1.Write a c program code for 2-3 tree
50,90,20,10,30,40,70,60,80,120,150,100,110,130,140,160

Code:

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

// Structure for 2-3 tree node
struct Node {
    int data1, data2;
    struct Node *left, *middle, *right;
};

// Function to create a new node
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct
Node));
    newNode->data1 = data;
    newNode->data2 = -1;
    newNode->left = newNode->middle = newNode->right = NULL;
    return newNode;
}

// Function to insert a value into the 2-3 tree
struct Node* insert(struct Node* root, int data) {
    if (root == NULL) {
        return createNode(data);
    }
    if (root->data2 == -1) {
        if (data < root->data1) {
            root->data2 = root->data1;
            root->data1 = data;
        } else {
            root->data2 = data;
        }
```

```
    } else if (data < root->data1) {
        root->left = insert(root->left, data);
    } else if (data > root->data2) {
        root->right = insert(root->right, data);
    } else {
        root->middle = insert(root->middle, data);
    }
    return root;
}

// Function to print the 2-3 tree in inorder traversal
void inorderTraversal(struct Node* root) {
    if (root != NULL) {
        inorderTraversal(root->left);
        printf("%d ", root->data1);
        if (root->data2 != -1) {
            printf("%d ", root->data2);
        }
        inorderTraversal(root->middle);
        inorderTraversal(root->right);
    }
}

int main() {
    int values[] = {50, 90, 20, 10, 30, 40, 70, 60, 80, 120, 150, 100, 110,
130, 140, 160};
    struct Node* root = NULL;

    for (int i = 0; i < sizeof(values) / sizeof(values[0]); i++) {
        root = insert(root, values[i]);
    }

    printf("Inorder traversal of the 2-3 tree: ");
```

```
    inorderTraversal(root);

    return 0;
}
```

Output: Inorder traversal of the 2-3 tree: 10 20 30 40 50 90 60 70 80 100 110 120 150 130 140 160

2.Write a c program code for 2-3-4 tree

30,60,22,10,17,24,2629,48,40,41,52,70,62,65,80,72,90,81,85,95

Code:

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

#define MAX_KEYS 3
#define MIN_KEYS (MAX_KEYS / 2)
#define MAX_CHILDREN (MAX_KEYS + 1)

// Node structure for 2-3-4 tree
typedef struct Node {
    int keys[MAX_KEYS];
    struct Node* children[MAX_CHILDREN];
    int numKeys;
    int isLeaf;
} Node;

// Function prototypes
Node* createNode(int isLeaf);
void insertNonFull(Node* node, int key);
void splitChild(Node* parent, int index, Node* child);
void insert(Node** root, int key);
void printTree(Node* root, int level);
void freeTree(Node* root);
```

```
int main() {
    int keys[] = {30, 60, 22, 10, 17, 24, 26, 29, 48, 40, 41, 52, 70, 62, 65,
80, 72, 90, 81, 85, 95};
    int n = sizeof(keys) / sizeof(keys[0]);

    Node* root = NULL;

    for (int i = 0; i < n; i++) {
        insert(&root, keys[i]);
    }

    printf("2-3-4 Tree:\n");
    printTree(root, 0);

    freeTree(root);
    return 0;
}

// Create a new node
Node* createNode(int isLeaf) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->isLeaf = isLeaf;
    newNode->numKeys = 0;
    for (int i = 0; i < MAX_CHILDREN; i++) {
        newNode->children[i] = NULL;
    }
    return newNode;
}

// Split the child of a node
void splitChild(Node* parent, int index, Node* child) {
    Node* newChild = createNode(child->isLeaf);
```

```
    parent->numKeys++;

    // Shift children of parent to make space for new child
    for (int i = parent->numKeys; i > index; i--) {
        parent->children[i + 1] = parent->children[i];
    }
    parent->children[index + 1] = newChild;

    // Move the middle key from the child to the parent
    parent->keys[index] = child->keys[MIN_KEYS];

    // Move keys and children from child to newChild
    newChild->numKeys = MIN_KEYS;
    for (int i = 0; i < MIN_KEYS; i++) {
        newChild->keys[i] = child->keys[i + MIN_KEYS + 1];
    }
    if (!child->isLeaf) {
        for (int i = 0; i <= MIN_KEYS; i++) {
            newChild->children[i] = child->children[i + MIN_KEYS + 1];
        }
    }
    child->numKeys = MIN_KEYS;
}

// Insert a key into a non-full node
void insertNonFull(Node* node, int key) {
    int i = node->numKeys - 1;
    if (node->isLeaf) {
        while (i >= 0 && key < node->keys[i]) {
            node->keys[i + 1] = node->keys[i];
            i--;
        }
        node->keys[i + 1] = key;
```

```
        node->numKeys++;
    } else {
        while (i >= 0 && key < node->keys[i]) {
            i--;
        }
        i++;
        if (node->children[i]->numKeys == MAX_KEYS) {
            splitChild(node, i, node->children[i]);
            if (key > node->keys[i]) {
                i++;
            }
        }
        insertNonFull(node->children[i], key);
    }
}

// Insert a key into the 2-3-4 tree
void insert(Node** root, int key) {
    Node* r = *root;
    if (r == NULL) {
        *root = createNode(1);
        (*root)->keys[0] = key;
        (*root)->numKeys = 1;
    } else {
        if (r->numKeys == MAX_KEYS) {
            Node* s = createNode(0);
            *root = s;
            s->children[0] = r;
            splitChild(s, 0, r);
            insertNonFull(s, key);
        } else {
            insertNonFull(r, key);
        }
```

```
    }
}

// Print the tree
void printTree(Node* root, int level) {
    if (root != NULL) {
        for (int i = 0; i < root->numKeys; i++) {
            printTree(root->children[i], level + 1);
            for (int j = 0; j < level; j++) {
                printf("    ");
            }
            printf("%d\n", root->keys[i]);
        }
        printTree(root->children[root->numKeys], level + 1);
    }
}

// Free the tree memory
void freeTree(Node* root) {
    if (root != NULL) {
        for (int i = 0; i <= root->numKeys; i++) {
            freeTree(root->children[i]);
        }
        free(root);
    }
}
```

Output: 2-3-4 Tree:

```
30
  24
    10
    17
    22
```

41
  26
  29
52
  65
    62
    70
    72
  80
    81
    85
    90
    95