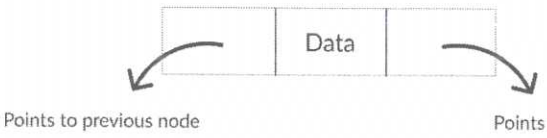
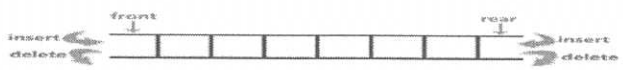
Apr-25

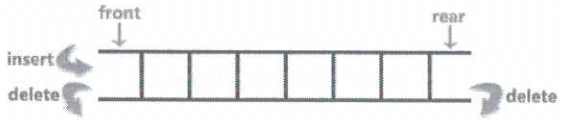
P-11

Scoring Indicators

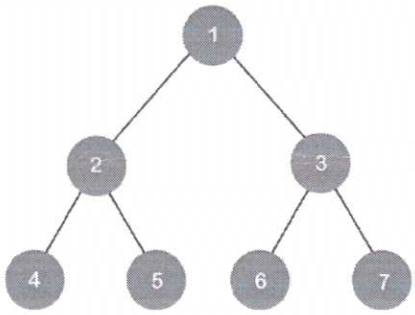
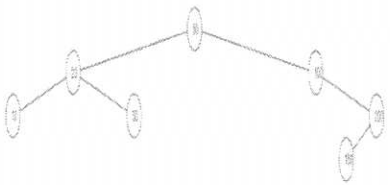
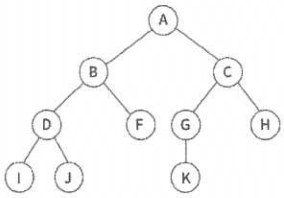
COURSE NAME : Data Structures

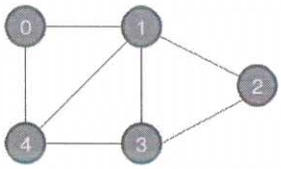
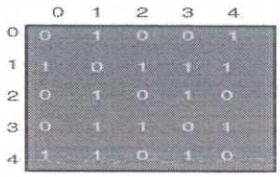
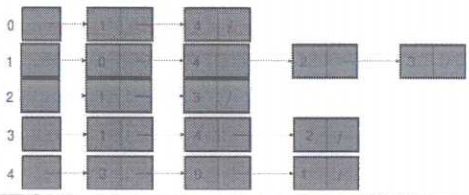
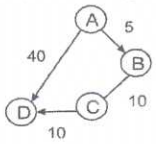
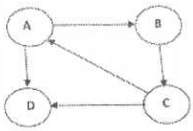
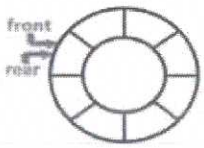
COURSE CODE :4133 **QID :2103230213**

Q No	Scoring Indicators	Split score	Sub Total	Total score
	PART A			**9**
I. 1	A data structure is basically a group of data elements that are put together under one name, and which defines a particular way of storing and organizing data in a computer so that it can be used efficiently.	**1**	1	
I. 2	1) LIFO - Stack 2) FIFO- Queue	½ +½	1	
I. 3		**1**	1	
I. 4	The linked list is a linear data structure that contains a sequence of elements such that each element links to its next element in the sequence. Each element in a linked list is called "Node".	**1**	1	9
I. 5	Full binary tree	1	1	
I. 6	HEIGHT	1	1	
I. 7	Graph is a collection of nodes and edges in which nodes are connected with edges A graph **G** is represented as **G = (V , E)**, where **V is a set of vertices** and **E is a set of edges.**	1	1	
I. 8	If a graph (digraph) does not have any cycle, then it is called an acyclic graph.	**1**	1	
I. 9	front.	1	1	
	PART B			24
II. 1	Deque stands for double ended queue. Here elements can be inserted or deleted at either end.  There are two types of Deque: 1. Input Restricted Queue. 2. Output Restricted Queue **Input Restricted Queue** This allows insertion only at one end of the array, but deletion allowed at both ends.	1 1	3	3

	 **Output Restricted Queue.** This allows insertion at both ends, but deletion allowed at only one end of the array.	1		
II. 2	Infix Expression We write expressions in **infix** notation, e.g. a - b + c, where operators are used **in**-between operands. Prefix Expression In this notation, operator is **prefix**ed to operands, i.e. operator is written ahead of operands. For example, +**ab**. This is equivalent to its infix notation **a + b**. Prefix notation is also known as **Polish Notation**. Postfix Expression This notation style is known as **Reverse Polish Notation**. In this notation style, the operator is **postfix**ed to the operands i.e., the operator is written after the operands. For example, **ab**+. This is equivalent to its infix notation **a + b**.	1 1 1	3	3
II. 3	Applications of stack 1. Evaluation of arithmetic expression 2. Implementation of Recursion 3. Backtracking (any three)	3 * 1	3	3

II. 4	void Delfrombeg()
	{
	struct node *temp;
	if(head == NULL)
	{
	printf("\nList is empty\n");
	}
	else
	{
	temp = head;
	printf("\nNode %d deleted from the beginning ...",temp->data);
	head = temp->next;
	free(temp);
	}
	}
		3	3	3

II. 5	**Types of Linked list** The following are the types of linked list: • Singly Linked list • Doubly Linked list • Circular Linked list • Doubly Circular Linked list (any three)	1 * 3	3	3
II. 6	**Perfect Binary Tree** A tree is a perfect binary tree if all the internal nodes have 2 children, and all the leaf nodes are at the same level.  **All the perfect binary trees are the complete binary trees as well as the full binary tree, but vice versa is not true.**	Explanation - 2 marks Example - 1 mark	3	3
II. 7		3	3	3
II. 8	A binary tree data structure is represented using two methods. Those methods are as follows... 1. Array Representation 2. Linked List Representation  1. Array Representation of Binary Tree In array representation of a binary tree, we use a one-dimensional array (1-D Array) to represent a binary tree. 	1.5 + 1.5	3	3

II. 9	Graph Representations Graph data structure is represented using the following representations... 1. Adjacency Matrix 2. Adjacency List  Adjacency matrix  Adjacency List 	1 1 1	1 2 	3
II. 10	1) Weighted Graph A graph is termed a weighted graph, if all the edges in it are labeled with some weights.  2) Directed Graph (Digraph) A digraph is called directed graph, if there is an directed edge from vi to vj indicated by arrow head. 	1 + 0.5 1 + 0..5	3	3
	PART C			42
III. 1	A circular queue is a linear data structure in which the operations are performed based on FIFO (First In First Out) principle and the last position is connected back to the first position to make a circle. Limitation of normal queue is overcomed by circular queue. Graphical representation of a circular queue is as follows... 	3 1	4	7

	Priority Queue It is a collection of elements where they are stored according to their priority levels. Each element is assigned a value, called the priority of the element. Insertion and removing elements from the queue is decided by the priority of the elements from the queue. An element with higher priority is processed first.Two elements of the same priority are processed on a first-come-first-served basis. Eg: VIP service., The basic examples of Priority Queue • Routing • Operating System Scheduler • Dijkstra's Shortest Path Algorithm	3	3	

III. 2	Queue is a linear data structure in which the insertion and deletion operations are performed at two different ends. In a queue data structure, adding and removing elements are performed at two different positions. The insertion is performed at one end and deletion is performed at another end. In a queue data structure, the insertion operation is performed at a position which is known as 'rear' and the deletion operation is performed at a position which is known as 'front'. In queue data structure, the insertion and deletion operations are performed based on FIFO (First In First Out) principle.
	Basic Operations
	• enqueue() – add (store) an item to the queue.
	• dequeue() – remove (access) an item from the queue.
	• peek() – Gets the element at the front of the queue without removing it.
	• isfull() – Checks if the queue is full.
	• isempty() – Checks if the queue is empty.
	int peek()
	{
	return queue[front];
	}
	bool isfull()
	{
	if(rear == MAXSIZE - 1)
	return true;
	else
	return false;
	}
	bool isempty()
	{
	if(front < 0 || front > rear)
	return true;
	else
	return false;
	}
	int enqueue(int data)
	{
	if(isfull())
	return 0;
	else
		7	7	7

```
        {
                rear = rear + 1;
                queue[rear] = data;
                return 1;
        }
}
int dequeue()
{
if(isempty())
return 0;
int data = queue[front];
front = front + 1;
return data;
}
```

III. 3

Insertion

In a single linked list, the insertion operation can be performed in three ways. They are as follows...

1. Inserting At Beginning of the list
2. Inserting At End of the list
3. Inserting At Specific location in the list

Listing - 1 mark — 1

Insertion at the beginning of the list

1. Create a new node , temp=new Node(x,ptr)
2. temp->next= head.
3. head=temp.

explanation - any 2 * 3 marks — 6

Inserting At End of the list

```
void insertatend(int data)
 {
    struct node *newNode;
    newNode = malloc(sizeof(struct node));
    newNode->data = data;
    newNode->next = NULL;
    struct node *temp = head;
    while(temp->next != NULL)
    {
       temp = temp->next;

    }
    temp->next = newNode;
 }
```

Inserting At Specific location in the list

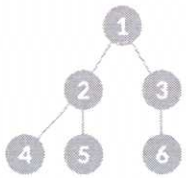
```
void insertatpos(int item)
{
   int position;
   printf("Enter the position to be inserted");
   scanf("%d",&position);
```

7

	struct node *newNode;
	newNode = malloc(sizeof(struct node));
	newNode->data = item;
	struct node* temp = head;
	for(int i=2; i < position; i++)
	{
	if(temp->next != NULL)
	{
	temp = temp->next;
	}
	}
	newNode->next = temp->next;
	temp->next = newNode;
	}			

III. 4	A stack is represented using nodes of a linked list. Each node consists of two parts: data and next(storing the address of the next node). The data part of each node contains the assigned value, and the next points to the node containing the next item in the stack. The top refers to the topmost node in the stack. Both the push() and pop() operations are carried out at the front/top of the linked list
	<u>Operations on a Stack</u>
	The following operations are performed on the stack...
	1. Push (To insert an element on to the stack)
	2. Pop (To delete an element from the stack)
	3. Display (To display elements of the stack)
	
	void push(int value) {
	struct Node *newNode;
	newNode = (struct Node *)malloc(sizeof(struct Node));
	newNode->data = value; // assign value to the node
	if (top == NULL) {
	newNode->next = NULL;
	} else {
	newNode->next = top; // Make the node as top
	}
	top = newNode; // top always points to the newly created node
	printf("Node is Inserted\n\n");
	}
	int pop() {
	if (top == NULL) {
	printf("\nStack Underflow\n");
	} else {
	struct Node *temp = top;
	int temp_data = top->data;
	top = top->next;
	free(temp);
	return temp_data;
	}
		Definition - 1 mark
		
		Explanation - 6 mark
			1
			
			6
				7

	}
	void display() {
	// Display the elements of the stack
	if (top == NULL) {
	printf("\nStack Underflow\n");
	} else {
	printf("The stack is \n");
	struct Node *temp = top;
	while (temp->next != NULL) {
	printf("%d--->", temp->data);
	temp = temp->next;
	}
	printf("%d--->NULL\n\n", temp->data);
	}
	}			

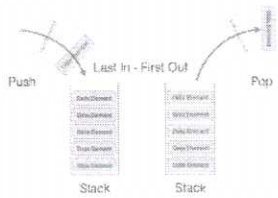
III. 5	 1. **Degree :-** Total number of edges connected to a vertex is said to be the degree of that vertex. 2. **Level**:- In a tree each step from top to bottom is called a Level and the Level count starts with '0' and incremented by one at each level (Step). 3. Leaf Node:- The nodes which do not have any child nodes are called leaf nodes 4. Sibling of a node:- The node which have same parent are called siblings.	Fig : -3 1 * 4 = 4	3 4	7

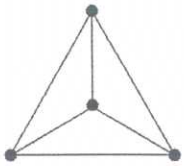
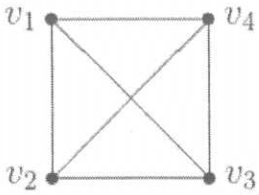
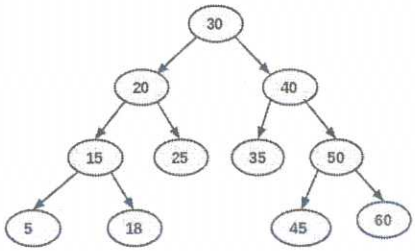
III. 6	Traversals
	Displaying (or) visiting order of nodes in a binary tree is called Binary Tree Traversal.
	There are three types of binary tree traversals.
	1. In - Order Traversal
	2. Pre - Order Traversal
	void inorder(struct node *root) {
	if (root != NULL) // checking if the root is not null
	{
	inorder(root -> left_child); // traversing left child
	printf(" %d ", root -> data); // printing data at root
	inorder(root -> right_child); // traversing right child
	}
	}
	void preorder(struct node *root) {
	if (root != NULL) // checking if the root is not null
	{
	printf(" %d ", root -> data); // printing data at root
	inorder(root -> left_child); // traversing left child
	inorder(root -> right_child); // traversing right child
	}
	}
		3.5 marks
		3.5 marks
			7	7

III. 7	DFS (Depth First Search) DFS traversal of a graph produces a **spanning tree** as final result. **Spanning Tree** is a graph without loops. We use **Stack data structure** with maximum size of total number of vertices in the graph to implement DFS traversal. We use the following steps to implement DFS traversal... • Step 1 - Define a Stack of size total number of vertices in the graph. • Step 2 - Select any vertex as **starting point** for traversal. Visit that vertex and push it onto the Stack. • Step 3 - Visit any one of the non-visited **adjacent** vertices of a vertex which is at the top of the stack and push it onto the stack. • Step 4 - Repeat step 3 until there is no new vertex to be visited from the vertex which is at the top of the stack. • Step 5 - When there is no new vertex to visit then use **backtracking** and pop one vertex from the stack. • Step 6 - Repeat steps 3, 4 and 5 until the stack becomes Empty. • Step 7 - When stack becomes Empty, then produce final spanning tree by removing unused edges from the graph **Backtracking** is coming back to the vertex from which we reached the current vertex.	4 + 3	7	7

III. 8	**Warshall's algorithm for all-pairs shortest path**
	The all pair shortest path algorithm, also known as Floyd-Warshall algorithm is used to find all pair shortest path problems from a given weighted graph. As a result of this algorithm, it will generate a matrix, which will represent the minimum distance from any node to all other nodes in the graph.
	At first the output matrix is the same as the given cost matrix of the graph. After that the output matrix will be updated with all vertices k as the intermediate vertex.
	The time complexity of this algorithm is O(V3), here V is the number of vertices in the graph.
	Algorithm
	floyd Warshall(cost)
	Input − The cost matrix of a given Graph.
	Output − Matrix to for shortest path between any vertex to any vertex.
	Begin
	for k := 0 to n, do
	for i := 0 to n, do
	for j := 0 to n, do
	if cost[i,k] + cost[k,j] < cost[i,j], then
	cost[i,j] := cost[i,k] + cost[k,j]
		7	7	7

	done done done display the current cost matrix End			
III. 9	(a) the postfix expression **6 2 + 8 4 / -** evaluates to **6**. (b) Postfix notation: 4 2 5 3 - + * Result: 16	3 4	7	7

III. 10	• A Stack is a linear data structure that follows the LIFO (Last-In-First-Out) principle.
	• Stack has one end. It contains only one pointer, pointing to the topmost element of the stack.
	• Whenever an element is added in the stack, it is added on the top of the stack, and the element can be deleted only from the stack.
	• A *stack can be defined as a container in which insertion and deletion can be done from the one end known as the top of the stack.*
	 
	
	void push(int data)
	{
	if(!isFull())
	{
	top = top + 1;
	stack[top] = data;
	}
	else
	{
	printf("Could not insert data, Stack is full.\n");
	}
	}
	int pop(int data)
	{
	if(!isempty())
	{
	data = stack[top];
	top = top - 1;
	return data;
	}
	else
	{
	printf("Could not retrieve data, Stack is empty.\n");
	}
	}			

III. 11	Complete Graph :-  Undirected Graph:-  	3.5 3.5	7	7
III. 12	 Pre order :- 30,20,15,5,18,25,40,35,50,45,60 Post order:- 5,18,15,25,20,35,45,60,50,40,30	3.5 * 2	7	7