Experiment 8:- Write a C program to perform the following operation:

A) Insertion into a B-tree.

[*] Exp 8a.c

```c
#include <stdio.h>
#include <stdlib.h>
#define MAX_KEYS 3
typedef struct Node {
    int keys[MAX_KEYS];
    struct Node* children[MAX_KEYS + 1];
    int num_keys;
    int is_leaf;
} Node;
Node* createNode() {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->num_keys = 0;
    newNode->is_leaf = 1;
    return newNode;
}
int searchKeyPosition(Node* node, int key) {
    int pos = 0;
    while (pos < node->num_keys && key > node->keys[pos]) {
        pos++;
    }
    return pos;
}
```

```c
void splitChild(Node* parent, int index, Node* child) {
    Node* newNode = createNode();
    newNode->is_leaf = child->is_leaf;
    newNode->num_keys = MAX_KEYS / 2;
    int i;
    for (i = 0; i < MAX_KEYS / 2; i++) {
        newNode->keys[i] = child->keys[i + MAX_KEYS / 2];
    }
    if (!child->is_leaf) {
        for (i = 0; i < MAX_KEYS / 2 + 1; i++) {
            newNode->children[i] = child->children[i + MAX_KEYS / 2];
        }
    }
    child->num_keys = MAX_KEYS / 2;
    parent->num_keys++;
    for (i = parent->num_keys - 1; i > index; i--) {
        parent->keys[i] = parent->keys[i - 1];
        parent->children[i + 1] = parent->children[i];
    }
    parent->keys[index] = child->keys[MAX_KEYS / 2 - 1];
    parent->children[index + 1] = newNode;
}
```

```c
void insertNonFull(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 {
        int pos = searchKeyPosition(node, key);
        if (node->children[pos]->num_keys == MAX_KEYS) {
            splitChild(node, pos, node->children[pos]);
            if (key > node->keys[pos]) {
                pos++;
            }
        }
        insertNonFull(node->children[pos], key);
    }
}
Node* insert(Node* root, int key) {
    if (root->num_keys == MAX_KEYS) {
        Node* newNode = createNode();
        newNode->is_leaf = 0;
        newNode->children[0] = root;
        splitChild(newNode, 0, root);
        insertNonFull(newNode, key);
        return newNode;
    } else {
        insertNonFull(root, key);
        return root;
    }
}
void printBTree(Node* root, int level) {
    if (root) {
        int i,j;
        for (i = root->num_keys - 1; i >= 0; i--) {
            printBTree(root->children[i + 1], level + 1);
            for (j = 0; j < level; j++) {
                printf("    ");
            }
            printf("%d\n", root->keys[i]);
        }
        printBTree(root->children[i + 1], level + 1);
    }
}
```

```c
int main() {
    Node* root = createNode();
    int i;
    int keys[] = {3, 7, 1, 5, 10, 8, 2, 6, 12, 4};
    for (i = 0; i < sizeof(keys) / sizeof(keys[0]); i++) {
        root = insert(root, keys[i]);
        printf("Inserted %d into B-tree.\n", keys[i]);
        printf(root, 0);
        printf("\n");
    }
    return 0;
}
```

Output:

```
Inserted 3 into B-tree.

Inserted 7 into B-tree.

Inserted 1 into B-tree.

Inserted 5 into B-tree.

Inserted 10 into B-tree.

Inserted 8 into B-tree.

Inserted 2 into B-tree.

Inserted 6 into B-tree.

Inserted 12 into B-tree.

Inserted 4 into B-tree.


--------------------------------
Process exited after 0.0289 seconds with return value 0
Press any key to continue . . .
```

B) Write a C program for implementing Heap sort algorithm for sorting a given list of integers in ascending order.

Exp 8b.c

```c
#include <stdio.h>
void heapify(int arr[], int n, int i) {
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;
    if (left < n && arr[left] > arr[largest]) {
        largest = left;
    }
    if (right < n && arr[right] > arr[largest]) {
        largest = right;
    }
    if (largest != i) {
        // Swap arr[i] and arr[largest]
        int temp = arr[i];
        arr[i] = arr[largest];
        arr[largest] = temp;
        // Recursively heapify the affected sub-tree
        heapify(arr, n, largest);
    }
}
```

```c
void heapSort(int arr[], int n) {
    int i;
    // Build heap (rearrange array)
    for (i = n / 2 - 1; i >= 0; i--) {
        heapify(arr, n, i);
    }
    // One by one extract an element from the heap
    for (i = n - 1; i > 0; i--) {
        // Move the current root to the end
        int temp = arr[0];
        arr[0] = arr[i];
        arr[i] = temp;
        // Call max heapify on the reduced heap
        heapify(arr, i, 0);
    }
}
```

```c
void printArray(int arr[], int size) {
    int i;
    for (i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}
int main() {
    int arr[] = {12, 11, 13, 5, 6, 7};
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("Original array: ");
    printArray(arr, n);
    heapSort(arr, n);
    printf("Sorted array: ");
    printArray(arr, n);
    return 0;
}
```

Output:

```
Original array: 12 11 13 5 6 7
Sorted array: 5 6 7 11 12 13

--------------------------------
Process exited after 0.02746 seconds with return value 0
Press any key to continue . . .
```