

TREES:

1. 2-3 TREE:

```
#include <stdio.h>

#include <stdlib.h>

// Define the structure for a 2-3 tree node
typedef struct Node {
    int keys[3]; // Array to store keys
    struct Node* children[4]; // Array to store child pointers
    int num_keys; // Number of keys in the node
} Node;

// Function to create a new 2-3 tree node
Node* create_node() {
    Node* node = (Node*) malloc(sizeof(Node));
    node->num_keys = 0;
    for (int i = 0; i < 4; i++) {
        node->children[i] = NULL;
    }
    return node;
}

// Function to insert a key into the 2-3 tree
void insert(Node** root, int key) {
    // If the tree is empty, create a new root node
    if (*root == NULL) {
        *root = create_node();
    }
}
```

```

    (*root)->keys[0] = key;
    (*root)->num_keys = 1;
    return;
}

```

// Find the leaf node where the key should be inserted

```

Node* current = *root;
while (current->num_keys == 3) {
    int i = 0;
    while (i < current->num_keys && current->keys[i] < key) {
        i++;
    }
    if (current->children[i] == NULL) {
        break;
    }
    current = current->children[i];
}

```

// Insert the key into the leaf node

```

int i = current->num_keys - 1;
while (i >= 0 && current->keys[i] > key) {
    current->keys[i + 1] = current->keys[i];
    i--;
}
current->keys[i + 1] = key;
current->num_keys++;

```

// Split the node if it has 3 keys

```

if (current->num_keys == 3) {

```

```

Node* new_node = create_node();
int mid_key = current->keys[1];
new_node->keys[0] = current->keys[2];
new_node->num_keys = 1;
current->num_keys = 2;
current->children[3] = new_node;
// Update the parent node
if (current == *root) {
    Node* new_root = create_node();
    new_root->keys[0] = mid_key;
    new_root->num_keys = 1;
    new_root->children[0] = current;
    new_root->children[1] = new_node;
    *root = new_root;
} else {
    Node* parent = *root;
    while (parent->children[0] != current) {
        parent = parent->children[0];
    }
    int i = 0;
    while (i < parent->num_keys && parent->keys[i] < mid_key) {
        i++;
    }
    parent->keys[i] = mid_key;
    parent->children[i + 1] = new_node;
    parent->num_keys++;
}
}
}

```

```

// Function to print the 2-3 tree
void print_tree(Node* node, int level) {
    if (node == NULL) {
        return;
    }
    for (int i = 0; i < level; i++) {
        printf(" ");
    }
    for (int i = 0; i < node->num_keys; i++) {
        printf("%d ", node->keys[i]);
    }
    printf("\n");
    for (int i = 0; i <= node->num_keys; i++) {
        print_tree(node->children[i], level + 1);
    }
}

```

```

int main() {
    Node* root = NULL;
    insert(&root, 10);
    insert(&root, 20);
    insert(&root, 5);
    insert(&root, 6);
    print_tree(root, 0);
    return 0;
}

```

2. 2-3-4 TREE:

```
#include <stdio.h>

#include <stdlib.h>

// Define the structure for a 2-3-4 tree node
typedef struct Node {
    int keys[4]; // Array to store keys
    struct Node* children[5]; // Array to store child pointers
    int num_keys; // Number of keys in the node
} Node;

// Function to create a new 2-3-4 tree node
Node* create_node() {
    Node* node = (Node*) malloc(sizeof(Node));
    node->num_keys = 0;
    for (int i = 0; i < 5; i++) {
        node->children[i] = NULL;
    }
    return node;
}

// Function to insert a key into the 2-3-4 tree
void insert(Node** root, int key) {
    // If the tree is empty, create a new root node
    if (*root == NULL) {
        *root = create_node();
        (*root)->keys[0] = key;
        (*root)->num_keys = 1;
    }
}
```

```

    return;
}

// Find the leaf node where the key should be inserted
Node* current = *root;
while (current->num_keys == 4) {
    int i = 0;
    while (i < current->num_keys && current->keys[i] < key) {
        i++;
    }
    if (current->children[i] == NULL) {
        break;
    }
    current = current->children[i];
}

// Insert the key into the leaf node
int i = current->num_keys - 1;
while (i >= 0 && current->keys[i] > key) {
    current->keys[i + 1] = current->keys[i];
    i--;
}
current->keys[i + 1] = key;
current->num_keys++;

// Split the node if it has 4 keys
if (current->num_keys == 4) {
    Node* new_node = create_node();
    int mid_key = current->keys[2];

```

```

new_node->keys[0] = current->keys[3];
new_node->num_keys = 1;
current->num_keys = 3;
current->children[4] = new_node;
// Update the parent node
if (current == *root) {
    Node* new_root = create_node();
    new_root->keys[0] = mid_key;
    new_root->num_keys = 1;
    new_root->children[0] = current;
    new_root->children[1] = new_node;
    *root = new_root;
} else {
    Node* parent = *root;
    while (parent->children[0] != current) {
        parent = parent->children[0];
    }
    int i = 0;
    while (i < parent->num_keys && parent->keys[i] < mid_key) {
        i++;
    }
    parent->keys[i] = mid_key;
    parent->children[i + 1] = new_node;
    parent->num_keys++;
}
}
}

```

```

// Function to print the 2-3-4 tree

```

```

void print_tree(Node* node, int level) {
    if (node == NULL) {
        return;
    }
    for (int i = 0; i < level; i++) {
        printf(" ");
    }
    for (int i = 0; i < node->num_keys; i++) {
        printf("%d ", node->keys[i]);
    }
    printf("\n");
    for (int i = 0; i <= node->num_keys; i++) {
        print_tree(node->children[i], level + 1);
    }
}

```

```

int main() {
    Node* root = NULL;
    insert(&root, 10);
    insert(&root, 20);
    insert(&root, 5);
    insert(&root, 6);
    insert(&root, 12);
    insert(&root, 30);
    print_tree(root, 0);
    return 0;
}

```