Roll Number:____________________ Number of Pages: 02

Thapar Institute of Engineering and Technology, Patiala

Department of Computer Science and Engineering

MID SEMESTER EXAMINATION

B. E. (Second Year): Sem-I (2022/23)	Course Code: UCS301 Course Name: Data Structures	September 26, 2022 Monday, 17:30 Hrs – 19:30 Hrs
Duration: 2 Hours	Maximum Marks: 50	Weightage: 25 Marks

Name of Faculty: Shreelekha Pandey, Rajesh Mehta, Rajendra Roul, Sumana Maiti, Vaibhav Pandey, Manisha Kaushal, Diwakar Tripathi

Note: Attempt subparts of a question in sequence at one place. Assume missing data, if any, suitably.

Q1. (a) For the code in **Table 1**, compute the time complexity by finding the frequency (i.e. the number of times a statement executes) of each statement. The entire code is in a tabular form, where each row represents a statement of the code (in the column ***Statement***). Fill in the column ***Frequency***. **(5)**

(b) Show that $\mathbf{3n + 7 = \Theta(n)}$ by finding the most appropriate values of $\mathbf{c_1}$, $\mathbf{c_2}$, and $\mathbf{n_0}$, where $\mathbf{c_1}$ and $\mathbf{c_2}$ are two positive constants and $\mathbf{n_0}$ represents the initial value of $\mathbf{n}$ (i.e., size of the input). Clearly show the steps for deciding values of $\mathbf{c_1}$, $\mathbf{c_2}$, and $\mathbf{n_0}$. **(5)**

Table 1. Code for Q1.

	Statement	Frequency
1.	`printf("Beginning of the code");`	
2.	`int a = 0, n;`	
3.	`for(int i = 0; i < n; i++){`	
4.	`  printf("%d", i);`	
5.	`  for( int j = n; j > i; j--){`	
6.	`    a = a + i + j;}`	
7.	`  printf("%d", a);}`	
8.	`printf("End of the code");`	

Table 2. Precedence of operators for Q4.

Operator	Precedence
^	Highest
*	↓
+, −	Lowest

Q2. (a) Consider a three-dimensional array **P** having **10** two-dimensional arrays of size **20×50**. The respective lower bounds on row, column, and blocks are **−2**, **−5**, and **6**. The **base address is 300** and each element takes **4 bytes** of memory. Assuming **P** is stored in a row major order; determine the address of an element stored in the **9th** two-dimensional array at **16th** row and **32th** column. **(4)**

(b) Write two efficient pseudocodes or algorithms to compute transpose of an **M×N** sparse matrix having **L** non-zero elements. Also discuss their time complexities. **(6)**

Q3. There are two single linked lists, **LL1** and **LL2**, having nodes arranged in non-decreasing order of their values. Write an efficient algorithm or pseudocode to create a single linked list, **LL3**, with the common nodes in **(10)**

LL1 and **LL2**. The sequence of nodes in **LL3** should also be in non-decreasing order of their values. Also, creation of new node is not allowed, i.e. **LL3** is formed only by repositioning of the existing nodes in **LL1** and **LL2**.

Example: *LL1: 1 → 3 → 4 → 5 → 6 → 7*
LL2: 2 → 4 → 6 → 8
After the execution, of the proposed algorithm
LL1: 1 → 3 → 5 → 7
LL1: 2 → 8
LL3: 4 → 4 → 6 → 6

Q4. (a) Convert the given infix expression into an equivalent postfix expression using stacks. Show contents of the stack at each intermediate step. Use the precedence table as shown in **Table 2**. **(5)**

$$A + B * (C\text{^}D - E)\text{^}F + G * H - I$$

(b) Let S be a stack of size $n >= 1$. Starting with the empty stack, suppose the first n natural numbers are pushed in sequence, and then popped. Assume that **Push()** and **Pop()** operation take X seconds each, and Y seconds elapse between the end of one such stack operation and the start of the next operation. For any natural number m (where $1 \leq m \leq n$), the stack-life of m is defined as the time elapsed from the end of **Push(m)** to the start of the **Pop()** that removes m from S. **(5)**

In terms of three known parameters n, X and Y, calculate

(i) Stack-life of an element pushed the first, i.e. m = 1.
(ii) Stack-life of an element pushed the last, i.e. m = n.
(iii) ***Average*** stack-life (ASL) of an element.

Q5. (a) Let **Q** be a circular queue having space for n elements. Initially, $front = rear = -1$. A student mistakenly understood that **enqueue(Q)** inserts an element at *front* and **dequeue(Q)** deletes an element from *rear*. In such a scenario, write the pseudocode written by the student to **(6)**

(i) Check that **Q** is full.
(ii) Check that **Q** is empty.
(iii) Insert an element in **Q** if it is empty.
(iv) Insert an element in **Q** if it has only one element.
(v) Delete an element from **Q** if it has only one element.
(vi) Delete an element from **Q**.

(b) Let **Q** be a simple queue with *n* integers. **isEmpty(Q)** returns *true* if **Q** is empty, *false* otherwise. **dequeue(Q)** deletes an integer at the *front* of **Q** and returns its value. **enqueue(Q, i)** inserts an integer *i* at the *rear* of **Q**. The task is to reverse the order of the integers in **Q**. Propose an efficient algorithm or pseudocode for the same using constant extra space. Show it's execution on a **Q** with **4** integers: (*front*) **4 6 1 9** (*rear*). **(4)**

--**ALL THE BEST**--