Register Number | | | | | | | | | |

Sri Sivasubramaniya Nadar College of Engineering, Kalavakkam – 603 110

(An Autonomous Institution, Affiliated to Anna University, Chennai)

Department of Computer Science and Engineering

Continuous Assessment Test – I

Question Paper

Degree & Branch	BE (CSE)				**Semester**	III
Subject Code & Name	UCS1302 Data Structures				**Regulation:**	**2018**
Academic Year	2021-2021 ODD	**Batch**	2020-2024	**Date**	**18.10.2021**	**FN**
Time: 90 Minutes	**Answer All Questions**				**Maximum: 50 Marks**	

Part – A (6×2 = 12 Marks)

<KL2>	1. Given the following functions for the running time of different algorithms: $f1(n) = n \log_{10} n + 1000n$ $f2(n) = 10n + 20$ $f3(n) = n(n - 1)/2$ Infer whether each function is of the time complexity log n, n, n log n or n^2 by writing Yes/No in each cell of the table given below.	<CO1>
<KL2>	2. Outline the main advantages of the linked-list based implementation and array-based implementation of the List ADT.	<CO1>
<KL2>	3. Show a C function that takes an array of integers and creates a new array that contains all the integers from input array with all negative integers at the end of the array. Example: Input : a[ ] = {2,3,-8.7,-9,6,-4.-2,0} Output: b[ ] = {2,3,7,6,0,-8,-9,-4,-2}	<CO1>
<KL2>	4. Outline the algorithm for *deleteFirst(*CLL)* that deletes the first node in the circular linked list where CLL is the pointer to the last node.	<CO1>
<KL2>	5. Outline the algorithm for inserting an element into the stack implemented with an array.	<CO2>
<KL2>	6. Why do you need circular arrays to implement a Queue? Explain with an example.	<CO2>

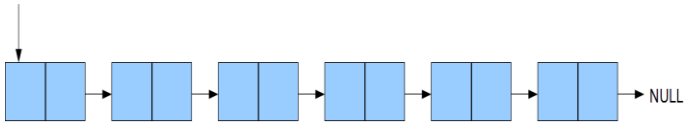
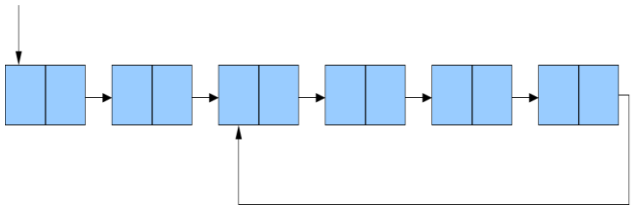
	O(log n)	O(n)	O(n log n)	$O(n^2)$
f1(n)				
f2(n)				
f3(n)				

Part – B (3×6 = 18 Marks)

<KL3>	7. Build a function in C to print all the *loosers* in the array. An element is a looser if it is lesser than all the elements to its right side. The rightmost element in the array is always looser. ***Example***: Input: a[ ] = {10, 8, 4, 3, 15, 28, 23} Output: loosers are 3, 15 and 23	<CO1>
<KL2>	8. Define polynomial ADT and outline the algorithm for difference of two polynomials. Make use of a linked list. Trace your algorithm with the following example. Support it with diagrams. Input: poly1 = -8x6+7x5+4x2 poly2= 3x6-9x2-8 Output: poly3 = -11 x6+7x5+13x2-8	<CO1>
<KL2>	9. Outline a function in C, *EvalExprn(str *exp)* that takes a postfix expression, evaluates it and returns the value. Assume that Stack ADT is available.	<CO2>

	Input : 546*+ Output: 29	

Part – C (2×10 = 20 Marks)

<KL3>	10. Develop a C function *Merge(Input1[], Input2[], Result[])* which takes two arrays of integers in sorted order and store the merged sorted data in a third array. Input: Input1[ ] = {4,7,11,45} Input2[ ] = {2,8,9,16,21,71} Output: Result[ ] = {2,4,7,8,9,11,16,21,45,71}	<CO1>
	(OR)	
<KL3>	11. Suppose you have a pointer to the head of singly linked list. Normally, each node in the list only has a pointer to the next element, and the last node's pointer is NULL. Unfortunately, your list might have been corrupted by a bug in somebody else's code, so that the last node has a pointer back to some other node in the list instead.  A Normal Linked List  Corrupted Linked List Develop an iterative algorithm that determines whether the linked list is corrupted or not. Your algorithm must not modify the list. Write the algorithm in $O(n^2)$ time and evolve another version in O(n).	<CO1>
<KL3>	12. Develop an algorithm to read a string of characters and determine whether it is a palindrome. A palindrome is a sequence of characters that reads the same forward and backward. Eg. Madam. The character "." is end of string. Write "Yes" if the input is a palindrome; else write "No". Use a Stack and a Queue data structure. Write all the required sub algorithms of Stack and Queue operations.	<CO2>
	(OR)	
<KL3>	13. Construct a Data Structure *SpecialStack* that supports operations such as *pushS(), popS(), isEmptyS(), isFullS()* and an additional operation *getMin()* which should return minimum element from the *SpecialStack*. *SpecialStack* uses a two traditional stacks; first one to stack the given elements and second one to keep track of the minimum of the elements currently in the first stack. It means that while pushing an element in the first stack, push the minimum in the second stack; while popping out from the first stack, pop out from second stack too. All these operations of *SpecialStack* must be O(1). Build *SpecialStack* ADT using a standard *Stack* ADT and write the algorithms for its operations.	<CO2>
