Internal Assessment Test II –January 2026
ANSWER KEY

Sub:	Data Structures and Applications				Sub Code:	BCS304	Branch:	AIML/CSE AIML	
Date:	8.1.26	Duration:	90 min	Max Marks: 50	Sem/Sec :	III A, B, C		OBE	

	Answer any FIVE FULL Questions	**MARKS**	**CO**	**R B T**
1 a	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. Traverse using in-order, pre-order, and post-order traversal techniques.	6M	CO4	L3

BST definition and construction -3Marks

A **Binary Search Tree (BST)** is a binary tree in which:

- Each node has **at most two children**.
- The **left subtree** of a node contains values **less than** the node's key.
- The **right subtree** of a node contains values **greater than** the node's key

```
            100
          /     \
        85       120
       /  \     /   \
     45    90  115   130
    /  \              \
  20    55             145
         \
          70
         /
       65
```

Traversal-3x1=3marks

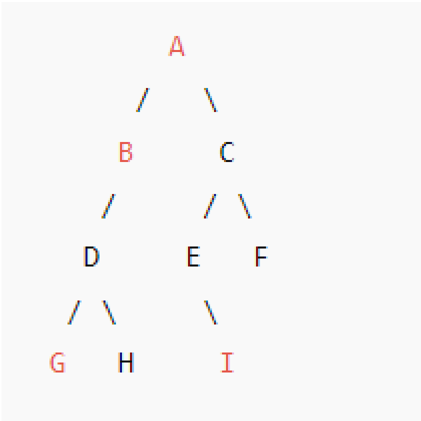
In-order:

20, 45, 55, 65, 70, 85, 90, 100, 115, 120, 130, 145

Pre-order:

100, 85, 45, 20, 55, 70, 65, 90, 120, 115, 130, 145

Post-order:

	20, 65, 70, 55, 45, 90, 85, 115, 145, 130, 120, 100			
1b	Construct a binary tree from the Inorder and Postorder sequence given below In-order: GDHBAEICF Post-order: GHDBIEFCA Construction of Tree-4Marks 	4M	CO4	L 3

2a	Write C function for Depth First Search(DFS) and show the graph traversal by taking an example.
	
	DFS
	Algorithm-3Marks
	
	void DFS(int v)
	
	{
	
	int i;
	
	visited[v] = 1;
	// Mark current
	vertex as visited
	
	printf("%d ", v);
	// Visit the vertex
	
	for (i = 1; i <= n; i++)
	
	{
	
	if (adj[v][i] == 1 && visited[i] == 0)
	
	{
	
	DFS(i);
	// Recursive call
	
	}
	
	}
		5M	CO4	L 2

	} Example-3Marks			
2b	Define graph. Explain Adjacency list and Adjacency matrix by taking an example. Graph-1Mark A **graph** is a mathematical structure used to represent relationships between objects. ● It consists of: 　○ **Vertices (nodes):** Represent objects. 　○ **Edges (links):** Represent connections or relationships between the vertices. ● Notation: **G = (V, E)** 　○ V = set of vertices 　○ E = set of edges Adjacency List-1 Mark **Adjacency List** ● Each vertex has a **list of vertices it is connected to** Example -1Mark Adjacency Matrix -1Mark **Adjacency Matrix** ● A **2D array** of size `n × n` where `n` = number of vertices ● Element `matrix[i][j] = 1` if there is an edge from vertex `i` to vertex `j` (0 otherwise) Example -1Mark	5M	CO4	L 2
3a	Define hashing. Explain different hashing functions with examples. Hashing-2Marks Hashing function- i)Division Method 2Marks ii)MidsquareHash Functio n2 Marks iii)Folding Method 2Marks iv)Digit analysis-1Mark v)Converting keys to integers-1Mark	10M	CO5	L 2

Hashing – 2 Marks

Definition:
Hashing is a technique used to map **keys** to **indices of a hash table** using a **hash function** to allow **fast insertion, deletion, and searching**.

Hashing Function – Methods

i) Division Method – 2 Marks

- Formula:

h(k)=kmod mh(k) = k \mod mh(k)=kmodm

- **k** = key, **m** = size of the hash table
- Example: Key = 123, Table size = 10 → 123mod 10=3123 \mod 10 = 3123mod10=3
- **Pros:** Simple and fast
- **Cons:** Table size should preferably be a prime number to reduce collisions

---

ii) Mid-Square Method – 2 Marks

- Steps:
  1. Square the key: k2k^2k2
  2. Take the middle digits as the hash value
- Example: Key = 123 → 1232=15129123^2 = 151291232=15129 → middle digits = 512 → index
- **Pros:** Good distribution
- **Cons:** Slightly more computation

---

iii) Folding Method – 2 Marks

- Steps:

1. Divide key into equal parts
2. Add the parts together
3. Apply modulo table size (optional)

- Example: Key = 123456, divide into 3 parts: 12, 34, 56 → 12+34+56 = 102 → index
- **Pros:** Handles large keys easily

---

iv) Digit Analysis – 1 Mark

- Use **specific digits of the key** as the hash value
- Example: Key = 45678 → use last 2 digits → 78 → index

---

v) Converting Keys to Integers – 1 Mark

- For **alphanumeric keys**, convert letters to numbers before hashing
- Example: Key = "ABC" → A=1, B=2, C=3 → 123 → use in hash function

4a	What is a leftist tree? Give the C declaration of it .Explain how meld operation is applied to two minimum leftist tree with the help of an example.	10M	CO5	L 2

Leftist Tree

Definition (2 Marks):
A **Leftist Tree** is a type of **priority queue** implemented as a binary tree where:

1. It satisfies the **heap property**: the key at each node is smaller (min-leftist) or larger (max-leftist) than the keys of its children.
2. It satisfies the **leftist property**: the **rank** (distance to nearest null node, also called **null path length, npl**) of the **left child** is always greater than or equal to the rank of the right child.

The purpose of the leftist property is to keep the tree **skewed to the left**, which ensures that **merging (meld) operations are efficient**.

C Declaration (2 Marks)

```
typedef struct LeftistNode {
    int key;                    // Value of the node
    int npl;                    // Null Path Length
    struct LeftistNode *left;   // Pointer to left
child
    struct LeftistNode *right;  // Pointer to right
child
} LeftistNode;

typedef LeftistNode* LeftistTree;  // Pointer to
root of the tree
```

- `npl` = null path length = shortest distance from node to a node without two children (null node).

Meld Operation (6 Marks)

Meld Operation:
The **meld** operation combines two leftist trees into one while maintaining **heap** and **leftist** properties.
Steps (for **min-leftist tree**):

1. Compare the roots of both trees. Make the root with the **smaller key** the new root.
2. Recursively **meld** the **right child** of this root with the other tree.
3. Swap **left and right children** if necessary to maintain the leftist property (`npl(left) ≥ npl(right)`).
4. Update the `npl` of the root.

5a	Write C Functions for the following, i) Inserting a node at the beginning of a Doubly linked list.	5M	CO3	L 2

C function-2.5Marks

```
Node* insertAtBeginning(Node* head, int newData) {
   // Step 1: Allocate memory for the new node
   Node* newNode = (Node*)malloc(sizeof(Node));
   if (!newNode) {
      printf("Memory allocation failed\n");
      return head;  // return existing head if malloc fails
   }

   // Step 2: Assign data to the new node
```

```
    newNode->data = newData;
    newNode->prev = NULL;    // New node becomes the first node
    newNode->next = head;    // Next points to the current head

    // Step 3: Update previous head's prev pointer if list is not empty
    if (head != NULL) {
        head->prev = newNode;
    }

    // Step 4: Return new node as the new head
    return newNode;
}
```

ii)Deleting a node at the end of the Doubly linked list.

C function-2.5Marks

```
Node* deleteAtEnd(Node* head) {
    // If the list is empty, nothing to delete
    if (head == NULL) {
        printf("List is empty.\n");
        return NULL;
    }

    // If the list has only one node
    if (head->next == NULL) {
        free(head);
        return NULL;  // List becomes empty
    }

    // Traverse to the last node
    Node* temp = head;
    while (temp->next != NULL) {
        temp = temp->next;
    }

    // Update previous node's next to NULL
    temp->prev->next = NULL;

    // Free the last node
    free(temp);

    // Return head of the list
    return head;
}
```

5b	Write C functions for the following, a)To search an item within a SLL(Singly Linked List)	5M	CO3	L 2

C function-2 .5Marks

```
Node* searchSLL(Node* head, int key) {
    Node* temp = head;

    // Traverse the list
    while (temp != NULL) {
```

```
        if (temp->data == key) {
            return temp;  // Item found, return pointer to the node
        }
        temp = temp->next;
    }

    // Item not found
    return NULL;
}
```

b) To concatenate two SLL.

C function-2.5Marks

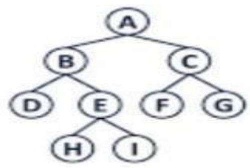
```
Node* concatenateSLL(Node* head1, Node* head2) {
    // If the first list is empty, return the second list
    if (head1 == NULL)
        return head2;

    // Traverse to the end of the first list
    Node* temp = head1;
    while (temp->next != NULL) {
        temp = temp->next;
    }

    // Link the last node of list1 to head of list2
    temp->next = head2;

    return head1;  // Return the head of the concatenated list
}
```

6a	Write recursive C functions for inorder, preorder and postorder traversals of a binary tree. Also, find all the traversals for the given tree.	6M	CO3	L 3



Each traversal -3 Marks

Preorder ABDEHICFG

INORDER: DBHEIAFCG

POSTODER:DHIEBFGCA

C function for each traversal -3Marks

```
/ Preorder traversal: Root -> Left -> Right
void preorder(Node* root) {
    if (root == NULL) return;
    printf("%c ", root->data);
    preorder(root->left);
    preorder(root->right);
}

// Inorder traversal: Left -> Root -> Right
void inorder(Node* root) {
    if (root == NULL) return;
    inorder(root->left);
```

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

```
    printf("%c ", root->data);
    inorder(root->right);
}

// Postorder traversal: Left -> Right -> Root
void postorder(Node* root) {
    if (root == NULL) return;
    postorder(root->left);
    postorder(root->right);
    printf("%c ", root->data);
}
```

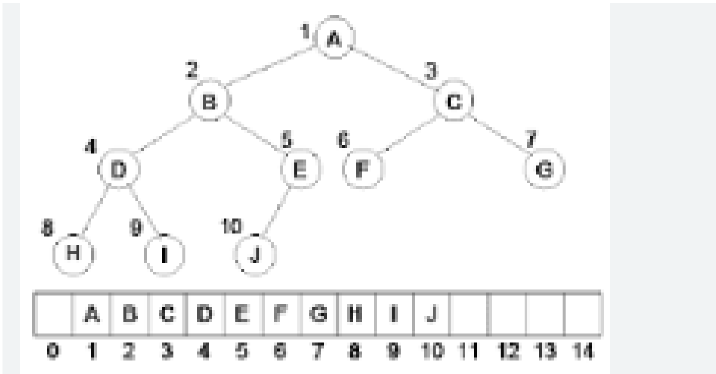
</td><td></td><td></td><td></td></tr>
<tr><td>6b</td><td>

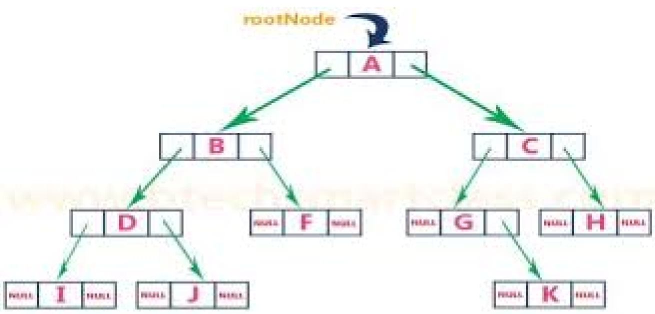
Define Binary tree. Explain the representation of a binary tree with a suitable example.

Definition (2 Marks)

A **binary tree** is a hierarchical data structure in which each node has at most **two children**, called **left child** and **right child**.

Array and Linked list representation of Binary tree-2Marks





</td><td>4M</td><td>CO3</td><td>L 2</td></tr>
</table>