8a) Write a program

a) To construct a binary Search tree.

b) To traverse the tree using all the methods i.e., in-order,

preorder and post order

To display the elements in the tree.

```
#include <stdio.h>

#include <stdlib.h>

// Structure for a node of the binary search tree struct Node {
  int data;
  struct Node* left; struct Node* right;
};

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

// Function to insert a new node with given key in BST struct Node*
insert(struct Node* root, int value) {
  // If the tree is empty, return a new node
  if (root == NULL) return createNode(value);

  // Otherwise, recur down the tree if (value < root->data)
    root->left = insert(root->left, value); else if (value > root->data)
```

```
    root->right = insert(root->right, value);

  // Return the (unchanged) node pointer return root;

}

// Function to perform inorder traversal of BST void inorder(struct Node*
root) {
  if (root != NULL) { inorder(root->left); printf("%d ", root->data);
    inorder(root->right);
  }
}

// Function to perform preorder traversal of BST void preorder(struct
Node* root) {
  if (root != NULL) { printf("%d ", root->data); preorder(root->left);
    preorder(root->right);
  }
}

// Function to perform postorder traversal of BST void postorder(struct
Node* root) {
  if (root != NULL) { postorder(root->left); postorder(root->right);
    printf("%d ", root->data);
  }
}

// Function to display elements in the tree void display(struct Node*
```

```
root) {


   printf("Inorder traversal: "); inorder(root); printf("\nPreorder traversal:
   "); preorder(root); printf("\nPostorder traversal: "); postorder(root);
}
int main() {
   struct Node* root = NULL;
   int elements[] = {50, 30, 70, 20, 40, 60, 80}; // Example elements
   // Construct the binary search tree
   for (int i = 0; i < sizeof(elements) / sizeof(elements[0]); i++) { root =
     insert(root, elements[i]);
   }
   // Display the elements in the tree using different traversal methods
   display(root);return 0;
}
```

```
Inorder traversal: 20 30 40 50 60 70 80
Preorder traversal: 50 30 20 40 70 60 80
Postorder traversal: 20 40 30 60 80 70 50
```

Q8b) You are given two binary trees root1 and root2.

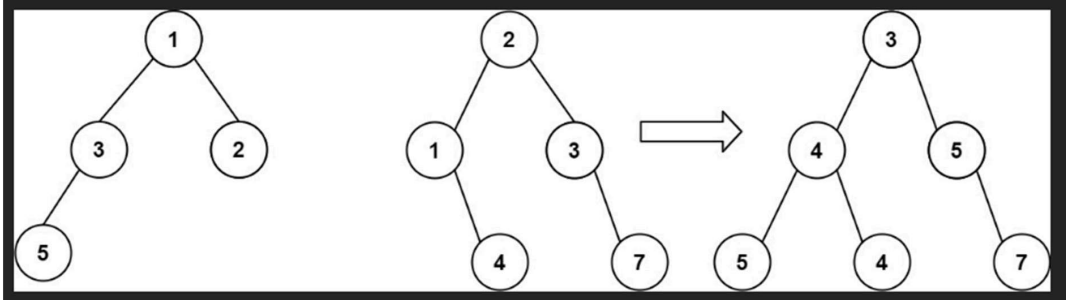
Imagine that when you put one of them to cover the other, some nodes of the two trees are overlapped while the others are not. You need to merge the two trees into a new binary tree. The merge rule is that if two nodes overlap, then sum node values up as the new value of the merged node. Otherwise, the NOT null node will be used as the node of the new tree.

Return the merged tree.

```
#include <stdlib.h>

// Definition for a binary tree node.

struct TreeNode* mergeTrees(struct TreeNode* root1, struct TreeNode*
  root2) { if (root1 == NULL) return root2;
  if (root2 == NULL) return root1;

  // Merge the current nodes
  struct TreeNode* merged = (struct TreeNode*)malloc(sizeof(struct
  TreeNode)); merged->val = root1->val + root2->val;

  // Recur for left subtree
  merged->left = mergeTrees(root1->left, root2->left);

  // Recur for right subtree
  merged->right = mergeTrees(root1->right, root2->right);
```

```
        return merged;
        }

}
```

**Example 1:**




```
Input: root1 = [1,3,2,5], root2 = [2,1,3,null,4,null,7]
Output: [3,4,5,5,4,null,7]
```

**Example 2:**

```
Input: root1 = [1], root2 = [1,2]
Output: [2,2]
```