Name:_____ Roll number:_____ Semester and Section:___

NATIONAL INSTITUTE OF TECHNOLOGY CALICUT
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
MID SEMESTER TEST-I WINTER 2018

CS 2005 DATA STRUCTURES AND ALGORITHMS
PART-B

QUESTION NUMBER	I	II	III	IV	V	TOTAL MARKS
MARKS OBTAINED						

Q1. a. Write an O(n) algorithm ***cumulate(p)***, which takes the pointer to a singly linked list of nodes containing integer data, changes the data at each node to the cumulative sum of data in nodes upto that node, and returns the start node.[2 Marks]

Eg. Given **p: p→5→5→6→3→2→null**, ***cumulate(p)*** would return the address contained in p itself, but the contents would be get changed as follows: **p→5→10→16→19→21→null**

Assume node structure


```
cumulate(p)
    q = p
    sum = 0
    while (q != null)
        q.data = q.data + sum
        sum = q.data
        q = q.next
    return (p)
```

b. Prove the correctness of your algorithm[1 Mark]

(Use the other side of this sheet to answer this question)

By the termination condition, the required tasks are done (see next page)

Loop Invariant

At the start of a while iteration, the following are true

(i) p is the start node address

(ii) sum contains the cumulative sum of data in all nodes before node q.

(iii) Node q and subsequent nodes have not had their data replaced by cumulative sum upto that ~~point~~ node

(iii) Nodes before q had their data replaced by the cumulative sum upto ~~the~~ node.

Initialization (i) as p is not changed
(ii) sum is 0 & there are no nodes before q
(iii) Not touched.
(iv) Trivially true

Maintenance : If ~~true~~ (i) to (iv) are true before the while iteration, after the iteration they are still true because

(i) p not changed
(ii) sum gets q:data added and sum would reflect the new cumulative sum when q is advanced.
(iii) q is advanced to q.next so those nodes are still not changed
(iv) Nodes before old q were changed from assumption, old q is changed now & q is advanced.

Termination: (i) p is not changed - start.
(ii) q is null → end of the list → sum contains cumulative sum.
(iii) ~~All node~~ Trivially true (iv) All nodes are changed.

Name:_____ Roll number:_____ Semester and Section:___

Q2. The problem is to split a given singly linked list to three equal parts, without counting the number of elements. Given a pointer p to a linked list with 3n number of nodes, where n>=1, and n is unknown, write an algorithm ***Trifurcate(p)***that takes such a pointer p , and returns a pointer to three element array, R, such that R[1] contains a pointer p itself, but p's linked list carries only the first n elements, R[2] contains a pointer, say q, to the list containing the next n elements, and R[3] contains a pointer, say r, to the list containing the last n elements.[3 marks]. Only one traversal to the end of the list is permitted. ***Please note that solutions that involve counting the nodes will not get any credits at all in this question.***

Eg.If P→4→2→5→6→8→9→1→5→3→null, then Trifurcate (p) returns R where

R[1]→4→2→5→null

R[2]→6→8→9→null

R[3]→1→5→3→null

```
Trifurcate(p)
    R[1] = p
    q = p.next
    r = p.next.next
    while (r.next != null)
        r = r.next.next.next
        q = q.next.next
        p = p.next
    R[2] = p.next
    R[3] = q.next
    return(R)
```

Name:_____ Roll number:_____ Semester and Section:___

Q3. Read the given algorithm, whose input is an array A[1..n] of distinct integers

```
Unknown(A)
    s=false;
    while(s==false)
            s=true;
            i=1
            while(i<=A.length-1)
                  if(A[i]>A[i+1])
                        swap(A[i],A[i+1])
                        s=false
                   i=i+2
             i=2
             while(i<=A.length-1)
                  if(A[i]>A[i+1])
                        swap(A[i],A[i+1])
                        s=false
                   i=i+2
```

a. What is the output/result of execution of the algorithm on A.[1 mark]

The array A gets sorted. That is, the elements a_1' to a_n' of A would be permuted version of original contents such that $a_1' < a_2' < a_3' \ldots a_{n-1}' < a_n'$

b. Derive a tight bound on its best case complexity. [1 mark]

In the best case s = false within the loops. statements would not get executed, because $A[i] < A[i+1]$ for all i's. This happens when the array was soiled. So there would be only one iteration of outer while

$T(n) = c + 2n.$ $\therefore T(n) \leq (c+2)n \quad \forall n \geq 1$

$T(n) \geq 2n \quad \forall n \geq 1$

$\Rightarrow T(n) = \Theta(n)$

c. Give a suitable loop invariant for the loops and prove the correctness of the algorithm. [2 marks]

The L.I is proved.
Termination proof was not asked.

LOOP INVARIANTS

Outer While Loop

$$(\text{sorted} == \text{FALSE}) \vee \left((\text{sorted} == \text{TRUE}) \wedge \left[\bigwedge_{i=1}^{n-1} A[i] < A[i+1]\right]\right)$$

Inner first while loop

for the beginning of i^{th} iteration :

(i) $\forall j \; (j \text{ is odd} \Rightarrow A[j] < A[j+1])$ $\quad j = 1 \text{ to } 2(i-1)$

AND

(ii) $(\text{sorted} == \text{true}) \Rightarrow$ for all odd indices j, $A[j]$ had been $< A[j+1]$ $\quad j = 1 \text{ to } 2(i-1)$

Inner second While loop

for the beginning of i^{th} iteration

(i) $\forall j \; (j \text{ is even} \Rightarrow A[j] < A[j+1])$ $\quad j = 1 \text{ to } 2(i-1)$

AND

(ii) $(\text{sorted} = \text{true}) \Rightarrow$ for all odd indices j, and for all even indices j in the range 1 to $2(i-1)$ $A[j]$ had been $< A[j+1]$.

Initialization: sorted is made false (outer)
Trivial (inner loops) + correctness of inner while 1.

Maintenance:

inner loops: → because of exchange.
and sorted remains true if no exchange.

Outer loop → Either sorted is false or if true, the inner loop's ~~for~~ termination conditions.

Termination

Terminates when SORTED is true, then sorted condition is fulfilled — Outer.
Inner loops — sorted = false or sorted = true and all elements had been in sorted order (no exchange was required)