USN | | | | | | | | | | |

Internal Assessment Test 2 – Jan 2026
Scheme and Solution

Sub:	Data Structures and Applications				Sub Code:	BCS304	Branch:	ISE	
Date:	07-01-2026	Duration:	90 min's	Max Marks:	50	Sem/Sec:	III / A,B,C	OBE	
	Answer any FIVE FULL Questions						MARKS	CO	RBT
1.a	Write C Functions for the following Doubly Linked list i. Inserting a node at the beginning of a DLL. ii. Deleting a node at the end of a DLL.						7	CO3	L3

Insert Front Function – 3.5 Marks
Delete End Function – 3.5 Marks

```
void insert_front() {
    EMPLOYEE node;
    node = create();
    if (start == NULL) {
        start = node;
    } else {
        node->rlink = start;
        start->llink = node;
        start = node;
    }
}
```

```
void delete_end() {
    EMPLOYEE temp, prev;
    temp = start;
    if (temp == NULL) {
        printf("\nList is Empty");
    } else if (temp->rlink == NULL) {
        printf("\nThe deleted employee ssn is %s", temp->ssn);
        free(temp);
        start = NULL;
    } else {
        while (temp->rlink != NULL) {
            prev = temp;
            temp = temp->rlink;
        }
        prev->rlink = NULL;
        printf("\nThe deleted employee ssn is %s", temp->ssn);
        free(temp);
    }
}
```

		MARKS	CO	RBT
1.b	Describe Linked list and list different types of Linked list.	3	CO3	L2

Definition – 1 mark
Types- 2 marks

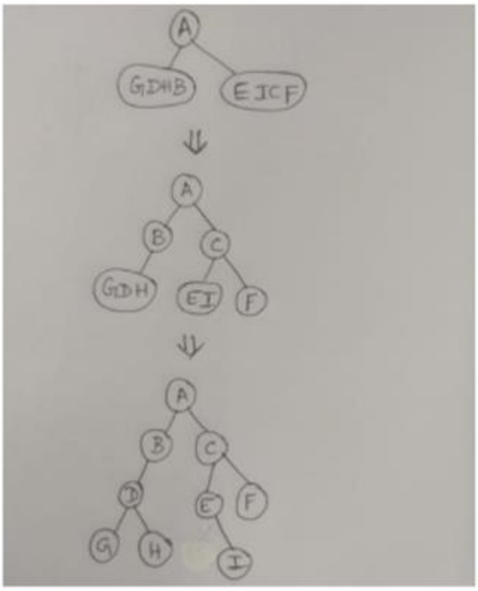
A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations. The elements in a linked list are linked using pointers. In simple words, a linked list consists of nodes where each node contains a data field and a reference (link) to the next node in the list.

Types of Linked Lists:

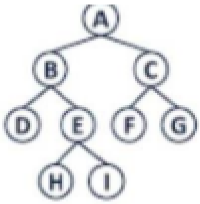
1. Singly Linked List

	2. Doubly Linked List 3. Singly Circular Linked List 4. Doubly Circular linked list			
2.a	Define sparse matrix. For the given sparse matrix, give the linked list representation: $A = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 4 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 \\ 8 & 0 & 0 & 1 \\ 0 & 0 & 6 & 0 \end{bmatrix}.$ **Definition – 2 marks** **Representation – 5 marks** **Sparse Matrix:** A sparse matrix is a matrix in which most of the elements are zero. This type of matrix is useful in scenarios where storage and computational efficiency are critical, as we can store only the non-zero elements rather than all elements, including the zeros. In linked list, each node has four fields. These four fields are defined as: • Row: Index of row, where non-zero element is located • Column: Index of column, where non-zero element is located • Value: Value of the non-zero element located at index – (row,column) • Next node: Address of the next node. Sparse Matrix (3-Tuple Representation): Row Column Value 1 1 2 2 1 4 2 4 3 4 1 8 4 4 1 5 3 6 **Linked List Representation (1-based indexing)** Head → (1,1,2) → (2,1,4) → (2,4,3) → (4,1,8) → (4,4,1) → (5,3,6) → NULL	7	CO3	L2
2.b	Construct a binary tree from the Post-order and In-order sequence given below In-order: GDHBAEICF Post-order: GHDBIEFCA **Constructing Binary tree – 3 Marks**	3	CO4	L2

Binary tree for the given inorder and postorder:



3.a	Write recursive C functions for in-order, pre-order and post-order traversals of a binary tree. Also, find all the traversals for the given tree.	7	CO4	L3



Recursive Function – Each 3 mark
Finding Traversal – 5 marks

Recursive c function:

INORDER:

```
void inorder(struct Node* root) {
    if (root == NULL)
      return;
   inorder (root->left);
   printf("%d ", root->data);
   inorder (root->right);
}
```

PRE-ORDER:

```
void preorder(struct Node* root) {
    if (root == NULL)
      return;
    printf("%d ", root->data);
    preorder (root->left);
    preorder (root->right);
}
```

POST-ORDER:

```
void postorder(struct Node* root) {
      if (root == NULL)
         return;
      postorder (root->left);
      postorder (root->right);
      printf("%d ", root->data);
```

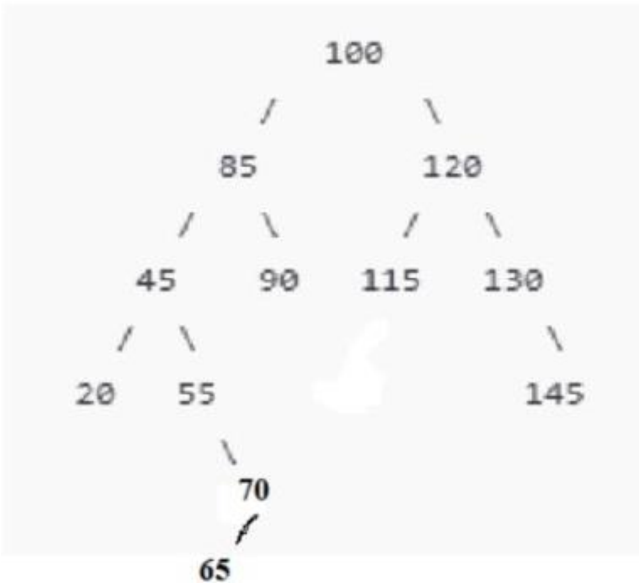
<table>
<tr><td></td><td>}

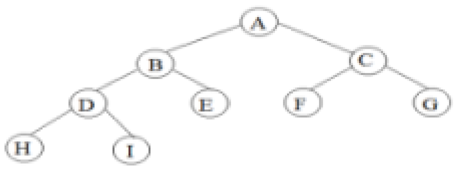
inorder - D B H E I A F C G
preorder - A B D E H I C F G
postorder - D H I E B F G C A</td><td></td><td></td><td></td></tr>
<tr><td>3.b</td><td>Define Binary Search tree. Construct a binary search tree (BST) for the following elements: 100, 85, 45, 55, 120, 20, 70, 90, 115, 65, 130, 145.

Definition and constructing Tree – 3 marks

A Binary Search Tree (BST) is a type of binary tree where:

1. Each node contains a key.
2. The key in the left subtree of a node is less than the key in the node.
3. The key in the right subtree of a node is greater than the key in the node.
4. Both the left and right subtrees must also be binary search trees.


</td><td>3</td><td>CO4</td><td>L3</td></tr>
<tr><td>4.a</td><td>Define the Threaded binary tree. Construct Threaded binary for the following tree.

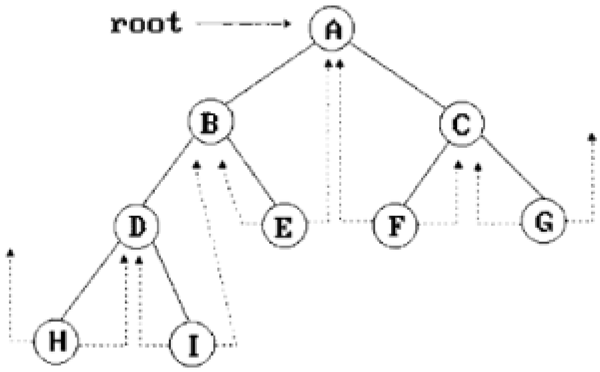


Definition and Construction – 3 marks

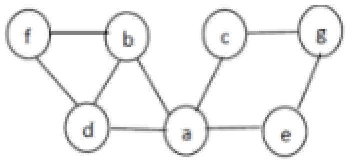
A Threaded Binary Tree is a binary tree where null pointers in leaf nodes are replaced with "threads" to allow in-order traversal without the use of recursion or a stack.

In a threaded binary tree, each node has an additional pointer called a thread, which pointsto either its in-order predecessor or successor. This allows us to efficiently traverse the tree without using recursion or a stack.</td><td>3</td><td>CO4</td><td>L2</td></tr>
</table>

Threaded Binary tree is:



4.b	Write the C Function for DFS and BFS. Find the traversal for the below graph.	7	CO4	L3



Function – 5 Marks
Traversal – 2 Marks

```
void bfs(int v) {
   int i, cur;
   bvisited[v] = 1;
   q[++rear] = v;

   while (front != rear) {
      cur = q[++front];
      for (i = 1; i <= n; i++) {
         if (a[cur][i] == 1 && bvisited[i] == 0) {
            q[++rear] = i;
            bvisited[i] = 1;
            printf("%d ", i);
         }
      }
   }
}
  void dfs(int v) {
     int i;
     dvisited[v] = 1;
     s[++top] = v;

     for (i = 1; i <= n; i++) {
        if (a[v][i] == 1 && dvisited[i] == 0) {
           printf("%d ", i);
           dfs(i);
        }
     }
  }
```

DFS: ABFDCGE
BFS: ABCDEFG

5.a	What is collision? What are the methods to resolve collision? Explain quadratic probing with an example.	7	CO5	L2

Definition – 2 marks
Types- 3 marks
Quadratic Probing – 2 marks

In hashing, a collision occurs when two or more different keys produce the same hash value and hence try to occupy the same index in a hash table.

Methods to Resolve Collision

1. Linear Probing
2. Quadratic Probing
3. Double Hashing
4. Separate Chaining

Quadratic Probing
In quadratic probing, when a collision occurs, the next position is found using a quadratic function of the probe number.

$$h_i(key) = (h(key) + i^2) mod m$$

Example of Quadratic Probing

Hash Table Size (m) = 10
Hash Function: h(key) = key % 10
Keys to insert: 22, 32, 42, 52

Step-by-step Insertion
1. Insert 22
   - h(22) = 2
   - Index 2 is empty → place 22
2. Insert 32
   - h(32) = 2 → collision
   - Try $(2 + 1^2)$ % 10 = 3
   - Index 3 is empty → place 32
3. Insert 42
   - h(42) = 2 → collision
   - Try $(2 + 1^2)$ % 10 = 3 → occupied
   - Try $(2 + 2^2)$ % 10 = 6
   - Index 6 is empty → place 42
4. Insert 52
   - h(52) = 2 → collision
   - $(2 + 1^2)$ = 3 → occupied
   - $(2 + 2^2)$ = 6 → occupied
   - $(2 + 3^2)$ = 11 % 10 = 1
   - Index 1 is empty → place 52

Final Hash Table

Index	Value
1	52
2	22
3	32
6	42

5.b	Define hashing. Write the different hashing functions and explain division method with examples.	3	CO5	L2

Definition of Hashing – 1 mark
Hash Function – 2 marks

Hashing is a technique used to convert a key into an index of a hash table using a hash function.

Hash Functions

1. Division Method
2. Mid-square Method
3. Folding Method
4. Multiplication Method

Division Method

Hash function:

h(k)=kmodmh(k) = k \bmod mh(k)=kmodm

Where:

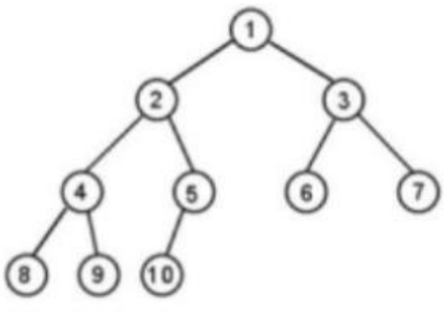
- *k* = key
- *m* = size of hash table

Example:
If `k = 52`, `m = 9`

h(52)=52mod9=7

So, **52 is stored at index 7**

6.a	Define the leftist tree. Give its declaration in C. Check whether the given binary tree is a leftist tree or not. Explain your answer.	5	CO5	L2



Definition - 2 Marks
Checking Leftist Tree – 3 Marks

A **Leftist Tree** is a binary tree that satisfies:

1. **Heap order property**
2. **Leftist property**:

npl(left)≥npl(right)npl(left) \ge npl(right)npl(left)≥npl(right)

Where **npl** = Null Path Length.

```
struct leftistNode
{
   int data;
   int npl;
   struct leftistNode *left;
   struct leftistNode *right;
};
```

Checking Leftist Tree (3 Marks)

Steps:

1. Compute `npl` of left and right child for each node
2. Check if `npl(left) ≥ npl(right)`
3. If condition holds for all nodes → **Leftist Tree**
4. Otherwise → **Not a Leftist Tree**

6.b	Explain Optimal BST with an example.	5	CO5	L2

Definition – 2 marks
Example – 3 marks

An Optimal Binary Search Tree (OBST) is a Binary Search Tree constructed in such a way that the expected search cost is minimum, based on the frequencies (probabilities) of searching the keys.

It is mainly used when:

- Search frequencies of keys are known in advance
- Some keys are accessed more frequently than others

Key	A	B	C
Frequency	3	2	4

```
A
 \
  B
   \
    C
```

Cost =

(3×1) + (2×2) + (4×3) = 3 + 4 + 12 = **19**

```
  B
 / \
A   C
```

Cost =

(2×1) + (3×2) + (4×2) = 2 + 6 + 8 = **16**

```
    C
   /
  B
 /
A
```

Cost =

(4×1) + (2×2) + (3×3) = 4 + 4 + 9 = **17**

The BST with minimum cost = 16 is Tree 2, so it is the Optimal BST.