**Student Information**

- **Name**: [ATUL KUMAR]
- **Student ID**: [590011190]
- **Branch**: [MCA]
- **Batch**: [B1]
- **Instructor**: [Dr. Sourbh Kumar]

**Lab Experiment: 09** **Batch: 1 & 2**
**Subject: Data Structures Lab** **MCA**
**Semester: 1st**

## 1st Assignment: Binary Tree Creation

**Using Arrays:**

- Represent a complete binary tree using an array.
  - Note that for a node at index i:
    - The left child is at 2 * i + 1
    - The right child is at 2 * i + 2

**Using Linked Lists:**

- Represent a binary tree where each node contains data and pointers to its left and right children.
- Include functions to create and insert nodes in the binary tree.

Solution:--

```
#include <stdio.h>

#define MAX_SIZE 100  // Maximum size of the array to store the binary tree

void insertInArray(int tree[], int *size, int value) {
if (*size < MAX_SIZE) {
tree[*size] = value;
(*size)++;
} else {
printf("Array is full, cannot insert more elements.\n");
}
}

void displayArrayTree(int tree[], int size) {
printf("Binary Tree represented as an array:\n");
for (int i = 0; i < size; i++) {
printf("%d ", tree[i]);
}
printf("\n");
}

int main() {
```

```
int tree[MAX_SIZE];
int size = 0;

// Insert elements into the binary tree
insertInArray(tree, &size, 1);  // Root node
insertInArray(tree, &size, 2);  // Left child of root
insertInArray(tree, &size, 3);  // Right child of root
insertInArray(tree, &size, 4);  // Left child of node at index 1
insertInArray(tree, &size, 5);  // Right child of node at index 1

// Display the array representation of the binary tree
displayArrayTree(tree, size);

return 0;
}
```

**Output:--**

```
Enter the size of the sorted array: 4
Enter 4 sorted elements of the array: 1
2
3
5
Enter the target value to search for: 2
Step 1: Searching between indexes 0 and 3
Target found at index 1.
```

## 2nd Assignment: Tree Traversal Methods

Implement the following traversal methods:

**In-order Traversal:**

- Traverse the left subtree, visit the root node, then traverse the right subtree.

**Pre-order Traversal:**

- Visit the root node, traverse the left subtree, then traverse the right subtree.

**Post-order Traversal:**

- Traverse the left subtree, traverse the right subtree, then visit the root node.

**Level-order Traversal:**

- Traverse the nodes level by level, starting from the root.

Implement each traversal function and test them with the binary tree created above.

**Soltuion:--**

```
#include <stdio.h>
#include <stdlib.h>

// Define the structure of a tree node
struct Node {
    int data;
    struct Node* left;
    struct Node* right;
};

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* node = (struct Node*)malloc(sizeof(struct
Node));
    node->data = data;
    node->left = NULL;
    node->right = NULL;
    return node;
}

// In-order Traversal: left, root, right
void inOrderTraversal(struct Node* node) {
    if (node == NULL) return;
    inOrderTraversal(node->left);
    printf("%d ", node->data);
    inOrderTraversal(node->right);
}

// Pre-order Traversal: root, left, right
void preOrderTraversal(struct Node* node) {
    if (node == NULL) return;
    printf("%d ", node->data);
    preOrderTraversal(node->left);
    preOrderTraversal(node->right);
}

// Post-order Traversal: left, right, root
void postOrderTraversal(struct Node* node) {
    if (node == NULL) return;
    postOrderTraversal(node->left);
    postOrderTraversal(node->right);
    printf("%d ", node->data);
}

// Level-order Traversal: level by level from root
void levelOrderTraversal(struct Node* root) {
    if (root == NULL) return;
```

```
    struct Node* queue[100]; // Simple queue for BFS, size is
limited for demonstration
    int front = 0, rear = 0;

    queue[rear++] = root;

    while (front < rear) {
        struct Node* current = queue[front++];

        printf("%d ", current->data);

        if (current->left != NULL)
            queue[rear++] = current->left;
        if (current->right != NULL)
            queue[rear++] = current->right;
    }
}

// Main function to test the traversal methods
int main() {
    struct Node* root = createNode(1);
    root->left = createNode(2);
    root->right = createNode(3);
    root->left->left = createNode(4);
    root->left->right = createNode(5);
    root->right->left = createNode(6);
    root->right->right = createNode(7);

    printf("In-order Traversal: ");
    inOrderTraversal(root);
    printf("\n");

    printf("Pre-order Traversal: ");
    preOrderTraversal(root);
    printf("\n");

    printf("Post-order Traversal: ");
    postOrderTraversal(root);
    printf("\n");

    printf("Level-order Traversal: ");
    levelOrderTraversal(root);
    printf("\n");

    return 0;
}
```

**Output:--**

```
In-order Traversal: 4 2 5 1 6 3 7
Pre-order Traversal: 1 2 4 5 3 6 7
Post-order Traversal: 4 5 2 6 7 3 1
Level-order Traversal: 1 2 3 4 5 6 7
 *  Terminal will be reused by tasks, press any key to close it.
```