**Lab Report 3: Sorted Linked List**

**I. Introduction**

The program description should be able to manage and create linked lists that are sorted in ascending order at all times. The functions that this program should implement are printing the list in order, inserting a new node, deleting a node, merging two sorted lists into one, return the index of an element in the list, clearing the list, returning the smallest element, returning the largest element, return if the list has duplicate values, returning the kth smallest number in a list. Finally, a function for creating a random list from the size that the user input was created as well.

**II. Proposed solution design and implementation**

The lab was structured very easily in the sense that it already had a structure on the order to work on things. I worked on every function in order. The printing function was the same as with a normal linked list.

The insertion required more thinking into deciding where the new node was going to be placed. This traversed the list until the next element was higher than the one that was going to be placed. If the new node had a lower value than the head, then it became the new head, if it was higher than the tail, then it became the new tail.; This avoided having to traverse the list in these cases.

Deleting a node consisted of traversing the list to find the element that will be deleted. After that, it was necessary to make the right connections of nodes.

Merging two lists was the trickiest for me. Since the merged list was going to be one of them, I could not create a new list in which to merge both lists. This was also going to occupy a lot of memory. The first issue encountered was how to add the second list before the first one if that was going to be the case. How I solved this was that if the first element in the second list was lower than the first element in the first list, the lowest became the head of the first list, and then the second element of the second list became the head of the second list. Now all elements were going to be added after the head and there were no more troubles. How the function merges the lists in the correct order is with two pointers in both lists, comparing the values until the next element from list 2 can be placed after the current element in list 1. If the list 1 ends, but the list 2 does not, then the rest of the elements at list 2 get added at the end.

Returning the index of an element required a counter and traversing the list. Each time the pointer moved to the next element, the counter incremented. If the element was not on the list. The function returns -1.

ttempt to solve) the problem? Provide an informal, high-level description. Description of your code (not the actual code). Explain the design choices you made, including how you broke the program into modules, your user interface, input and output, etc.

Clearing the list was setting the tail and the had to None.

Returning the smallest element is returning the head, because the list is sorted, the head must be the smallest element.

Returning the largest element is returning the tail, because the list is in ascending order and the last element will always be the largest.

Returning if the list has duplicate values consisted on traversing the list and comparing the current element with the next. Since the list is sorted, if the list has duplicate values, they will be right next to each other.

The kth smallest element consisted on returning the element at index k. This was done traversing the list until k and returning that element. If k was not in the list, math.inf was returned.

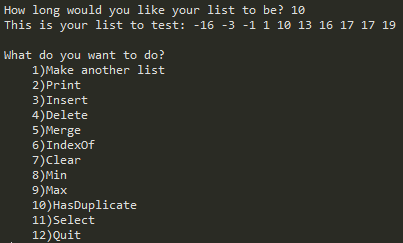
To create a function that makes a random list of the user’s input size, a for loop was created that appends a random number between -20 and 20.

Finally, I created the user interface for testing the list. The user can create a list first that can be put through all these tests and can change the list at any point. A while loop with a boolean variable “stay” was used to stay on the menu.

**III. Experimental results**

I created a menu that lets you test any random list with all the functions. These are the results:

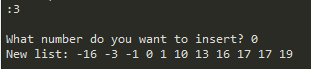
Testing a list of size 10:



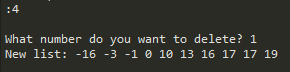
Printing



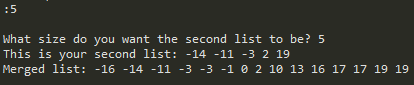
Inserting a number in the middle



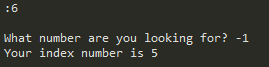
Deleting a number in the middle



Merging 2 lists



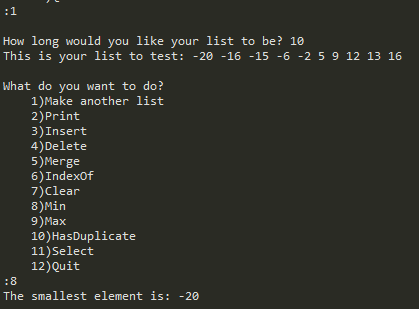
Index Of



Clearing a list



Making a new list and looking for the smallest element



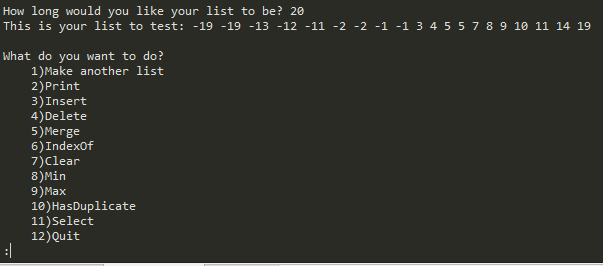
Looking for the largest element



Has Duplicates



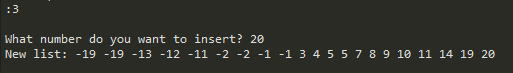
Testing a list of size 20:



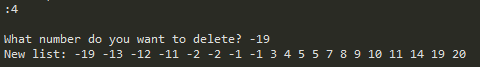
Printing



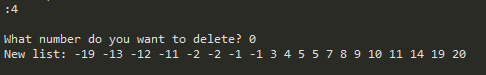
Inserting at the end



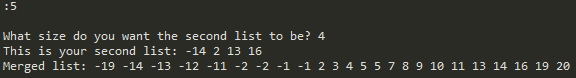
Deleting a negative number



Deleting a number that is not on the list



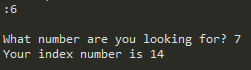
Merging



Has Duplicates



Index of



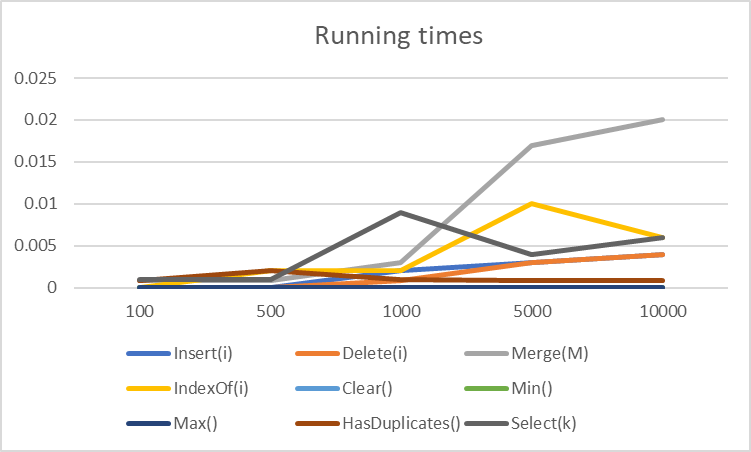
Kth smallest



**Tables:**

|  |  |  |
| --- | --- | --- |
| **FUNCTION** | **SortedList** | **List** |
| Print() | O(n) | O(n) |
| Insert(i) | O(n) | O(1) |
| Delete(i) | O(n) | O(n) |
| Merge(M) | O(n) | O(1) |
| IndexOf(i) | O(n) | O(n) |
| Clear() | O(1) | O(1) |
| Min() | O(1) | O(n) |
| Max() | O(1) | O(n) |
| HasDuplicates() | O(n) | O(n2) |
| Select(k) | O(n) | O(n2) |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **FUNCTION** | **SortedList** | Input: 100 | 500 | 1000 | 5000 | 10000 |
| Print() | O(n) | 0.009 | 0.03 | 0.16 | 1.16 | 2.24 |
| Insert(i) | O(n) | 0.0 | 0.0 | 0.0019 | 0.003 | 0.004 |
| Delete(i) | O(n) | 0.0 | 0.0 | 0.0009 | 0.003 | 0.004 |
| Merge(M) | O(n) | 0.0009 | 0.0009 | 0.003 | 0.017 | 0.02 |
| IndexOf(i) | O(n) | 0.0 | 0.002 | 0.002 | 0.01 | 0.006 |
| Clear() | O(1) | 0.0 | 0.0 | 0.0 | 0.0 | 0 |
| Min() | O(1) | 0.0 | 0.0 | 0.0 | 0.0 | 0 |
| Max() | O(1) | 0.0 | 0.0 | 0.0 | 0.0 | 0 |
| HasDuplicates() | O(n) | 0.0009 | 0.002 | 0.001 | 0.0009 | 0.0009 |
| Select(k) | O(n) | 0.001 | 0.001 | 0.009 | 0.004 | 0.006 |

(without print) 

**IV. Conclusions**

From this project I understood better how to work with linked lists, to bee careful of not losing the head and reassigning the tail. I learned that testing the program multiple times helps find problems that were unforeseen. I’d had trouble with merge when the items that were smaller than the original list were appended at the end, and I realized this only by testing. I learned that sorting the list has both advantages and disadvantages with respect to time. A linked list cannot be traversed as an array can, and therefore traversing the whole list to get to an element near the end is necessary. It is very useful, however, to have a pointer to the tail. I can wonder if using a doubly linked list increments or reducer time complexity.

**V. Appendix – Source code**

#"""

# COURSE: CS 2302 Data Structures

# AUTHOR: Elisa Jimenez Todd

# ASSIGNMENT: Lab 3: Sorted Linked Lists

# INSTRUCTOR: Olac Fuentes

# TA: Anindita Nath

# =============================================================================

# DATE: 10/04/2019

# =============================================================================

#

# Program description: This program manages and creates sorted lists in

# ascending order. The functions implented are:

# 1)Print: prints the list in order

# 2)Insert: inserts a new node on the correct position

# 3)Delete: deletes an element from the list

# 4)Merge: merges two lists into the first

# 5)IndexOf: returns the index(position) of an element in the list

# 6)Clear: deletes all elements from the list

# 7)Min: returns smallest element

# 8)Max: returns largest element

# 9)HasDuplicate: Returns true or false if there are duplicate numbers

# 10)Select: returns kth smallest element on a list

#"""

import math

import random

class Node(object):

# Constructor

def \_\_init\_\_(self, data, next=None):

self.data = data

self.next = next

#List Functions

class SortedList(object):

# Constructor

def \_\_init\_\_(self):

self.head = None

self.tail = None

def Print(self):

# Prints list's items in order

temp = self.head

while temp is not None:

print(temp.data, end=' ')

temp = temp.next

print() # New line

def Insert(self, i):

#if list is empty

if self.head == None:

self.head = Node(i)

self.tail = self.head

#look for i's position

else:

currNode=self.head

#if i is smaller than head, puts node as head

if i<=currNode.data:

self.head = Node(i)

self.head.next = currNode

#if i is larger than the elements at the list

elif i>=self.tail.data:

self.tail.next = Node(i)

self.tail = self.tail.next

#if i is larger than head looks for position

else:

while i>currNode.next.data:

currNode = currNode.next

tempNode = currNode.next #inserts the node

currNode.next = Node(i)

currNode.next.next = tempNode

def Delete(self,i):

#if list is empty

if self.head != None:

#looks for i

currNode = self.head

if currNode.data == i: #if i is head

self.head = currNode.next

return

while currNode.next.data != i: #look for position of i

currNode = currNode.next

if currNode.next == None:

return

if currNode.next.next == None: #removes last node

self.tail = currNode

self.tail.next = None

else: #removes node inside

currNode.next = currNode.next.next

def Merge(self,M):

if self.head == None: #makes M the list

self.head = M.head

self.tail = M.tail

#if self has something to add

elif M.head != None:

selfNode = self.head

newNode = M.head

#add before the start

if newNode.data < selfNode.data:

#removes node from M

M.head = newNode.next

#append before

newNode.next = selfNode

self.head = newNode

newNode = M.head

#reset

selfNode = self.head

newNode = M.head

#merges

while selfNode.next != None and newNode != None:

#find position to merge M

if newNode.data >= selfNode.data and newNode.data < selfNode.next.data:

#removes node from M

M.head = newNode.next

#inserts node of M to self

selfNode.next, newNode.next = newNode, selfNode.next

#next values

selfNode=selfNode.next

newNode = M.head

#adds rest of list if any

if newNode != None:

self.tail.next = newNode

self.tail = M.tail

def IndexOf(self,i):

#i would not be on list if smaller than first element or larger than last

if self.head == None or i> self.tail.data or i < self.head.data:

return -1

counter = 0 #index

currNode = self.head

while currNode.data < i : #iterate

currNode = currNode.next

counter += 1

if currNode.data > i: #does not have to traverse all list

return -1

return counter

def Clear(self):

self.head = None #makes head and tail none

self.tail = None

def Min(self): #returns first element, or math inf

return math.inf if (self.head == None) else self.head.data

def Max(self): #returns last element, or -math inf

return -math.inf if (self.tail == None) else self.tail.data

def HasDuplicates(self):

#if list is empty returs false

if self.head == None:

return False

#iterates to find a duplicate

currNode = self.head

while currNode != self.tail:

if currNode.data == currNode.next.data:

return True

currNode = currNode.next

#found no duplicates, returns false

return False

def Select(self,k):

#if list is empty returns math inf

if self.head == None:

return math.inf

currNode = self.head

#iterates to find k

for i in range(k):

currNode = currNode.next

#k is greater than length

if currNode == None:

return math.inf

return currNode.data

def FillList(self, n):

self.Clear()

for i in range(int(n)):

self.Insert(random.randint(-20,20))

#MAIN

stay = True #Stay on menu

print('Welcome to Sorted Lists!')

while stay:

print('What would you like to do?')

option1 = input('\t1)Make a list to test\n\t2)Exit\n:')

if option1 == '2':

stay = False

if option1 =='1':

#creates list

L = SortedList()

listSize = input('How long would you like your list to be? ')

L.FillList(listSize)

print('This is your list to test: ', end = '')

L.Print()

#Testing portion

while stay:

#Displays menu options and stores user's answer

option2 = input('What do you want to do?\n\t1)Make another list\n\t2)Print\n\t3)Insert\n\t4)Delete\n\t5)Merge\n\t6)IndexOf\n\t7)Clear\n\t8)Min\n\t9)Max\n\t10)HasDuplicate\n\t11)Select\n\t12)Quit\n:')

#Make new list

if option2 == '1':

listSize = input('How long would you like your list to be? ')

L.FillList(listSize) #Call to make new list

print('This is your list to test: ', end = '')

L.Print()

#Print List

elif option2 == '2':

L.Print() #call to print

#Insert

elif option2 == '3':

num = int(input('What number do you want to insert? '))

L.Insert(num) #Call to insert

print('New list: ', end='')

L.Print()

#Delete

elif option2 == '4':

num = int(input('What number do you want to delete? '))

L.Delete(num) #Call to delete

print('New list: ', end='')

L.Print()

#Merge

elif option2 == '5':

L2 = SortedList()

listSize2 = int(input('What size do you want the second list to be? '))

L2.FillList(listSize2) #create new list to merge

print('This is your second list: ', end = '')

L2.Print()

L.Merge(L2) #Call to merge

print('Merged list: ', end='')

L.Print()

#Index Of

elif option2 == '6':

num = int(input('What number are you looking for? '))

print('Your index number is ' + str(L.IndexOf(num))) #Call to index of

#Clear

elif option2 == '7':

print('Clearing list: ', end ='')

L.Clear() #Call to clear

L.Print()

#Min

elif option2 == '8':

print('The smallest element is: ', end ='')

print(L.Min()) #Call to min

#Max

elif option2 == '9':

print('The largest element is: ', end ='')

print(L.Max()) #Call to max

#Has Duplicate

elif option2 == '10':

if L.HasDuplicates(): #call to has duplicates

print('This list does have duplicates')

else:

print('This list does not have duplicates')

#Select

elif option2 == '11':

num = int(input('What kth smallest element are you looking for?'))

print('The kth smallest element is: ' + str(L.Select(num))) #call to select

#Quit

elif option2 == '12':

stay = False #Boolean false to exit the loop

else:

print('Invalid input. Type a number from 1 to 12')

else:

print('Invalid input. Type a number from 1 to 2')

**VI – Academic Honesty Certification**

I 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.

x- Elisa Jimenez Todd