Name: Emmanuel Jesus Estallo	
Student Number: 2018-02355	Section: MABMXY

Answer the following questions with clear and concise statements given the code snippets below.

Answers may be handwritten or typewritten on the spaces provided. You may attach additional pages if more space is needed and state all assumptions used. References, both offline and online, need to be cited somewhere in the space provided if ever they are used.

```
class LLNode:
  def __init__(self, data=0, next):
    self.data = data
    self.next = next
```

a. (2 *pts*) Considering the implementation of the linked list node above (LLNode) and the linked list on the right (LinkedList), what is the time complexity of the print_backward function assuming n is the total number of elements in the linked list?

print_backward runs in $O(n^2)$.

The while loop has $O(n)$ and inside the while loop, self.access has $O(n)$ complexity.

Thus, the worst case complexity is $O(n^2)$

```
class LinkedList:
  def __init__(self):
    self.head = LLNode(0)
    self.size = 0

  def access(self, k):
    temp = self.head
    while k >= 0 and temp.next != None:
      temp = temp.next
      k -= 1
    return temp

  def insert(self, new_element, k=0):
    temp = self.access(k-1)
    new_node = LLNode(new_element, temp.next)
    temp.next = new_node
    self.size += 1
    return

  def delete(self, k=0):
    temp = self.access(k-1)
    to_delete = temp.next
    temp.next = to_delete.next
    self.size -= 1
    del to_delete
    return

  def print_forward(self):
    temp = self.head.next
    out = ""
    while temp != None:
      out = out + str(temp.data) + " -> "
      temp = temp.next
    print(out)
    return

  def print_backward(self):
    n = self.size
    out = ""
    while n > 0:
      n -= 1
      temp = self.access(n)
      out = out + str(temp.data) + " <- "
    print(out)
    return
```

b. (6 *pts*) Modify the linked list implementation above such that `print_backward` runs in $O(n)$ time or better. A HackerRank challenge is set-up for testing your implementation:

www.hackerrank.com/contests/eee-121-2s2223-hkrb/challenges/from-the-back

You may attach a link to your HackerRank submission or you may add your code directly in this area:

https://www.hackerrank.com/contests/eee-121-2s2223-hkrb/challenges/from-the-back/submissions/code/1359849682

c.1. (1 *pts*) Briefly describe what was done to accomplish the time complexity required in (b) and
c.2. (3 *pts*) what are the drawbacks of your approach (i.e. memory considerations, logical complexity, etc…).

c.1.) To do print_backwards faster, we need a sentinel node at the end such that the traversal is straightforward. (Doubly linked list)

c.2.) The drawback is that more memory is consumed. The extra memory is needed to track the previous link and for the tail sentinel.

d. (4 *pts*) Consider the implementation of insertion sort that uses a modified binary search algorithm on the right. Assuming `arr` is implemented as a ***dynamic list*** and `binary_search(arr[i], arr[:i])` runs in $O(\lg i)$ time, is the worst-case time complexity of this algorithm $O(n \lg n)$? Why or why not?

```
def insertion_sort(arr):
  for i in range(1, len(arr)):
    # store element at index i
    t = arr[i]

    # remove element at index i
    arr.delete(i)

    # get index of t in
    # sorted sub-array arr[:i]
    j = binary_search(t, arr[:i])

    # insert arr[i] at index j
    arr.insert(j, arr[i])
```

Yes, the worst case time complexity is nlgn similar to just an ordinary list. There really is not much difference between the static list and dynamic list aside from the memory /size allocation.