## Introduction

The problem proposed to us was to provide 4 different types sorting algorithms of nonnative python list which use nodes (linked list)

The solutions I could come up with for the problems were in this order:

Bubble sort – For bubble sort we began by creating 2 while loops, one for iterating the whole list, and the other one is for iterating the list until we find no sorting left. This sorting algorithm did not make me struggle a whole lot, but it was pretty fun.

Merge sort – For merge sort, a lot of problems were met. I had trouble with the merge method, along with methods that were called from my methods. I did like debugging, but it did get a bit hectic. Merge sort mainly consisted of 2 methods; merge and mergesort. Mergesort took in l and n as inputs and increased n while iterating recursively. Mergesort also went through the list and repeatedly split it using a method I created (split). The split method split the list into 2, and the recursive call of merge sort called twice for both left and right list. After the recursive calls, we called merge, which combined both lists using loops.

Quicksort – for quicksort I used a quicksort method, and the same combine method from above. Quicksort also used a pivot method which got the last element of our list. Quicksort did not split, but it did create 2 new list which held high and low from pivot numbers. When the numbers were found to be higher or lower, and it was sorted, I used combine to put all the numbers back into 1 list.

Modsort – I did not complete this problem, but did attempt to solve it. The way I chose to tackle the problem, was to check the first half of the numbers and then continue to do so until we reach the end of the method. The problem with this solution is that it will not check the first half.

## Experimental

Lots of issues I came across required lots of debugging. Once I had finished the problems and got solutions, I reached more problems. One of the issues I could not find a solution for was a problem in my merge sort that returned n = 1 and should have checked a lot more than just once.

## Learned

This lab taught me more about debugging my own code in python, and also taught me that more about linked list, and how difficult some problems can be.

## Conclusion

The lab taught me more things about python, showed me sorting and explained a couple of things to me. This lab also helped me study up a bit before the exam and showed me a lot of mistakes I could have made on the test.

Big O of Bubble Sort - O(n^2)

Big O of Merge Sort - O(n)

Big O of Quick Sort – O(nlog(n))

Big O of Mod Sort – O(n)

## Appendix

1. #Cesar Lopez
2. #Lab 2
3. #Node Functions
5. **class** Node(object):
6. # Constructor
7. **def** \_\_init\_\_(self, item, next=None):
8. self.item = item
9. self.next = next
11. **def** PrintNodes(N):
12. **if** N != None:
13. **print**(N.item, end=' ')
14. PrintNodes(N.next)
16. **def** PrintNodesReverse(N):
17. **if** N != None:
18. PrintNodesReverse(N.next)
19. **print**(N.item, end=' ')
21. #List Functions
22. **class** List(object):
23. # Constructor
24. **def** \_\_init\_\_(self):
25. self.head = None
26. self.tail = None
28. **def** IsEmpty(L):
29. **return** L.head == None
31. **def** Append(L,x):
32. # Inserts x at end of list L
33. **if** IsEmpty(L):
34. L.head = Node(x)
35. L.tail = L.head
36. **else**:
37. L.tail.next = Node(x)
38. L.tail = L.tail.next
40. **def** Print(L):
41. # Prints list L's items in order using a loop
42. temp = L.head
43. **while** temp **is** **not** None:
44. **print**(temp.item, end=' ')
45. temp = temp.next
46. **print**()  # New line
48. **def** PrintRec(L):
49. # Prints list L's items in order using recursion
50. PrintNodes(L.head)
51. **print**()
53. **def** Remove(L,x):
54. # Removes x from list L
55. # It does nothing if x is not in L
56. **if** L.head==None:
57. **return**
58. **if** L.head.item == x:
59. **if** L.head == L.tail: # x is the only element in list
60. L.head = None
61. L.tail = None
62. **else**:
63. L.head = L.head.next
64. **else**:
65. # Find x
66. temp = L.head
67. **while** temp.next != None **and** temp.next.item !=x:
68. temp = temp.next
69. **if** temp.next != None: # x was found
70. **if** temp.next == L.tail: # x is the last node
71. L.tail = temp
72. L.tail.next = None
73. **else**:
74. temp.next = temp.next.next
76. **def** PrintReverse(L):
77. # Prints list L's items in reverse order
78. PrintNodesReverse(L.head)
79. **print**()
81. **def** Median(L):
82. #sets temp as head, then calls get element, whick returns the element at the length of temp/2, which is the middle
83. temp = L.head
84. **return** ElementAt(temp,GetLength(temp)//2)
86. #Gets the length of our nodes, if empty, returns 0
87. # creates n, creates temp, and while temp is not equal to none, updates both n and temp, until None is reached.
88. **def** GetLength(L):
89. **if** L **is** None:
90. **return** 0
91. **else**:
92. n = 0
93. temp = L.head
94. **while** temp **is** **not** None:
95. n +=1
96. temp = temp.next
97. **return** n
98. #uses while loop to find the location of the element at location n.
99. #Checks if c is None, and if yes, returns None
100. **def** ElementAt(c, n):
101. **if**(c **is** None):
102. **return** None
103. **else**:
104. **while**(n > 0):
105. c = c.next
106. n-=1
107. **return** c.item
109. #Splits list into 2 pieces, lower, and upper.
110. #creates I and J to hold length, and half the length.
111. #uses n to get the first half, and uses n to get second half.
112. #Appends left for all values less than half, and right for values bigger than half
113. #returns the 2 new list
114. **def** Split(L):
115. temp = L.head
116. left = List()
117. right = List()
118. i = GetLength(L)
119. j = GetLength(L)//2
120. n = 0
121. **while** n < j:
122. Append(left, temp.item)
123. n+=1
124. temp = temp.next
125. **while** n < i:
126. Append(right, temp.item)
127. n+=1
128. temp = temp.next
129. **return** left, right

132. **def** BubbleSort(L, n):
133. unsorted = True
134. **while** unsorted:#Uses while loop to proceed until no longer able to
135. temp = L.head
136. unsorted = False
137. **while** temp.next **is** **not** None:
138. n+=1
139. **if** temp.item > temp.next.item:#Swaps if current item is greater than next item
140. temp2 = temp.item
141. temp.item = temp.next.item
142. temp.next.item = temp2
143. unsorted = True#Allows us to keep doing the while loop
144. temp = temp.next
145. **return** temp, n
147. **def** MergeSort(L, n):
148. **if** L **is** None **or** GetLength(L) == 1:
149. n+=1
150. **return** L, n
151. **else**:
152. n+=1
153. l1, l2 = Split(L)#splits list into 2
154. MergeSort(l1, n)#calls recursive calls for left and right list(s)
155. MergeSort(l2, n)
156. **return** Merge(l1, l2, n), n#calls merge method, which merges the 2 list and compares
158. **def** Merge(l, r, n):
159. merge = List()
160. left = l.head
161. right = r.head
162. n+=1
163. **while**(left **is** **not** None **and** right **is** **not** None):
164. **if**(left.item > right.item):#checks if left list is bigger than right, and appends right to new list if true
165. n+=1
166. Append(merge, right.item)
167. right = right.next
168. **else**:#else statement that appends left to the new list if the first case is false
169. n+=1
170. Append(merge, left.item)
171. left = left.next
172. **while**(left **is** **not** None):#appends leftover items
173. n+=1
174. Append(merge, left.item)
175. left = left.next
176. **while**(right **is** **not** None):#appends leftover items
177. n+=1
178. Append(merge, right.item)
179. right = right.next
180. **return** merge, n
182. **def** QuickSort(L, n):#checks for empty list
183. **if**(L **is** None **or** IsEmpty(L)):
184. **return** L
185. **else**:
186. hold = Pivot(L)#gathers pivot (last node)
187. temp = L.head
188. high = List()
189. low = List()
190. i = GetLength(L)-1#allows us to stop before reaching the last node
191. j = 0
192. hic = 0#high counter
193. loc = 0#low counter
194. **while**(j<i):
195. **if**(hold > temp.item):#checks if item is less than pivot, and is placed in lower list if true
196. Append(low, temp.item)
197. temp = temp.next
198. j+=1
199. n+=1
200. loc+=1
201. **else**:#if item is greater than pivot, it is placed in the higher list
202. Append(high, temp.item)
203. temp = temp.next
204. j+=1
205. n+=1
206. hic+=1
207. QuickSort(low, n)#recurses both new list
208. QuickSort(high, n)
209. **if**(loc < hic):#checks if pivot is going to left or right list
210. Prepend(high, temp.item)
211. n+=1
212. merged = List()
213. merged = Merge(low, high, n)
214. **return** merged, n
215. **else**:#if pivot did not go into high list, it goes here, to the low list
216. Append(low, temp.item)
217. n+=1
218. merged = List()
219. merged = Merge(low, high, n)
220. **return** merged, n
221. #prepend method, to add things at the fornt of a list
222. **def** Prepend(L,x):
223. **if** IsEmpty(L):
224. L.head = Node(x)
225. L.tail = L.head
226. **else**:
227. L.head=Node(x,L.head)
228. #gets us the last node item
229. **def** Pivot(L):
230. n = GetLength(L)-1
231. temp = L.head
232. **for** i **in** range(n):
233. temp = temp.next
234. **return** temp.item
236. #incomplete, but tried following instructions
237. **def** Mod(L, n):
238. p = Pivot(L)
239. m = Median(L)
240. e = List()
241. **for** i **in** range(m):
242. n+=1
243. **if**(p > L.head.item):
244. Append(e, L.head.item)
245. L.head = L.head.next
246. **else**:
247. L.head = L.head.next
248. Mod(e, n)
249. **return** e, n
251. L = List()
252. **print**(IsEmpty(L))
253. **for** i **in** range(5):
254. Append(L,i)
255. Print(L)
256. PrintRec(L)
257. PrintReverse(L)
258. a = List()
259. b = List()
260. c = List()
261. a, n1 = BubbleSort(L, 0)
262. **print**(n1)
263. b, n2 = MergeSort(L, 0)
264. **print**(n2)
265. c, n3 = QuickSort(L, 0)
266. **print**(n3)
267. #Mod(L, 0)