```c
/**
 * Implementation for inorder traversal of a 2-3 Tree
 * Author: Mithusayel Murmu
 */

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

/* Implementation of a 2-3 tree */
typedef struct _T3Tree T3Tree;
typedef struct _T3Node T3Node;
typedef enum { FALSE, TRUE } BOOL;

/* Nodes are supposed to split as soon as the key size reaches 3 */
#define T3_KEY_THRESHOLD 3

struct _T3Node {
    size_t n;                           // Number of keys in use
    int key[T3_KEY_THRESHOLD];          // Key array
    T3Node *ptr[T3_KEY_THRESHOLD+1];    // Child pointer array
    BOOL leaf;                          // If node is a leaf
};
struct _T3Tree { size_t size; T3Node *root; };

static T3Node * t3tree_create_node(BOOL leaf) {
    T3Node *node = (T3Node *) malloc(sizeof(T3Node));
    node->n = 0;

    size_t i;
    for (i = 0; i < T3_KEY_THRESHOLD; i++) {
        node->key[i] = 0; node->ptr[i] = NULL;
    }
    node->ptr[i] = NULL; node->leaf = leaf;
    return node;
}

T3Tree * t3tree_create() {
    T3Tree *tree = (T3Tree *) malloc(sizeof(T3Tree));
    tree->size = 0; tree->root = t3tree_create_node(TRUE);
    return tree;
}

static void t3tree_destroy_node(T3Node *node) {
    if (!node->leaf) {
        size_t i;
        for (i = 0; i <= node->n; i++)
            t3tree_destroy_node(node->ptr[i]);
    }
    free(node);
}

void t3tree_destroy(T3Tree *tree) {
    if (!tree) return;

    t3tree_destroy_node(tree->root);
    tree->root = NULL; tree->size = 0;
    free(tree);
}

static void t3tree_split_child(T3Tree *tree, T3Node *node, size_t pi, T3Node *parent) {
    T3Node *znode = t3tree_create_node(node->leaf);
    znode->n = node->n / 2;

    size_t i, median = znode->n;
    // Copy right half
    for (i = 0; i < median; i++) {
        znode->key[i] = node->key[i + median + 1];
        znode->ptr[i] = node->ptr[i + median + 1];
    }
    znode->ptr[i] = node->ptr[i + median + 1];
    // Update left key size
    node->n = median;

    // Make space in the parent
    if (parent == NULL) {
        // We were splitting the root, create new root
        T3Node *nroot = t3tree_create_node(FALSE);
        nroot->ptr[0] = node; tree->root = nroot;
        parent = nroot;
    }

    for (i = parent->n; i > pi; i--) {
        // Shift keys to the right by 1
        parent->key[i] = parent->key[i-1];
        // Shift child pointers to the right by 1
        parent->ptr[i+1] = parent->ptr[i];
    }
    // Copy median element to the parent
    parent->key[pi] = node->key[median];
    // Attach right child to parent
    parent->ptr[pi+1] = znode;
    // Update parent key size
    parent->n++;
}

/**
 * Recursively finds a node in the tree to insert @k in.
 * @tree:   Pointer to the 2-3 tree to use
 * @node:   Pointer to the node currently being inspected for insertion
 * @k:      Key; The integer value to insert
 * @pi:     Parent index; @node => @parent->child[pi]. 0 for root
 * @parent: Pointer to the parent of @node. NULL for root
 */
static void t3tree_insert_in_node(T3Tree *tree, T3Node *node, int k, size_t pi, T3Node *parent) {
    int i = node->n - 1;

    if (node->leaf) {
        while (i >= 0 && k < node->key[i]) { node->key[i+1] = node->key[i]; i--; }
        node->key[i+1] = k; node->n++;
    } else {
        while (i >= 0 && k < node->key[i]) i--;

        T3Node *child = node->ptr[i+1];
        t3tree_insert_in_node(tree, child, k, i + 1, node);
    }

    // If reached threshold size, split.
    // Guarantees that on next insertion, every node is less than the threshold size.
    if (node->n == T3_KEY_THRESHOLD)
        t3tree_split_child(tree, node, pi, parent);
}

void t3tree_insert(T3Tree *tree, int k) {
    t3tree_insert_in_node(tree, tree->root, k, 0, NULL);
    tree->size++;
}

static void t3tree_traverse_in_node(const T3Node *node, void (*callback)(int)) {
    size_t i;
    if (!node->leaf) {
        for (i = 0; i < node->n; i++) {
            t3tree_traverse_in_node(node->ptr[i], callback);
            callback(node->key[i]);
        }
        t3tree_traverse_in_node(node->ptr[i], callback);
    } else {
        for (i = 0; i < node->n; i++)
            callback(node->key[i]);
    }
}

void t3tree_traverse_in(const T3Tree *tree, void (*callback)(int)) {
    if (!tree || tree->size == 0) return;
    t3tree_traverse_in_node(tree->root, callback);
}

void print_utility(int k) { printf("%d ", k); }
#define _scand(n) scanf("%d", &n)

int main(int argc, char const *argv[]) {
    int N, k;
    T3Tree *tree = t3tree_create();

    printf("Number of elements to be inserted: ");
    _scand(N);
    printf("Enter %d space separated integers: ", N);

    while (N--) {
        _scand(k);
        t3tree_insert(tree, k);
    }

    printf("\nPrinting while traversal:\n");
    t3tree_traverse_in(tree, print_utility);
    printf("\n"); t3tree_destroy(tree);

    return 0;
}

```