Lab Experiment: 09

Student Detail:

• **Name:** Prashant Joshi

• **Student ID:** 590010879

• **Branch:** MCA

• **Batch:** B1

• **Instructor:** Dr. Sourbh Kumar

Implement the following tasks in C. Use appropriate data structures (array or linked list) to create the binary tree and demonstrate traversal methods and heap sorting.

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");
}
```

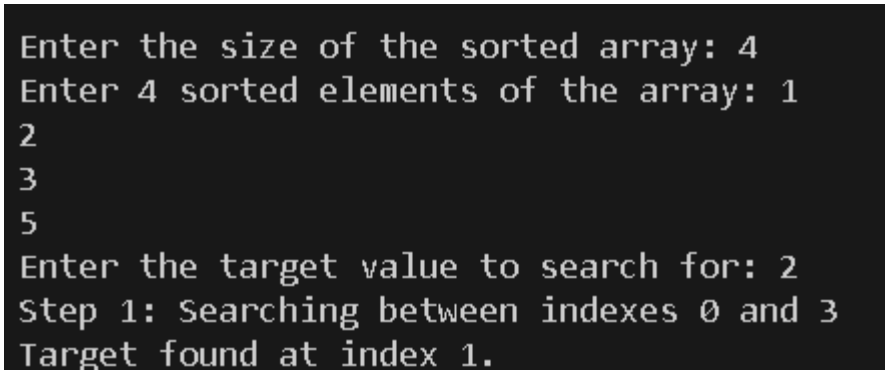
```
int main() {
int tree[MAX_SIZE];
int size = 0;

// 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:



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.

Solution:

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

// Definition of a node in the binary tree
struct TreeNode {
int data;
struct TreeNode* left;
struct TreeNode* right;
};

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

```
// Traversal functions
void inOrderTraversal(struct TreeNode* root) {
if (root != NULL) {
inOrderTraversal(root->left);
printf("%d ", root->data);
inOrderTraversal(root->right);
}
}

void preOrderTraversal(struct TreeNode* root) {
if (root != NULL) {
printf("%d ", root->data);
preOrderTraversal(root->left);
preOrderTraversal(root->right);
}
}

void postOrderTraversal(struct TreeNode* root) {
if (root != NULL) {
postOrderTraversal(root->left);
postOrderTraversal(root->right);
printf("%d ", root->data);
}
}

// Level-order Traversal (Breadth-first traversal)
void levelOrderTraversal(struct TreeNode* root) {
if (root == NULL) return;

struct TreeNode* queue[100];
int front = 0, rear = 0;
```

```
queue[rear++] = root;

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

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

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

// Insert helper functions
void insertLeft(struct TreeNode* parent, int value) {
parent->left = createNode(value);
}

void insertRight(struct TreeNode* parent, int value) {
parent->right = createNode(value);
}

int main() {
// Create the binary tree
struct TreeNode* root = createNode(1);
insertLeft(root, 2);
insertRight(root, 3);
insertLeft(root->left, 4);
insertRight(root->left, 5);

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 3
Pre-order Traversal: 1 2 4 5 3
Post-order Traversal: 4 5 2 3 1
Level-order Traversal: 1 2 3 4 5
```