Example of Mid-Exam

Problem 1: Given the classes, DoublyNode and DoublyLinkedList, representing a doubly node and a doubly linked list of numbers, respectively:

A. (0.40) In the DoublyLinkedList class, add a new method, **isSorted(self),** that returns true if the list is sorted (in ascending order) and false otherwise. *Examples: 1, 2, 5 -> true; 1, 2, 3, 3, 4, 8, 8, 9 -> true; 1, 9, 2, 0, 1 -> false.* What is the complexity of this method (0.10)? (Just explain i.e., there is no need to calculate T(n)!!!). Discuss about the best and worst cases.

Solution:

```
def isSorted(self):
      if self.isEmpty():
            return True

      prev=self.head
      node=prev.next

      while node:
            if prev.element>node.element:
                  return False
            prev=node
            node=node.next

      return True
```

The time complexity is O(n), because in the worst case you need to visit all the nodes to know if the list is sorted. The best case is when the list is empty or when the list only has one element. If the list has two or more elements, the best case is when the first element is greater than the second one. This is a best case because the loop only runs once. When the list is sorted in ascending order is the worst case because the method must traverse all the nodes.

B. (0.80) Add a **removeSortDuplicates**(**self**) method that removes any duplicate number from the invoking object list. You must use *isSorted* method to check if the list is sorted or not. If the list is not sorted, the method just shows the following message “the list is not sorted!!!” and then ends. Otherwise, the method removes the duplicate elements. Your solution should be the most efficient possible one algorithm. *Example: if the linked list is: 1, 3, 3, 5, 7, 7, 7, 10, 11, 11, after the method, the list should be: 1, 3, 5, 7, 10, 11.* What is the complexity of this method ? (Just explain i.e., there is no need to calculate T(n)!!!). Discuss about the best and worst cases.

Note: You cannot use any of the methods of the DoublyLinkedList class (except isEmpty or isSorted). It is not allowed to use any other data structure such as Python List, stacks or queues, among others.

Solution:

```
def removeSortDuplicates(self):
      if self.isEmpty():
            return
      if not self.isSorted():
            print('The list is not sorted')
            return


      prev=self.head
      node=prev.next

      while node:
            if prev.element==node.element:
                  if node==self.tail:
                        prev.next=None
                        self.tail=prev
                  else:
                        prev.next=node.next
                        node.next.prev=prev

                  self.size-=1
            else:
                  prev=node

            node=node.next
```

The time complexity is O(n), because in the worst case the method must traverse all the nodes to find the duplicate values and remove them. The best case is when the list is empty or when the list is not sorted. If the list is sorted and not empty, the best case is when the list only contains an element.

Otherwise, the method must always visit all the nodes until to reach the last one.

Problem 2 – **Recursive functions**

1. (0.25) Implement a recursive method, **count3**(n), which takes a a non-negative integer, n, and returns the count of the occurrences of the number 3 in n. *Example: if n=31315 then the method returns 2. Hint: You can use the integer division and modulus operators to solve this problem. Please, note that if n=31315, n%10=5, n/10=3131.*

Solution:

```
def count3(n):
        if n<0:
            return 0

        if n==3:
            return 1

        if n<10:
            return 0

        result=0
        if n%10==3:
            result=1

        return result + count3(n//10)
```

2. (0.25) Write a recursive method, **largestEven(l)**, that takes a Python list of integers and returns the largest even integer of the array. *Example: Input : A = {1, 4, 3, -5, -4, 8, 6}; Output : 8*

Solution:

```
def largestEven(l):
  if len(l)==0:
      return None
```

```
if len(l)==1 and l[0]%2==0:
    return l[0]

if len(l)==1 and l[0]%2!=0:
    return None

#recursive case

if l[0]%2!=0:
    return largestEven(l[1:])

largest=largestEven(l[1:])
if largest is None:
    return l[0]
else:
    return max(l[0],largest)
```