**3. a. IMPLEMENTATION OF MERGE SORT**

#include <iostream>

using namespace std;

void merge(int \*,int, int , int );

void merge\_sort(int \*arr, int low, int high)

{

int mid;

if (low < high)

{

//divide the array at mid and sort independently using merge sort

mid=(low+high)/2;

merge\_sort(arr,low,mid);

merge\_sort(arr,mid+1,high);

//merge or conquer sorted arrays

merge(arr,low,high,mid);

}

}

// Merge sort

void merge(int \*arr, int low, int high, int mid)

{

int i, j, k, c[50];

i = low;

k = low;

j = mid + 1;

while (i <= mid && j <= high)

{

if (arr[i] < arr[j])

{

c[k] = arr[i];

k++;

i++;

}

else

{

c[k] = arr[j];

k++;

j++;

}

}

while (i <= mid)

{

c[k] = arr[i];

k++;

i++;

}

while (j <= high)

{

c[k] = arr[j];

k++;

j++;

}

for (i = low; i < k; i++)

{

arr[i] = c[i];

}

}

// read input array and call mergesort

int main()

{

int myarray[30], num, i,j;

cout<<"Enter number of elements to be sorted:";

cin>>num;

cout<<"Enter "<<num<<" elements to be sorted:";

for (i=0;i<num;i++)

{

cin>>myarray[i];

}

merge\_sort(myarray, 0, num-1);

cout<<"Sorted array\n";

for (j=0;j<num;j++)

{

cout<<myarray[j]<<"\t";

}

return 0;

}

**OUTPUT:**

Enter number of elements to be sorted:5

Enter 5 elements to be sorted: 5 2 6 3 1

Sorted array 1 2 3 5 6

**3. b. IMPLEMENTATION OF QUICK SORT**

#include <iostream>

using namespace std;

// Swap two elements - Utility function

void swap(int\* a, int\* b)

{

int t = \*a;

\*a = \*b;

\*b = t;

}

// partition the array using last element as pivot

int partition (int arr[], int low, int high)

{

int pivot = arr[high]; // pivot

int i = (low - 1);

for (int j = low; j <= high- 1; j++)

{

//if current element is smaller than pivot, increment the low element

//swap elements at i and j

if (arr[j] <= pivot)

{

i++; // increment index of smaller element

swap(&arr[i], &arr[j]);

}

}

swap(&arr[i + 1], &arr[high]);

return (i + 1);

}

//quicksort algorithm

void quickSort(int arr[], int low, int high)

{

if (low < high)

{

//partition the array

int pivot = partition(arr, low, high);

//sort the sub arrays independently

quickSort(arr, low, pivot - 1);

quickSort(arr, pivot + 1, high);

}

}

void displayArray(int arr[], int size)

{

int i;

for (i=0; i < size; i++)

cout<<arr[i]<<"\t";

}

int main()

{

int arr[] = {12,23,3,43,51,35,19,45};

int n = sizeof(arr)/sizeof(arr[0]);

cout<<"Input array"<<endl;

displayArray(arr,n);

cout<<endl;

quickSort(arr, 0, n-1);

cout<<"Array sorted with quick sort"<<endl;

displayArray(arr,n);

return 0;

}

**OUTPUT:**

Input array 12 23 3 43 51 35 19 45

Array sorted with quick sort 3 12 19 23 35 43 45 51

**4. IMPLEMENTATION OF A BINARY SEARCH TREE**

#include <iostream>

using namespace std;

struct node {

    int key;

    struct node \*left, \*right;

};

// A utility function to create a new BST node

struct node\* newNode(int item)

{

    struct node\* temp

        = new struct node;

    temp->key = item;

    temp->left = temp->right = NULL;

    return temp;

}

// A utility function to insert

// a new node with given key in BST

struct node\* insert(struct node\* node, int key)

{

    // If the tree is empty, return a new node

    if (node == NULL)

        return newNode(key);

    // Otherwise, recur down the tree

    if (key < node->key)

        node->left = insert(node->left, key);

    else if (key > node->key)

        node->right = insert(node->right, key);

    // Return the (unchanged) node pointer

    return node;

}

// Utility function to search a key in a BST

struct node\* search(struct node\* root, int key)

{

    // Base Cases: root is null or key is present at root

    if (root == NULL || root->key == key)

        return root;

    // Key is greater than root's key

    if (root->key < key)

        return search(root->right, key);

    // Key is smaller than root's key

    return search(root->left, key);

}

// Driver Code

int main()

{

    struct node\* root = NULL;

    root = insert(root, 50);

    insert(root, 30);

    insert(root, 20);

    insert(root, 40);

    insert(root, 70);

    insert(root, 60);

    insert(root, 80);

    // Key to be found

    int key = 6;

    // Searching in a BST

    if (search(root, key) == NULL)

        cout << key << " not found" << endl;

    else

        cout << key << " found" << endl;

    key = 60;

    // Searching in a BST

    if (search(root, key) == NULL)

        cout << key << " not found" << endl;

    else

        cout << key << " found" << endl;

    return 0;

}

**Output**

6 not found

60 found