2-3 tree:

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

typedef struct Node {
    int keys[2]; // Node can have up to 2 keys (for 2-3 tree)
    struct Node *children[3]; // Node can have up to 3 children
    int num_keys; // Number of keys in the node
    int is_leaf;  // Boolean to check if the node is a leaf
} Node;

Node* create_node(int is_leaf) {
    Node *node = (Node*)malloc(sizeof(Node));
    node->num_keys = 0;
    node->is_leaf = is_leaf;
    for (int i = 0; i < 3; i++)
        node->children[i] = NULL;
    return node;
}

void insert_non_full(Node *root, int key) {
    Node *node = root;
    // If root is a leaf node
    if (node->is_leaf) {
        int i = node->num_keys - 1;
        while (i >= 0 && key < node->keys[i]) {
            node->keys[i + 1] = node->keys[i];
            i--;
        }
        node->keys[i + 1] = key;
        node->num_keys++;
    } else {
        // If root is not a leaf node
        int i = node->num_keys - 1;
        while (i >= 0 && key < node->keys[i]) {
            i--;
        }
        i++;
        if (node->children[i]->num_keys == 2) {
            // Split child node here (simplified version, real implementation needed)
        }
        insert_non_full(node->children[i], key);
    }
}

void traverse(Node *root) {
    if (root == NULL) return;
    for (int i = 0; i < root->num_keys; i++) {
        if (!root->is_leaf)
            traverse(root->children[i]);
        printf("%d ", root->keys[i]);
    }
    if (!root->is_leaf)
        traverse(root->children[root->num_keys]);
}

int main() {
    Node *root = create_node(1); // Create a leaf node
    insert_non_full(root, 10);
    insert_non_full(root, 20);
    insert_non_full(root, 5);
    printf("Tree keys: ");
    traverse(root);
    return 0;
}
```

2-3-4 tree:

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

```
#define MAX_KEYS 3
#define MAX_CHILDREN 4

typedef struct Node {
    int keys[MAX_KEYS];
    struct Node *children[MAX_CHILDREN];
    int num_keys;
    int is_leaf;
} Node;

Node* create_node(int is_leaf) {
    Node *node = (Node*)malloc(sizeof(Node));
    node->num_keys = 0;
    node->is_leaf = is_leaf;
    for (int i = 0; i < MAX_CHILDREN; i++)
        node->children[i] = NULL;
    return node;
}

void insert_into_non_full(Node *node, int key) {
    int i = node->num_keys - 1;
    if (node->is_leaf) {
        while (i >= 0 && key < node->keys[i]) {
            node->keys[i + 1] = node->keys[i];
            i--;
        }
        node->keys[i + 1] = key;
        node->num_keys++;
    } else {
        while (i >= 0 && key < node->keys[i]) {
            i--;
        }
        i++;
        if (node->children[i]->num_keys == MAX_KEYS) {
            // Split child node here (simplified version, real implementation needed)
        }
        insert_into_non_full(node->children[i], key);
    }
}

void traverse(Node *node) {
    if (node == NULL) return;
    for (int i = 0; i < node->num_keys; i++) {
        if (!node->is_leaf)
            traverse(node->children[i]);
        printf("%d ", node->keys[i]);
    }
    if (!node->is_leaf)
        traverse(node->children[node->num_keys]);
}

int main() {
    Node *root = create_node(1); // Create a leaf node
    insert_into_non_full(root, 10);
    insert_into_non_full(root, 20);
    insert_into_non_full(root, 30);
    printf("Tree keys: ");
    traverse(root);
    return 0;
}
```