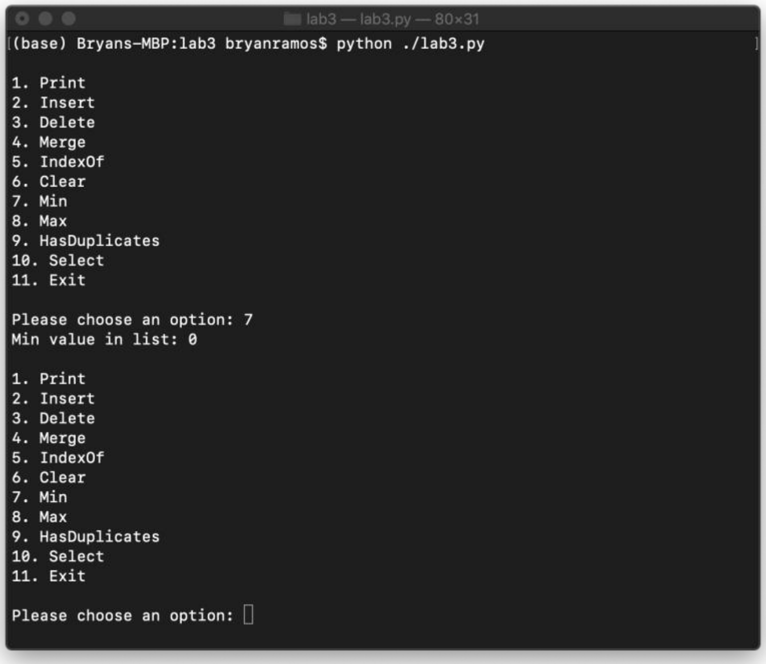
```
lab3 — lab3.py — 80×33
Please choose an option: 4
0 1 2 4 5 6 7 8 9 12
0 0 1 2 2 2 3 5 9 9
0 0 0 1 1 2 2 2 2 3 4 5 5 6 7 8 9 9 9 12

1. Print
2. Insert
3. Delete
4. Merge
5. IndexOf
6. Clear
7. Min
8. Max
9. HasDuplicates
10. Select
11. Exit

Please choose an option: 1
0 0 0 1 1 2 2 2 2 3 4 5 5 6 7 8 9 9 9 12

1. Print
2. Insert
3. Delete
4. Merge
5. IndexOf
6. Clear
7. Min
8. Max
9. HasDuplicates
10. Select
11. Exit

Please choose an option: 
```

Clear (empty output since list is cleared/deleted when I try to print the list after clearing)



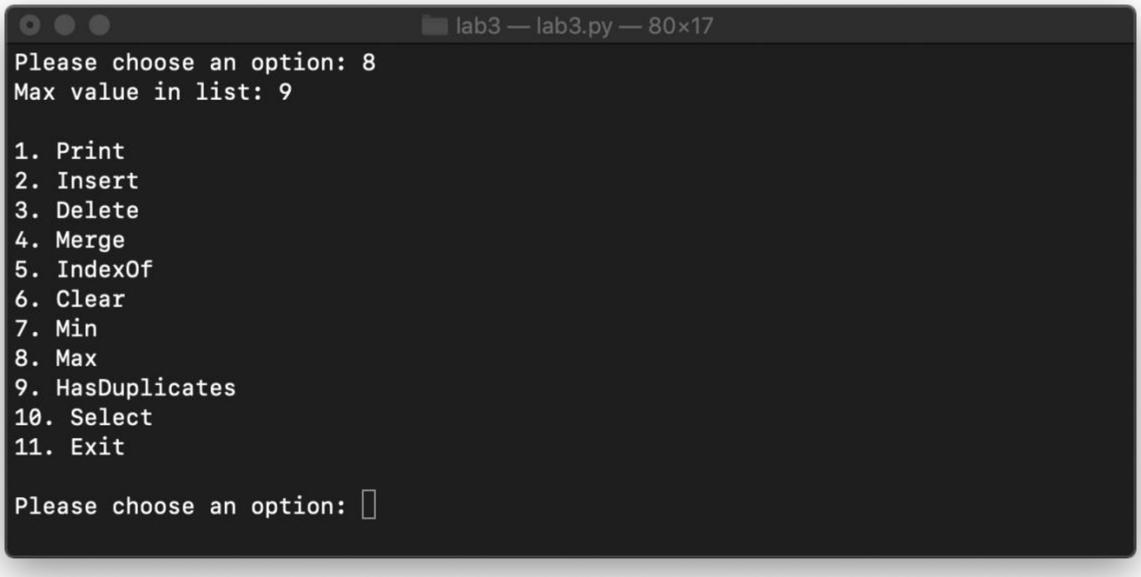
```
Please choose an option: 1
0 1 2 3 4 5 6 7 8 9

1. Print
2. Insert
3. Delete
4. Merge
5. IndexOf
6. Clear
7. Min
8. Max
9. HasDuplicates
10. Select
11. Exit

Please choose an option: 6

1. Print
2. Insert
3. Delete
4. Merge
5. IndexOf
6. Clear
7. Min
8. Max
9. HasDuplicates
10. Select
11. Exit

Please choose an option: 
```

Min value - should return first value in list - being 0, again, I later noticed traversal was not necessary since the smallest element will be at the front of the list, the head of list



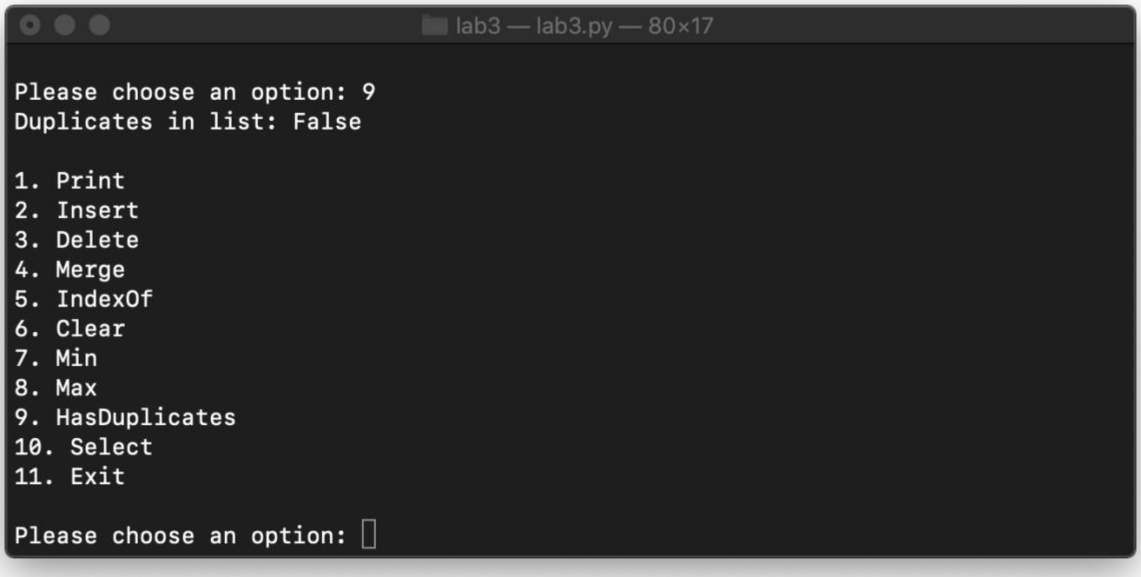
```
(base) Bryans-MBP:lab3 bryanramos$ python ./lab3.py

1. Print
2. Insert
3. Delete
4. Merge
5. IndexOf
6. Clear
7. Min
8. Max
9. HasDuplicates
10. Select
11. Exit

Please choose an option: 7
Min value in list: 0

1. Print
2. Insert
3. Delete
4. Merge
5. IndexOf
6. Clear
7. Min
8. Max
9. HasDuplicates
10. Select
11. Exit

Please choose an option: 
```

Max value - should return last value in list - being 9, again, I later noticed traversal was not necessary since the largest element will be the last element of the list, the tail of list



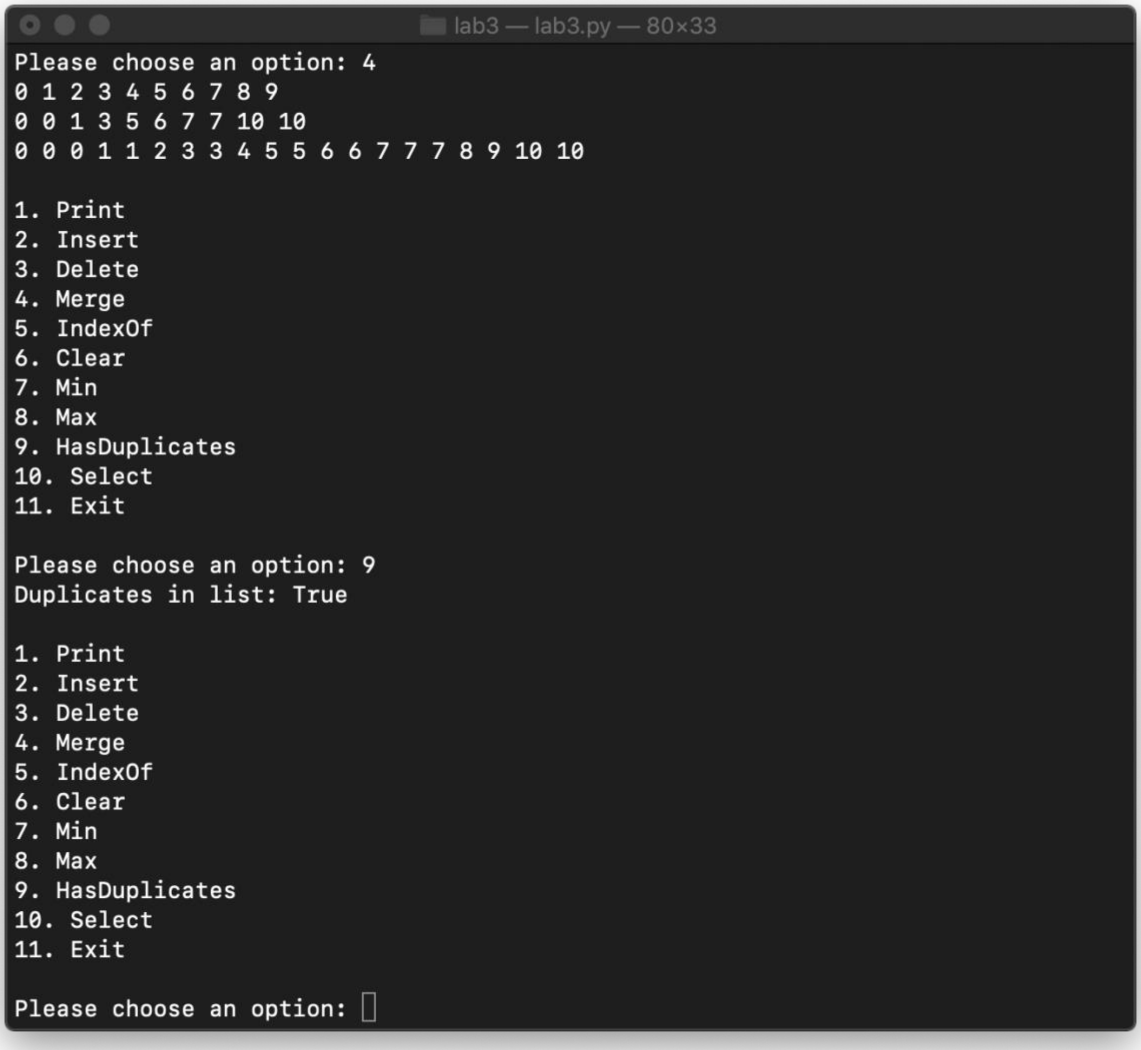
```
Please choose an option: 8
Max value in list: 9

1. Print
2. Insert
3. Delete
4. Merge
5. IndexOf
6. Clear
7. Min
8. Max
9. HasDuplicates
10. Select
11. Exit

Please choose an option: 
```

HasDuplicates, tested with original, should return false



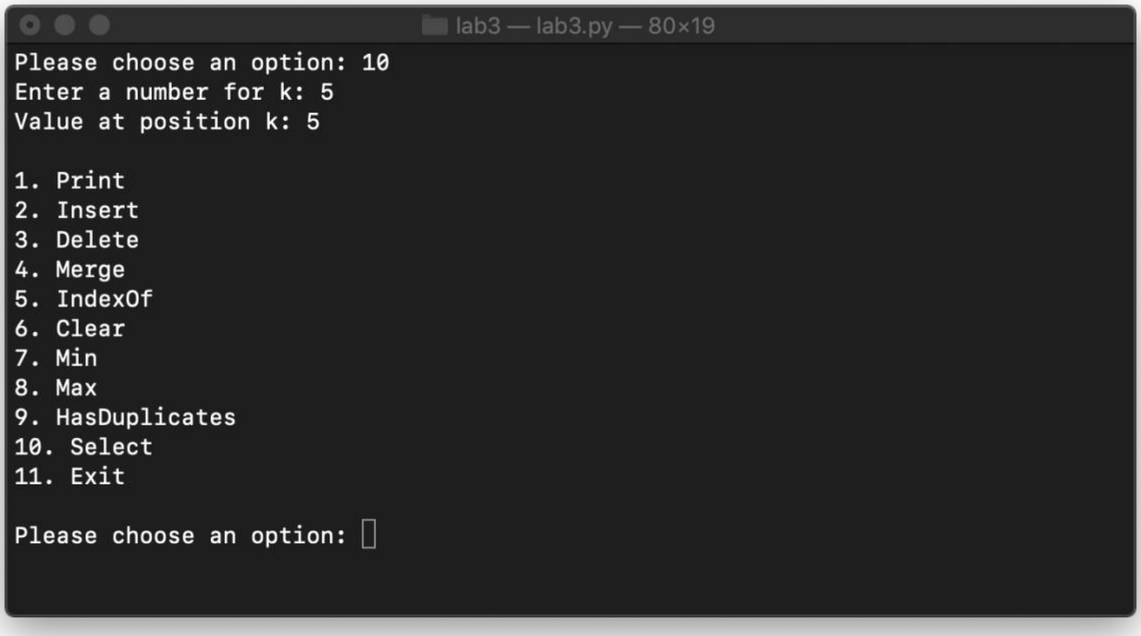
```

Please choose an option: 9
Duplicates in list: False

1. Print
2. Insert
3. Delete
4. Merge
5. IndexOf
6. Clear
7. Min
8. Max
9. HasDuplicates
10. Select
11. Exit

Please choose an option: 
```

HasDuplicates: merged M into L and then tested again, should return true for duplicates



```
Please choose an option: 4
0 1 2 3 4 5 6 7 8 9
0 0 1 3 5 6 7 7 10 10
0 0 0 1 1 2 3 3 4 5 5 6 6 7 7 7 8 9 10 10

1. Print
2. Insert
3. Delete
4. Merge
5. IndexOf
6. Clear
7. Min
8. Max
9. HasDuplicates
10. Select
11. Exit

Please choose an option: 9
Duplicates in list: True

1. Print
2. Insert
3. Delete
4. Merge
5. IndexOf
6. Clear
7. Min
8. Max
9. HasDuplicates
10. Select
11. Exit

Please choose an option: 
```

Select: After completing my first iteration of my report, I realized that in the source code already on GitHub I neglected to increment the index so the index will always be zero, so for any k value greater than 0, None will be returned, I went ahead and fixed that in the source code I included below.



```
Please choose an option: 10
Enter a number for k: 5
Value at position k: 5

1. Print
2. Insert
3. Delete
4. Merge
5. IndexOf
6. Clear
7. Min
8. Max
9. HasDuplicates
10. Select
11. Exit

Please choose an option: 
```

Function	Sorted *List*	*List*
Print()	O(n)	O(n)
Insert(i)	O(n)	O(1)
Delete(i)	O(n)	O(n)
Merge(M)	O(n)	O(n)
IndexOf(i)	O(n)	O(n)
Clear(i)	O(1)	O(1)
Min()	O(n) should be O(1)	O(n)
Max()	O(n) should be O(1)	O(n)
HasDuplicates()	O(n)	O(n)
Select(k)	O(n)	O(n)

Since this requires that the list always be sorted, some methods in the sorted list class will have longer time complexities, take the insert method, whereas in a regular list insert is constant, in the sorted list class it is constant. Also, because I didn't realize that for the min and max functions list traversal was not necessary, for me, those two functions had constant time like a normal list when in reality because no traversal was required the runtime complexity should have been O(1). Thus, if I had a linked list of significant size my *Min* and *Max* methods would be very slow. Instead, I should have simply returned the head as the min value and the tail as max value so that constant time would be maintained no matter the size of the list.

Conclusion

Overall, this lab was a great piece of deliberate practice with linked lists to understand how they can be manipulated in several ways. Ever since I learned about linked lists I value their importance in CS because they are key to understanding other data structures like queues, stacks, and trees. It's important to understand the running time for various functions because in a real-world situation that could deal with millions of pieces of data will the implemented linked lists methods be efficient.

Appendix

```
'''
    Course: CS2302 Data Structures Fall 2019
    Author: Bryan Ramos [88760110]
    Assignment: Lab 3 Linked Lists
    Instructor: Dr. Olac Fuentes
    TA: Anindita Nath
    Last Modified: October 4th 2019
    Purpose:
'''

import random as rand
import math
import sys

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

#List Functions
class List(object):
    # Constructor
    def __init__(self):
        self.head = None
        self.tail = None

    def Append(self, data):
        node = Node(data)
        current = self.head
        previous = None

        while current is not None and current.data < data:
            previous = current
            current = current.next

        if previous == None:
            self.head = node
        else:
            previous.next = node

        node.next = current

    def Print(self):
        # Prints list L's items in order using a loop
        t = self.head
        while t is not None:
            print(t.data, end=' ')
            t = t.next
        print()

    def Insert(self, i):
        self.Append(i)

    def Delete(self, i):
        previous = None
        t = self.head

        try:
            while t.data != i:
                previous = t
                t = t.next
        except:
            return None

        if t == self.head:
            self.head = t.next
        else:
            previous.next = t.next

    def IndexOf(self, i):
        t = self.head
        index = 0
        while t is not None:
            if t.data == i:
                return index
            index += 1
            t = t.next
        return -1

    def Clear(self):
        self.head = None

    def Min(self):
        t = self.head
        if t is None:
            return math.inf
        min = sys.maxsize
        while t is not None:
            if t.data < min:
                min = t.data
            t = t.next
        return min

    def Max(self):
        t = self.head
        if t is None:
            return -math.inf
        max = -sys.maxsize - 1
        while t is not None:
            if t.data > max:
                max = t.data
            t = t.next
        return max

    def HasDuplicates(self):
        t = self.head
        if t is None:
            return False
        while t.next is not None and t.next is not None:
            if t.data == t.next.data:
                return True
            t = t.next
        return False

    def Select(self, k):
        if self is None:
            return math.inf
        index = 0
        t = self.head
        while t is not None:
            if index == k:
                return t.data
            index += 1
            t = t.next

    def Merge(self, M):
        s = M.head
        while s is not None:
            self.Append(s.data)
            s = s.next

if __name__ == "__main__":
    L = List() # create empty list to use for main output
    M = List() # list for merging

    # fill list with random values between 0-50
    for i in range(0, 10):
        L.Append(i)

    for i in range(0, 10):
        M.Append(rand.randint(0,12))

    while (True):
        print("\n1. Print\n2. Insert\n3. Delete\n4. Merge\n5. IndexOf")
        print("6. Clear\n7. Min\n8. Max\n9. HasDuplicates\n10. Select\n11. Exit\n")

        option = input("Please choose an option: ")

        if option == "1": # print all contents of list
            L.Print()

        elif option == "2": # insert node
            try:
                number = int(input("Enter a number to add to the list: "))
            except ValueError:
                print("Invalid input provided! Please provide an integer number as inp
ut.")
                break
            L.Insert(number)

        elif option == "3": # delete node
            try:
                number = int(input("Enter a number to remove from the list: "))
            except ValueError:
                print("Invalid input provided! Please provide an integer number as inp
ut.")
                break
            L.Delete(number)

        elif option == "4": # merge
```

```
            L.Print()
            M.Print()
            L.Merge(M)
            L.Print()

        elif option == "5": # indexof
            try:
                key = int(input("Enter a number to find in the list: "))
            except ValueError:
                print("Invalid input provided! Please provide an integer number as inp
ut.")
                break
            print(key, "is found at the following index:", L.IndexOf(key))

        elif option == "6": # clear list - delete all elements
            L.Clear()

        elif option == "7": # min value of list
            print("Min value in list:", L.Min())

        elif option == "8": # max value of list
            print("Max value in list:", L.Max())

        elif option == "9": # returns whether duplicates exist in list
            print("Duplicates in list:", L.HasDuplicates())

        elif option == "10": # returns the kth smallest element in the list
            try:
                K = int(input("Enter a number for k: "))
            except ValueError:
                print("Invalid input provided! Please provide an integer number as inp
ut.")
                break
            print("Value at position k:", L.Select(K))

        elif option == "11": # quit program
            print("Bye! Thanks for using the program!")
            break

        else: # invalid
            print("Invalid input!")
```

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