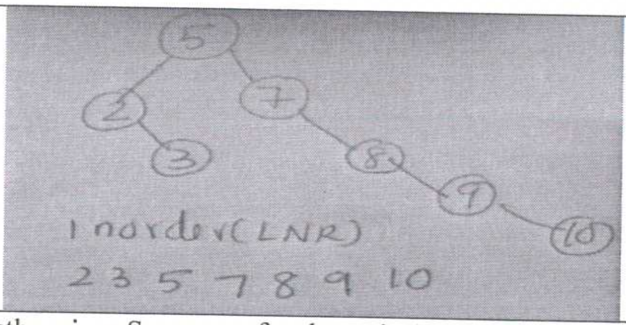
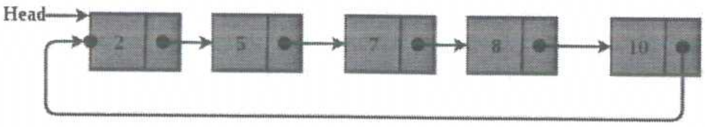
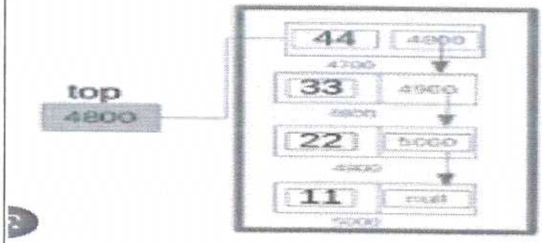
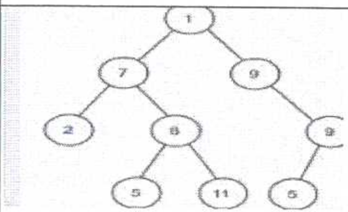
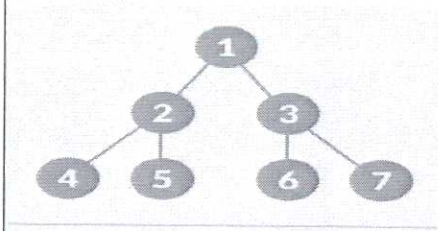
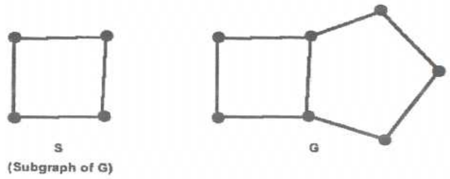
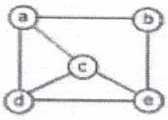
Scoring Indicators

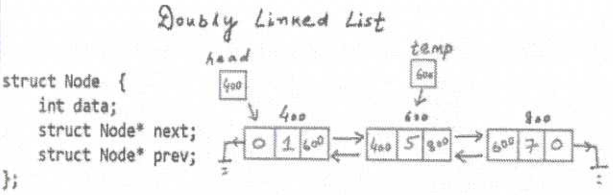
COURSE NAME: DATASTRUCTURES

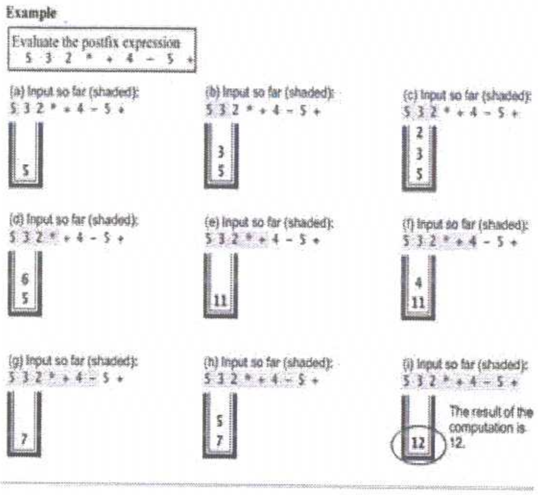
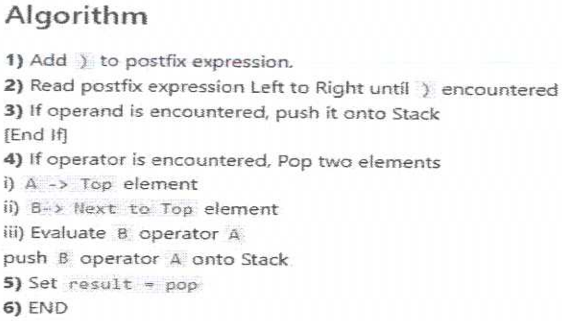
COURSE CODE: 4133 **QID: 2103230215**

Q No	Scoring Indicators	Split score	Sub Total	Total score
	PART A			**9**
I. 1	Datastructure is a logical arrangement of data so that it can be accessed and updated efficiently		1	
I. 2	Rear end		1	
I. 3	Circular queue		1	
I. 4	Singly linked list		1	
I. 5	Circular linked list		1	
I. 6	 5 2 7 3 8 9 10 Inorder(LNR) 2 3 5 7 8 9 10	1	1	
I. 7	Path :unique Sequence of nodes and edges Path length :path length of the tree is the number of edges in the path	0.5+0.5	1	
I. 8	Parallel edges		1	
I. 9	Complete graph		1	
				24
II. 1	10,20,15,22,27 (draw queue operations)(1.5)+answer(1.5)	1.5+1.5	3	
II. 2	Dequeue or Double Ended Queue is **a type of queue in which insertion and removal of elements can either be performed from the front or the rear.** Operations are • Add an element at the rear end(insert right) • Delete an element from the rear end(delete right) • Add an element at the front end (insert left) • Delete an element from the front end(delete left)		3	
II. 3	Circular Linked list *The **circular linked list** is a linked list where all nodes are connected to form a circle. In a circular linked list, the first node and the last*		3	

	node are connected to each other which forms a circle. There is no NULL at the end.  			
II. 4	 top 4800 44 4700 33 4900 22 5000 11 null	3	3	
II. 5	 (i)Degree :number of edges incident on a node or number of children nodes degree(node 7)=2 (ii)height of tree: The height of a tree (also known as depth) is the maximum distance between the root node of the tree and the leaf node of the tree. It can also be defined as the number of edges from the root node to the leaf node. Height=3 Drawing(1)+degree(1)+height(1)	1+1+1	3	
II. 6	Perfect binary tree A perfect binary tree is a type of binary tree in which every internal node has exactly two child nodes and all the leaf nodes are at the same level.  Perfect Binary Tree Diagram(1.5)+explanation(1.5)	1.5+ 1.5	3	
II. 7	Subgraph A graph whose vertices and edges are subsets of another graph.	1.5+1.5	3	

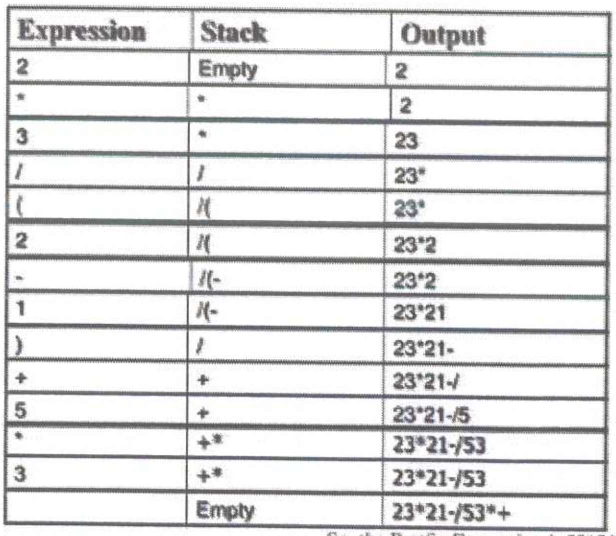
	 Adjacent vertices n a graph, two vertices are said to be adjacent, if there is an edge between the two vertices.  Vertex a is adjacent to c and vertex c is adjacent to a			
II. 8	Different types of graph Directed graph,undirected graph,simple,complete,cyclic,acyclic,bipartite,complete bipartite,connected,disconnected and regular graph (Any 6-1/2 mark each)	0.5x6	3	
II.9	(i)traversal (ii)overflow condition:when rear>max-1 Under flow condition :when front=-1 and rear=-1 When front>rear		3	

II.10	
	struct Node {
	    int data;
	    struct Node* next;
	    struct Node* prev;
	};
	 Doubly Linked List
	head
	400
	temp
	600
	400
	600
	800
	0
	1
	600
	400
	5
	800
	600
	7
	0
	a doubly linked list is a linked data structure that consists of a set of sequentially linked records called nodes. Each node contains three fields: two link fields and one data field.next points to next field and prev points to previous node
			3	

	PART C			42
III. 1	 Example Evaluate the postfix expression 5 3 2 * + 4 - 5 + (a) Input so far (shaded): 5 3 2 * + 4 - 5 + 5 (b) Input so far (shaded): 5 3 2 * + 4 - 5 + 3 5 (c) Input so far (shaded): 5 3 2 * + 4 - 5 + 2 3 5 (d) Input so far (shaded): 5 3 2 * + 4 - 5 + 6 5 (e) Input so far (shaded): 5 3 2 * + 4 - 5 + 11 (f) Input so far (shaded): 5 3 2 * + 4 - 5 + 4 11 (g) Input so far (shaded): 5 3 2 * + 4 - 5 + 7 (h) Input so far (shaded): 5 3 2 * + 4 - 5 + 5 7 (i) Input so far (shaded): 5 3 2 * + 4 - 5 + 12 The result of the computation is 12.  **Algorithm** **1)** Add) to postfix expression. **2)** Read postfix expression Left to Right until) encountered **3)** If operand is encountered, push it onto Stack [End If] **4)** If operator is encountered, Pop two elements i) A -> Top element ii) B-> Next to Top element iii) Evaluate B operator A push B operator A onto Stack **5)** Set result = pop **6)** END	3.5+3.5	7	7

III. 2

Suppose we want to convert 2*3/(2-1)+5*3 into Postfix form,



Expression	Stack	Output
2	Empty	2
*	*	2
3	*	23
/	/	23*
(	/(	23*
2	/(	23*2
-	/(-	23*2
1	/(-	23*21
)	/	23*21-
+	+	23*21-/
5	+	23*21-/5
*	+*	23*21-/53
3	+*	23*21-/53
	Empty	23*21-/53*+

algorithm

- Initialize an empty stack and an empty output string.
- Iterate through the characters of the infix expression. For each character:
  - If the character is a digit, append it to the output string.
  - If the character is an operator, compare its precedence with the operator on the top of the stack. If the operator on the top of the stack has higher or equal precedence, pop operators from the stack and append them to the output string. Then push the current operator onto the stack.
  - If the character is an open parenthesis, push it onto the stack.
  - If the character is a close parenthesis, pop operators from the stack and append them to the output string until an open parenthesis is encountered. Discard the open and close parenthesis.
- After iterating through the characters, pop any remaining operators from the stack and append them to the output string.

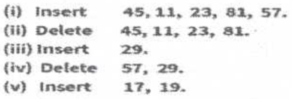
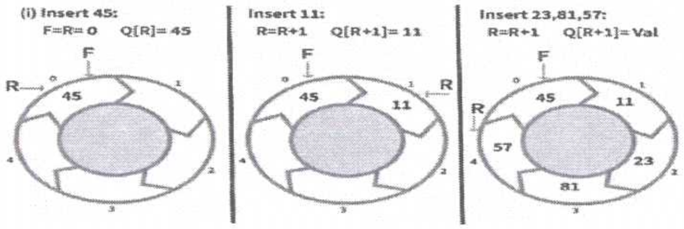
The output string now contains the postfix notation of the infix expression

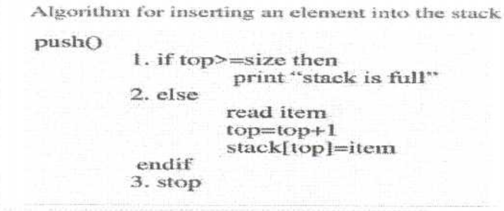
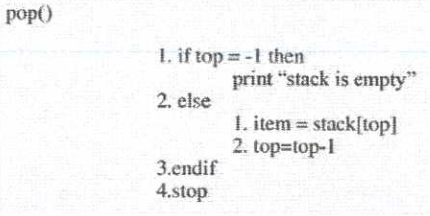
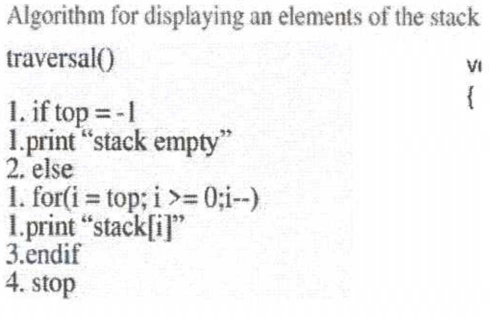
3.5+3.5 7 7

III. 3

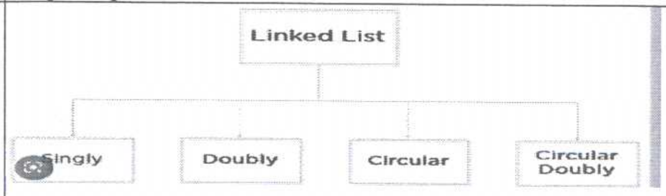
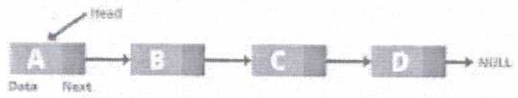
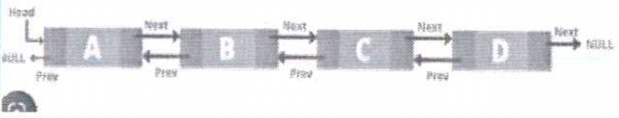
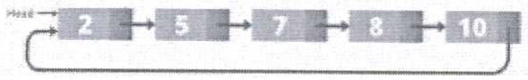
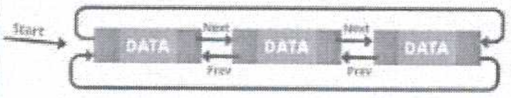
- Circular Queue is **a special version of queue where the last element of the queue is connected to the first element of the queue forming a circle**.
- The operations are performed based on FIFO (First In First Out) principle. It is also called 'Ring Buffer‘
- Insertion happens at rear end (rear=(rear+1)%MAX)
- Deletion happens at front end (front=(front+1)%MAX)
- **CQ Overflow condition**:

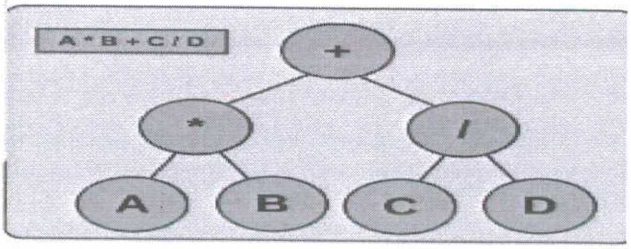
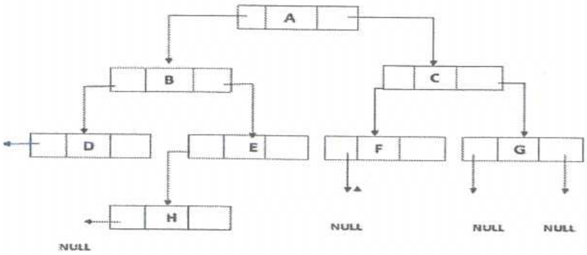
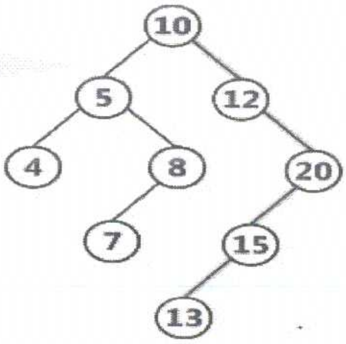
3.5+3.5 7 7

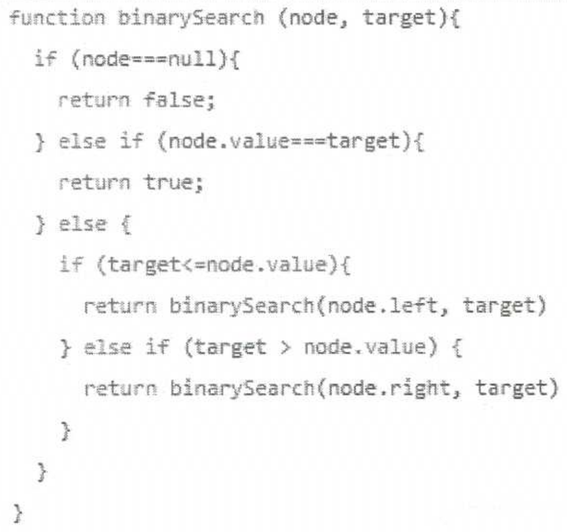
	if ((rear + 1) % MAX_SIZE == front) { printf("Queue overflow\n");
	• **CQ Underflow condition:**
	if (front == rear) { printf("Queue underflow\n");
	
	(i) Insert 45, 11, 23, 81, 57.
	(ii) Delete 45, 11, 23, 81.
	(iii) Insert 29.
	(iv) Delete 57, 29.
	(v) Insert 17, 19.
	
	 (i) Insert 45:
	F=R= 0 Q[R]= 45
	F
	R
	0
	1
	2
	3
	4
	45
	Insert 11:
	R=R+1 Q[R+1]= 11
	F
	R
	45
	11
	Insert 23,81,57:
	R=R+1 Q[R+1]= Val
	F
	R
	45
	11
	23
	81
	57			

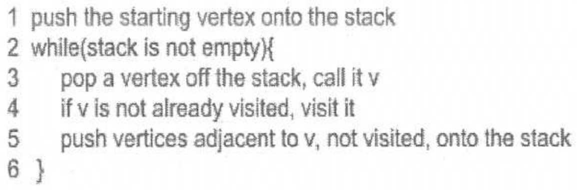
III. 4	Stack is a Linear data structure which can store homogeneous data where insertion (push) and deletion(pop) can occur in one end called **top** . →It follows LIFO(Last in first out) →There are 3 basic operations in stack :push(insert an element) ,pop(delete an element) , traverse(process all elements)  Algorithm for inserting an element into the stack push() 1. if top>=size then print "stack is full" 2. else read item top=top+1 stack[top]=item endif 3. stop  pop() 1. if top = -1 then print "stack is empty" 2. else 1. item = stack[top] 2. top=top-1 3.endif 4.stop  Algorithm for displaying an elements of the stack traversal() Vi { 1. if top = -1 1.print "stack empty" 2. else 1. for(i = top; i >= 0;i--) 1.print "stack[i]" 3.endif 4. stop	1+2+2+ 2	7	7

III. 5	**Delete an element**
	temp=head
	while(temp!=NULL)
	  if(temp→data=search-ele)
	   prev→next=temp→next
	   free(temp)
	   temp=prev→next
	prev=temp
	temp=temp→next
	Traversal
	temp=head
	while(temp!=NULL)
	print temp→data
	temp-temp→next
		3.5+3.5	7	7

III. 6	 Linked List Singly Doubly Circular Circular Doubly Singly Linked List  Doubly Linked List  Circular Linked List  Doubly Circular Linked List  	1+2+2+2	7	7
III. 7	**Expression trees** are used to express a mathematical expression in the form of a binary tree. Expression trees are binary trees in which each internal (non-leaf) node is an operator and each leaf node is an	3.5+3.5	7	7

	operand.  **A Threaded Binary Tree** is a binary tree in which every node that does not have a right child has a THREAD (in actual sense, a link) to its INORDER successor  A B C D E F G H NULL NULL NULL NULL			
III. 8		1+2+2	7	7

	
	function binarySearch (node, target){
	  if (node===null){
	    return false;
	  } else if (node.value===target){
	    return true;
	  } else {
	    if (target<=node.value){
	      return binarySearch(node.left, target)
	    } else if (target > node.value) {
	      return binarySearch(node.right, target)
	    }
	  }
	}			

III. 9	DFS
	
	push the starting vertex onto the stack
	while(stack is not empty){
	    pop a vertex off the stack, call it v
	    if v is not already visited, visit it
	    push vertices adjacent to v, not visited, onto the stack
	}
	Explanation(3.5)+example(3.5)
		3.5+3.5	7	7

III. 10	A graph can be represented using 3 data structures- **adjacency matrix, adjacency list and adjacency set.** In **adjacency list** representation of a graph, every vertex is represented as a node object. The node may either contain data or a reference to a linked list **Adjacency set** is quite similar to adjacency list except for the difference that instead of a linked list; a set of adjacent vertices is provided. Explanation(3.5)+diagrams(3.5)	3.5+3.5	7	7
III. 11	Binary search tree, expression tree and threaded binary tree. Explain any two binary trees(3.5+3.5)	3.5+3.5	7	7
III. 12	**Preorder traversal** **1.**visit the node 2.traverse in preorder in left recursively 3.Traverse in preorder in right recursively Algorithm(3.5)+example(3.5)	3.5+3.5	7	7