**Author:** Julian Gonzalez

**Assignment:** Lab 2 Report

**Course:** CS 2302 - Data Structures 10:30-11:50

**Instructor:** Fuentes, Olac

**T.A.:** Nath, Anindita

***Introduction***

In this lab we are using user defined Node and List objects, and with them we are implementing different sorting algorithms. In this lab I was able to implement bubble sort, merge sort, quick sort, and finally a modified version of quick sort. I will be also comparing the running times of all these sorting methods, the way I will do this is by keeping track of the number of comparisons that will happen when sorting N random sized lists.

My proposed solution design and implementation would involve creating 4 List filled with random numbers of N size, then sorting the lists with the 4 Sorting algorithms then finding the median of each of the lists.

**Bubble Sort**

I had to iterate through a list of n size. For every loop/iteration the list will have one less item to sort through. The comparisons that happen every iteration compare elements next to each other. Then swaps the elements in ascending order. This will have bigger items go to the “top” of the list eventually sorting it

**Merge Sort**

Implementation has the list being separated into to equal sized list. Keep doing this until you can’t split the lists. Start merging the lists together again while checking if the value of the left list is less than the other append into the list that fits. Then finally after checking the lists, I use a method to combine the two sorted list into one list that should be in order.

**Quick Sort**

For quicksort I first selected a pivot which would be the first element in the list. Then iterate through the rest of the list while comparing them to the pivot, while doing so I would put them into two separate lists this will continue to happen with the two lists in separate recursive calls. Then eventually the two list will be put together with the pivot in mind so items smaller than the pivot will be on the left side of the pivot and larger on the right side of the pivot.

**Modquicksort**

I was unable to finish this problem in a working manner, but I did attempt to solve it. Everything that I had done left the list unsorted at the end of the function call.

***Experimental***

For my runs of the programs, I did the most in my capability to track the number of comparisons that happened in each of the sorting method. I used randomly created lists of different sizes (10,100,1000,2000). I ran each size at least 3 times and got the average comparisons for them.

To keep track of the comparisons that happened in each sorting method I created a variable called comparisons in the List object that would keep track of them. Then I tried to pinpoint places where comparisons would happen inside each of the sorting methods and inserted this line of code (L.comparisons +=1).

One thing that I noticed is that Merge sort and quick sort had very similar number of comparisons. I did not include modified quicksort as I did not have data recorded for it.

***Conclusion***

The lab has taught me how many different things can affect an algorithm time complexity especially in python. Using comparisons instead of time also showed me the inner workings of how sorting methods work. This lab also gave me greater insight on how Time complexity and Big-O notation works.

Appendix:

1. **import** random
2. #Node Functions#######################################
3. **class** Node(object):
4. # Constructor
5. **def** \_\_init\_\_(self, item, next=None):
6. self.item = item
7. self.next = next
9. **def** PrintNodes(N):
10. **if** N != None:
11. **print**(N.item, end=' ')
12. PrintNodes(N.next)
14. **def** PrintNodesReverse(N):
15. **if** N != None:
16. PrintNodesReverse(N.next)
17. **print**(N.item, end=' ')
19. #List Functions###########################################
20. **class** List(object):
21. # Constructor
22. **def** \_\_init\_\_(self):
23. self.head = None
24. self.tail = None
25. self.length = 0
26. self.comparisons = 0
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. L.length +=1
37. **else**:
38. L.tail.next = Node(x)
39. L.tail = L.tail.next
40. L.length +=1
42. **def** Print(L):
43. # Prints list L's items in order using a loop
44. temp = L.head
45. **while** temp **is** **not** None:
46. **print**(temp.item, end=' ')
47. temp = temp.next
48. **print**()  # New line
50. **def** PrintRec(L):
51. # Prints list L's items in order using recursion
52. PrintNodes(L.head)
53. **print**()
55. **def** Remove(L,x):
56. # Removes x from list L
57. # It does nothing if x is not in L
58. **if** L.head==None:
59. **return**
60. **if** L.head.item == x:
61. **if** L.head == L.tail: # x is the only element in list
62. L.head = None
63. L.tail = None
64. **else**:
65. L.head = L.head.next
66. **else**:
67. # Find x
68. temp = L.head
69. **while** temp.next != None **and** temp.next.item !=x:
70. temp = temp.next
71. **if** temp.next != None: # x was found
72. **if** temp.next == L.tail: # x is the last node
73. L.tail = temp
74. L.tail.next = None
75. **else**:
76. temp.next = temp.next.next
78. **def** PrintReverse(L):
79. # Prints list L's items in reverse order
80. PrintNodesReverse(L.head)
81. **print**()
83. **def** Prepend(L,x):
84. **if** IsEmpty(L):
85. L.head = Node(x)
86. L.tail = L.head
87. L.length+=1
88. **else**:
89. L.head=Node(x,L.head)
90. L.length+=1
92. **def** getLength(L):
93. temp = L.head
94. count = 0
95. **while** temp **is** **not** None:
96. count+=1
97. temp = temp.next
98. **return** count
99. **def** search(L,i):
100. cur = L.head
101. count = 0
103. **while** count != i:
104. cur = cur.next
105. count += 1
106. **return** cur.item

109. #########################################################################
110. ## BubbleSort
112. **def** bubbleSort(L):
113. unsorted = True
114. **while** unsorted:#loops until the list has done every comparison it can
115. temp = L.head
116. unsorted = False
117. **while** temp.next **is** **not** None:
118. **if** temp.item > temp.next.item:#swaps the the current item from the list if it is greater then the next tiem
119. temp2 = temp.item
120. temp.item = temp.next.item
121. temp.next.item = temp2
122. L.comparisons +=1
123. unsorted = True#keeps in the while loop as something was swapped
124. temp = temp.next
126. #############################################################
127. ## MergeSort
128. **def** mergeSort(L):
129. **if** L.length > 1:
130. left = List()
131. right = List()
132. left,right=split(L,left,right)#calls the split function
134. mergeSort(left)#
135. mergeSort(right)
137. temp = left.head
138. temp2 = right.head
139. **while** temp != None **and** temp2 != None:#this sorts the left and right list compared to each other
140. L.comparisons += 1#marks the comparisons happening
141. **if** temp.item < temp2.item:
142. Append(L, temp.item)
143. temp = temp.next
144. **else**:
145. Append(L, temp2.item)
146. temp2 = temp2.next
148. merge(L,temp,temp2)#merges left and right lists

151. **def** split(L,left,right):
152. temp = L.head
153. i=0
154. **while** i < getLength(L)//2:#while loop to create a left list that is half the size of the L list
155. Append(left, temp.item)
156. Remove(L, temp.item)#removes from List so when right list is created it wont have left list numbers
157. temp = temp.next
158. i+=1
159. **while** temp **is** **not** None:#while loop to add to right list from the remains of L list
160. Append(right,temp.item)
161. Remove(L,temp.item)
162. temp = temp.next
164. **return**(left,right)
166. **def** merge(L,left,right):
167. **while** left **is** **not**  None:#adds the items from the left list into the L list
168. Append(L,left.item)
169. left = left.next
170. **while** right **is** **not** None:#add the items from the right list to the L list
171. Append(L,right.item)
172. right = right.next
173. #############################################################
174. ## QuickSort
175. **def** quickSort(L):
176. **if** L.length > 1:
177. pivot = L.head.item#Item to be compared too
178. left = List()
179. right = List()
180. temp = L.head.next
181. **while** temp **is** **not** None:#sorts left and right list according to the pivot
182. L.comparisons += 1#marks the comparisons happening
183. **if** temp.item < pivot:
184. Append(left,temp.item)
185. **else**:
186. Append(right,temp.item)
187. temp = temp.next
188. #recusive calls to sort the lists again from less than pivot and greater and pivot
189. quickSort(left)
190. quickSort(right)
192. **if** IsEmpty(left):#makes the pivot the last element of the left list
193. Append(left,pivot)
194. **else**:
195. Prepend(right,pivot)#makes the pivot the first element of the right list
197. **if** IsEmpty(left):#if somehow the left list becomes completly empty this makes the right list head and tail the equal to L
198. L.head = right.head
199. L.tail = right.tail
200. left.tail.next = right.head#merges the two list by having the tail of the left list the head of the right list
201. L.head = left.head#makes parameter L head equal to left list head
202. L.tail = right.tail#makes parameter L tail equal to right list tail
204. #############################################################
205. # ModQuickSort
206. **def** modQuickSort(L,m):
207. **if** L.length > 0:
208. pivot = L.head.item#Item to be compared too
209. left = List()
210. right = List()
211. temp = L.head.next
212. **while** temp **is** **not** None:#sorts left and right list according to the pivot
213. **if** temp.item < pivot:
214. Append(left,temp.item)
215. **else**:
216. Append(right,temp.item)
217. temp = temp.next
219. **if** m==getLength(left) **or** (m==0 **and** m==getLength(left)):
220. **return** pivot
221. **if** m > getLength(left):
222. **return** modQuickSort(right, m-getLength(left)-1)
223. **if** m <= getLength(left):
224. **return** modQuickSort(left, m)
225. ##        if m < left.length:
226. ##            return modQuickSort(left,m)
227. ##        elif m==0 and m==getLength(left):
228. ##            return pivot
229. ##        elif m == left.length:
230. ##            return pivot
231. ##        else:
232. ##            return modQuickSort(right,m-left.length-1)

235. #############################################################
237. **def** newList(n):#function to create a list of n length with random ints in range of 1 to 100
238. L = List()
239. **for** x **in** range(0,n):
240. num = random.randint(1,100)
241. Append(L,num)
243. **return** L
244. L1 = List()
245. L2 = List()
246. L3 = List()
247. L4 = List()
249. L1=newList(20)
250. L2=newList(20)
251. L3=newList(20)
252. L4=newList(20)
254. **print**('Bubble sort for list 1')
255. **print**('Unsorted List 1:', end=' ')
256. Print(L1)
257. bubbleSort(L1)
258. **print**('Sorted List 1:', end=' ')
259. Print(L1)
260. **print**('Median is',end=' ')
261. **print**(search(L1, L1.length // 2))
262. **print**("Num of comparisons")
263. **print**(L1.comparisons)
265. **print**()
267. **print**('Merge Sort for list 2')
268. **print**('Unsorted List 2:', end=' ')
269. Print(L2)
270. mergeSort(L2)
271. **print**('Sorted List 2:', end=' ')
272. Print(L2)
273. **print**('Median is',end=' ')
274. **print**(search(L2, getLength(L2) // 2))
275. **print**("Num of comparisons")
276. **print**(L2.comparisons)

279. **print**()
281. **print**('Quick Sort for list 3')
282. **print**('Unsorted List 3:', end=' ')
283. Print(L3)
284. quickSort(L3)
285. **print**('Sorted List 3:', end=' ')
286. Print(L3)
287. **print**('Median is',end=' ')
288. **print**(search(L3, L3.length // 2))
289. **print**("Num of comparisons")
290. **print**(L3.comparisons)

293. **print**()
295. **print**('Better Quick Sort for list 4')
296. **print**('Unsorted List 4:', end=' ')
297. Print(L4)
298. modQuickSort(L4,getLength(L4) // 2)
299. **print**('Sorted List 4:', end=' ')
300. Print(L4)
301. **print**('Median is',end=' ')
302. **print**(search(L4, L4.length // 2))
303. **print**("Num of comparisons")
304. **print**(L4.comparisons

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.

* Julian Gonzalez