WEST BENGAL STATE UNIVERSITY
BSc HONOURS IN COMPUTER SCIENCE
SEMESTER IV ASSIGNMENT
COURSE CODE-CMSACOR08P
REGISTRATION NUMBER-1072211401334

NAME-SWAPNIL GHOSH

ROLL NUMBER-6135

SERIAL NUMBER	ASSIGNMENT NAME	DATE	SIGNATURE
1	DFS	22/5/24	
2	BFS	22/5/24	
3	FLOYD	22/5/24	
4	PRIM’S ALGORITHM	22/5/24	
5	KRUSKAL’S ALGORITHM	22/5/24	
6	RECURSIVE QUICK SORT	9/6/14	
7	RECURSIVE MERGE SORT	9/6/24	
8	HEAP SORT	9/6/24	
9	RADIX SORT	9/6/24	
10	COUNTING SORT	9/6/24	
11	NON RECURSIVE QUICK SORT	9/6/24	
12	NON RECURSIVE MERGE SORT	9/6/24	

1.DFS

Code:

```
#include <iostream>
#include <conio.h>
#include <stdlib.h>

using namespace std;


int cost[10][10], i, j, k, n, stk[10], top, v, visit[10], visited[10]; int main() {
        int m;
        cout << "Enter no of vertices:"; cin >> n;
        cout << "Enter no of edges:"; cin
        >> m;
        cout << "\nEDGES \n";
        for (k = 1; k <= m; k++) { cin
                >> i >> j; cost[i][j] = 1;
                cout << endl;
        }
        cout << "Enter initial vertex to traverse from:"; cin >> v;
        cout << "DFS ORDER OF VISITED VERTICES:";
        cout << v << " "; visited[v] = 1;
        k = 1;
        while (k < n) {
                for (j = n; j >= 1; j--)
                        if (cost[v][j] != 0 && visited[j] != 1 && visit[j] != 1) { visit[j] = 1;
                                stk[top] = j; top++;
                        }
                v = stk[--top]; cout <<
                v << " "; k++;
                visit[v] = 0;
                visited[v] = 1;
        }
        cout << endl;
        return 0;
}
```

Output:

```
PS D:\> g++ .\DFS.cpp PS
D:\> .\a.exe
Enter no of vertices:5 Enter no
of edges:5

EDGE
S 1
```

```
2

1
5

2
5

2
3

3
4

Enter initial vertex to traverse from:1 DFS
ORDER OF VISITED VERTICES:1 2 3 4 5 PS
D:\>
```

2.BFS

Code:

```
#include <stdio.h>

int n, i, j, visited[10], queue[10], front = -1, rear = -1; int adj[10][10];

void bfs(int v)
{
        for (i = 1; i <= n; i++)
                if (adj[v][i] && !visited[i]) queue[+
                        +rear] = i;
        if (front <= rear)
        {
                visited[queue[front]] = 1;
                bfs(queue[front++]);
        }
}

int main()
{
        int v;
        printf("Enter the number of vertices: "); scanf("%d", &n);
        for (i = 1; i <= n; i++)
        {
                queue[i] = 0;
                visited[i] = 0;
        }
        printf("Enter graph data in matrix form:                        \n");
        for (i = 1; i <= n; i++)
                for (j = 1; j <= n; j++) scanf("%d",
                        &adj[i][j]);
        printf("Enter the starting vertex: "); scanf("%d",
        &v);
        bfs(v);
        printf("The node which are reachable are:                       \n");
        for (i = 1; i <= n; i++)
                if (visited[i]) printf("%d\t",
                        i);
                else
                        printf("BFS is not possible. Not all nodes are reachable");
        return 0;
}
```

Output:

```
PS D:\> g++ .\BFS.c
PS D:\> .\a.exe
Enter the number of vertices: 5 Enter
graph data in matrix form: 0
1
0
0
1
1
0
1
0
1
```

```
0
1
0
1
0
0
0
1
0
0
1
1
0
0
0
Enter the starting vertex: 1
The node which are reachable are:
1       2       3       4       5
PS D:\>
```

3.FLOYD

Code:

```
#include <bits/stdc++.h>
using namespace std;

#define V 4
#define INF 33333

void printSolution(int dist[][V]) {
    cout << "The following matrix shows the shortest distances between every pair of vertices
\n";
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) { if (dist[i]
            [j] == INF)
                 cout << "INF"
                      << " ";
            else
                cout << dist[i][j] << "        ";
        }
        cout << endl;
    }
}

void floydWarshall(int dist[][V]) { int i, j, k;
    for (k = 0; k < V; k++) {
        for (i = 0; i < V; i++) {
            for (j = 0; j < V; j++) {
                if (dist[i][j] > (dist[i][k] + dist[k][j]) && (dist[k][j] != INF && dist[i][k] != INF))
                    dist[i][j] = dist[i][k] + dist[k][j];
            }
        }
    }
    printSolution(dist);
}

int main() {
    int cost[V][V];
    cout << "Enter the costs for a graph with 4 edges: "; for (int i = 0; i
    < V; i++) {
        for (int j = 0; j < V; j++) { cin >>
            cost[i][j];
        }
    }
    floydWarshall(cost); return
    0;
}
```

Output:

```
PS D:\> g++ .\Floyd.cpp PS
D:\> .\a.exe
```

```
Enter the costs for a graph with 4 edges: 0 1
3
5
1
0
2
1
3
2
0
4
5
1
4
0
The following matrix shows the shortest distances between every pair of vertices
0    1    3    2
1    0    2    1
3    2    0    3
2    1    3    0
PS D:\>
```

4. PRIM'S ALGORITHM

Code:

```
#include <iostream> using
namespace std;
// Number of vertices in the graph const int V
= 6;
// Function to find the vertex with minimum key value int
min_Key(int key[], bool visited[])
{
        int min = 333, min_index; // 333 represents an Infinite value for (int v = 0; v
        < V; v++)
        {
                if (visited[v] == false && key[v] < min)
                {
                        // vertex should not be visited min =
                        key[v];
                        min_index = v;
                }
        }
        return min_index;
}
// Function to print the final MST stored in parent[] void
print_MST(int parent[], int cost[V][V])
{
        int minCost = 0;
        cout << "Edge \tWeight\n"; for
        (int i = 1; i < V; i++)
        {
                cout << parent[i] << " - " << i << " \t" << cost[i][parent[i]] << " \n"; minCost += cost[i]
                [parent[i]];
        }
        cout << "Total cost is" << minCost;
}
// Function to find the MST using adjacency cost matrix representation void
find_MST(int cost[V][V])
{
        int parent[V], key[V]; bool
        visited[V];
        // Initialize all the arrays for (int i =
        0; i < V; i++)
        {
```

```
    key[i] = 333; // 33 represents an Infinite value
    visited[i] = false; parent[i] = -
    1;
}
key[0] = 0;          // Include first vertex in MST by setting its key vaue to 0. parent[0] = -
1; // First node is always root of MST
// The MST will have maximum V-1 vertices for (int
x = 0; x < V - 1; x++)
{
    // Finding the minimum key vertex from the
    // set of vertices not yet included in MST int u =
    min_Key(key, visited);
```

```
        visited[u] = true; // Add the minimum key vertex to the MST
        // Update key and parent arrays for (int
        v = 0; v < V; v++)
        {
            // cost[u][v] is non zero only for adjacent vertices of u
            // visited[v] is false for vertices not yet included in MST
            // key[] gets updated only if cost[u][v] is smaller than key[v]
            if (cost[u][v] != 0 && visited[v] == false && cost[u][v] < key[v])
            {
                parent[v] = u; key[v] =
                cost[u][v];
            }
        }
    }
    // print the final MST
    print_MST(parent, cost);
}
// main function
int main()
{
    int cost[V][V];
    cout << "Enter the costs for a graph with 6 edges: "; for (int i = 0; i
    < V; i++)
    {
        for (int j = 0; j < V; j++)
        {
            cin >> cost[i][j];
        }
    }
    find_MST(cost);
    return 0;
}
```

Output:

```
PS D:\> g++ .\Prims.cpp PS
D:\> .\a.exe
Enter the costs for a graph with 6 edges: 0 1
3
3
5
7
1
0
2
6
2
1
3
2
0
1
3
4
3
6
1
0
5
```

```
2
4
2
3
5
0
5
7
1
4
2
5
0
Edge      Weight
0 - 1     1
1 - 2     2
2 - 3     1
1 - 4     2
1 - 5     1
Total cost is7 PS
D:\>
```

5.KRUSKAL’S ALGORITHM

Code:

```
#include <iostream> #include
<algorithm>

using namespace std; const

int MAX = 1e4 + 5;
int id[MAX], nodes, edges;
pair<long long, pair<int, int>> p[MAX];

void init() {
    for (int i = 0; i < MAX; ++i) id[i] = i;
}

int root(int x) {
    while (id[x] != x) {
        id[x] = id[id[x]]; x =
        id[x];
    }
    return x;
}

void union1(int x, int y) { int p =
    root(x);
    int q = root(y); id[p] =
    id[q];
}

long long kruskal(pair<long long, pair<int, int>> p[]) { int x, y;
    long long cost, minimumCost = 0; for
    (int i = 0; i < edges; ++i) {
        x = p[i].second.first; y =
        p[i].second.second; cost =
        p[i].first;
        if (root(x) != root(y))
            { minimumCost += cost;
            union1(x, y);
        }
    }
    return minimumCost;
}

  int main() {
       int x, y;
    long long weight, cost, minimumCost; init();
    cout << "Enter number of Nodes: "; cin >>
    nodes;
    cout << "Enter number of edges: "; cin >>
    edges;
```

```
        for (int i = 0; i < edges; ++i) {
                cout << "Enter the value of X, Y and edges: "; cin >> x
                >> y >> weight;
                p[i] = make_pair(weight, make_pair(x, y));
        }
        sort(p, p + edges); minimumCost =
        kruskal(p);
        cout << "Minimum cost is " << minimumCost << endl; return 0;
}
```

Output:

```
PS D:\> g++ .\Krushkal.cpp PS
D:\> .\a.exe
Enter number of Nodes: 4 Enter
number of edges: 6
Enter the value of X, Y and edges: 1 2 1 Enter
the value of X, Y and edges: 1 3 3 Enter the
value of X, Y and edges: 2 3 2 Enter the value of
X, Y and edges: 2 4 1 Enter the value of X, Y
and edges: 3 4 4 Enter the value of X, Y and
edges: 1 4 5 Minimum cost is 4
PS D:\>
```

6. RECURSIVE QUICK SORT.

Code:

```
#include<iostream>

class QuickSort {
public:
        void swap(int &a, int &b) {
                a = a + b;
                b = a - b;
                a = a - b;
        }

        int partition(int arr[], int lower, int higher) { int left, right,
                temp, pivot, flag;
                pivot = left = lower; right =
                higher;
                flag = 0; while(flag !=
                1) {
                        while((arr[pivot] <= arr[right]) && (pivot != right)) { right--;
                        }
                        if(pivot == right) flag =
                                1;
                        else if(arr[pivot] > arr[right]) {
                                std::swap(arr[pivot], arr[right]); pivot = right;
                        }
                        if(flag != 1) {
                                while((arr[pivot] >= arr[left]) && (pivot != left)) { left++;
                                }
                                if(pivot == left) flag
                                        = 1;
                                else if(arr[pivot] < arr[left]) {
                                        std::swap(arr[pivot], arr[left]); pivot =
                                        left;
                                }
                        }
                }
                return pivot;
        }

        void quickSort(int arr[], int lower, int higher) { int pivot;
                if(lower < higher) {
                        pivot = partition(arr, lower, higher); quickSort(arr,
                        lower, pivot); quickSort(arr, (pivot + 1), higher);
                }
        }

        void display(int array[], int length) { std::cout
                << "[ ";
                for (int i = 0; i < length; i++) { std::cout <<
                        array[i];
                                     if (i != (length - 1))
                                         std::cout << ", ";
                }
                std::cout << " ]" << std::endl;
        }
};

int main() {
        QuickSort ob;
        int length;
        std::cout << "Enter the number of elements: "; std::cin >>
        length;
        int* array = new int[length];
        std::cout << "Enter " << length << " array elements" << std::endl; for (int i = 0; i <
        length; i++) {
                std::cout << "a[" << i << "] = "; std::cin >>
                array[i];
        }
        ob.quickSort(array, 0, (length - 1)); std::cout <<
        "sorted array" << std::endl; ob.display(array, length);
        return 0;
}
```

Output:

```
PS D:\> g++ .\QuickSort.cpp PS
D:\> .\a.exe
Enter the number of elements: 7 Enter 7
array elements
a[0] = 37
a[1] = 750
a[2] = 6
a[3] = 10
a[4] = 128
a[5] = 582
a[6] = 81
sorted array
[ 6, 10, 81, 37, 128, 582, 750 ] PS D:\>
```

7. RECURSIVE MERGE SORT.

Code:

```
#include<iostream>

class MergeSort {
public:
        void merge(int arr[], int low, int mid, int high) {
                int lengthl = (mid - low) + 1, lengthr = (high - mid), l, r, index; int left[lengthl],
                right[lengthr];
                for(int i = 0, j = low; j <= mid; i++, j++) { left[i] =
                        arr[j];
                }
                for(int i = 0, j = (mid + 1); j <= high; i++, j++) { right[i] = arr[j];
                }
                l = 0;
                r = 0;
                index = low;
                while((l < lengthl) && (r < lengthr)) { if(left[l] <=
                        right[r]) {
                                arr[index] = left[l]; l++;
                                index++;
                        } else {
                                arr[index] = right[r]; r++;
                                index++;
                        }
                }
                while(l < lengthl) { arr[index]
                        = left[l]; l++;
                        index++;
                }
                while(r < lengthr) { arr[index] =
                        right[r]; r++;
                        index++;
                }
        }

        void mergeSort(int arr[], int low, int high) { if(low < high) {
                        int mid = (low + high) / 2; mergeSort(arr,
                        low, mid); mergeSort(arr, (mid + 1), high);
                        merge(arr, low, mid, high);
                }
        }

        void display(int array[], int length) { std::cout
                << "[ ";
                for (int i = 0; i < length; i++) {
                    std::cout << array[i]; if (i !=
                                        (length - 1))
                              std::cout << ", ";
                }
                std::cout << " ]" << std::endl;
        }
};

int main() {
        MergeSort ob;
        int length;
        std::cout << "Enter the number of elements: "; std::cin >>
        length;
        int* array = new int[length];
        std::cout << "Enter " << length << " array elements" << std::endl; for (int i = 0; i <
        length; i++) {
                std::cout << "a[" << i << "] = "; std::cin >>
                array[i];
        }
        ob.mergeSort(array, 0, (length - 1)); std::cout <<
        "sorted array" << std::endl; ob.display(array, length);
        return 0;
}
```

Output:

```
PS D:\> g++ .\MergeSort.cpp PS
D:\> .\a.exe
Enter the number of elements: 8 Enter 8
array elements
a[0] = 378
a[1] = 57
a[2] = 582
a[3] = 65
a[4] = 367
a[5] = 2
a[6] = 45
a[7] = 11
sorted array
[ 2, 11, 45, 57, 65, 582, 367, 378 ] PS D:\>
```

8. HEAP SORT

Code:

```
#include <iostream>

using namespace std;

void heapify(int arr[], int N, int i) { int largest = i;
        int l = 2 * i + 1; int r =
        2 * i + 2;
        if (l < N && arr[l] > arr[largest]) largest = l;
        if (r < N && arr[r] > arr[largest]) largest = r;
        if (largest != i) { swap(arr[i],
                arr[largest]); heapify(arr, N,
                largest);
        }
}

void heapSort(int arr[], int N) {
        for (int i = N / 2 - 1; i >= 0; i--) heapify(arr, N,
                i);
        for (int i = N - 1; i > 0; i--) {
                swap(arr[0], arr[i]);
                heapify(arr, i, 0);
        }
}

void printArray(int arr[], int N) { for (int i =
        0; i < N; ++i)
                cout << arr[i] << " "; cout <<
        "\n";
}

int main() {
        int length;
        cout << "Enter the number of elements: "; cin >>
        length;
        int* array = new int[length];
        cout << "Enter " << length << " array elements" << std::endl; for (int i = 0; i
        < length; i++) {
                cout << "a[" << i << "] = "; cin >>
                array[i];
        }
        heapSort(array, length);  cout <<
        "Sorted array is \n";
        printArray(array, length);
}
```

Output:

```
PS D:\> g++ .\HeapSort.cpp PS
D:\> .\a.exe
```

```
Enter the number of elements: 5 Enter 5
array elements
a[0] = 3
a[1] = 7
a[2] = 7
a[3] = 5
a[4] = 0
Sorted array is
0 5 7 7 3
PS D:\>
```

9. RADIX SORT

Code:

```
#include<iostream>

class RadixSort {
public:
        void radixSort(int arr[], int size) {
                int i = 1, j = 10, count = 0, max = arr[0], k, index, n1, n2; int temp[10],
                arr2D[10][size];
                for(int t = 1; t < size; t++) { if(max < arr[t])
                        {
                                max = arr[t];
                        }
                }
                while(max > 0) { max
                        = max / 10;
                        count++;
                }
                for(k = 1; k <= count; k++) { for(int t = 0; t
                        < 10; t++) {
                                temp[t] = 0;
                        }
                        for(index = 0; index < size; index++) { n1 =
                                arr[index] % j;
                                n2 = n1 / i;
                                arr2D[n2][temp[n2]] = arr[index]; temp[n2]+
                                +;
                        }
                        int temp1 = 0;
                        for(int row = 0; row < 10; row++) {
                                for(int column = 0; column < temp[row]; column++) { arr[temp1] =
                                        arr2D[row][column];
                                        temp1++;
                                }
                        }
                        i = i * 10;
                        j = j * 10;
                }
        }

        void display(int array[], int length) { std::cout
                << "[ ";
                for (int i = 0; i < length; i++) { std::cout <<
                        array[i];
                                    if (i != (length - 1))
                                        std::cout << ", ";
                }
                std::cout << " ]" << std::endl;
        }
};

int main() {
        RadixSort ob;
        int length;
        std::cout << "Enter the number of elements: ";
```

```
    std::cin >> length;
    int* array = new int[length];
    std::cout << "Enter " << length << " array elements" << std::endl; for (int i = 0; i <
    length; i++) {
        std::cout << "a[" << i << "] = "; std::cin >>
        array[i];
    }
    ob.radixSort(array,  length);
    std::cout << "sorted array" << std::endl; ob.display(array,
    length);
    return 0;
}
```

Output:

```
PS D:\> g++ .\RadixSort.cpp PS
D:\> .\a.exe
Enter the number of elements: 8 Enter 8
array elements
a[0] = 50
a[1] = 21
a[2] = 62
a[3] = 312
a[4] = 722
a[5] = 43
a[6] = 35
a[7] = 338
sorted array
[ 21, 43, 50, 62, 35, 722, 312, 338 ] PS D:\>
```

10. COUNTING SORT

Code:

```
#include <iostream>

class CountingSort { public:
    void countingSort(int input[], int length) { int max = -
        33333, min = 33333, i;
        for (int i = 0; i < length; i++) { if (input[i] >
            max) {
                max = input[i];
            }
            if (input[i] < min) { min
                = input[i];
            }
        }
        int countingArrayLength = max - min + 1;
        int countingArray[countingArrayLength], sorted[length]; for (int i =
        0; i < countingArrayLength; i++) {
            countingArray[i] = 0;
        }
        for (int i = 0; i < length; i++) { countingArray[input[i] -
            min]++;
        }
        for (int i = 1; i < countingArrayLength; i++) { countingArray[i] =
            countingArray[i] + countingArray[i - 1];
        }
        for (int i = 0; i < length; i++) {
            sorted[--countingArray[input[i] - min]] = input[i];
        }
    }

    void display(int array[], int length) { std::cout
        << "[ ";
        for (int i = 0; i < length; i++) { std::cout <<
            array[i];
                   if (i != (length - 1))
                     std::cout << ", ";
        }
        std::cout << " ]" << std::endl;
    }
};

int main() {
    CountingSort ob;
    int length;
    std::cout << "Enter the number of elements: "; std::cin >>
    length;
    int* array = new int[length];
    std::cout << "Enter " << length << " array elements" << std::endl; for (int i = 0; i <
    length; i++) {
        std::cout << "a[" << i << "] = "; std::cin >>
        array[i];
    }
    ob.countingSort(array,   length);
    std::cout << "sorted array" << std::endl; ob.display(array,
    length);
    return 0;
}
```

Output:

```
PS D:\> g++ .\CountingSort.cpp PS
D:\> .\a.exe
Enter the number of elements: 8 Enter 8
array elements
a[0] = 833
a[1] = 53
a[2] = 34
a[3] = 227
a[4] = 213
a[5] = 26
a[6] = 12
a[7] = 5

output array
[ 5, 12, 26, 34, 53, 213, 227, 833 ] PS D:\>
```

11. NON-RECURSIVE QUICK SORT.

Code-

```
#include <bits/stdc++.h>
using namespace std;

// A utility function to swap two elements
void swap(int* a, int* b)
{
    int t = *a;
    *a = *b;
    *b = t;
}

/* This function is same in both iterative and recursive*/
int partition(int arr[], int l, int h)
{
    int x = arr[h];
    int i = (l - 1);

    for (int j = l; j <= h - 1; j++) {
        if (arr[j] <= x) {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[h]);
    return (i + 1);
}

/* A[] --> Array to be sorted,
l --> Starting index,
h --> Ending index */
void quickSortIterative(int arr[], int l, int h)
{
    // Create an auxiliary stack
    int stack[h - l + 1];

    // initialize top of stack
    int top = -1;

    // push initial values of l and h to stack
    stack[++top] = l;
    stack[++top] = h;

    // Keep popping from stack while is not empty
    while (top >= 0) {
        // Pop h and l
        h = stack[top--];
        l = stack[top--];

        // Set pivot element at its correct position
        // in sorted array
        int p = partition(arr, l, h);

        // If there are elements on left side of pivot,
        // then push left side to stack
        if (p - 1 > l) {
            stack[++top] = l;
            stack[++top] = p - 1;
        }

        // If there are elements on right side of pivot,
        // then push right side to stack
        if (p + 1 < h) {
            stack[++top] = p + 1;
            stack[++top] = h;
        }
    }
}

// A utility function to print contents of arr
void printArr(int arr[], int n)
{
    int i;
    for (i = 0; i < n; ++i)
        cout << arr[i] << " ";
}

// Driver code
int main()
{
    int arr[] = { 4, 3, 5, 2, 1, 3, 2, 3 };
    int n = sizeof(arr) / sizeof(*arr);
    quickSortIterative(arr, 0, n - 1);
    printArr(arr, n);
    return 0;
}
```

OUTPUT:

```
swapnil-ghosh@swapnil-ghosh-HP-Laptop-
15s-eq2xxx:~$ g++ nonrecursivequick.cpp
swapnil-ghosh@swapnil-ghosh-HP-Laptop-
15s-eq2xxx:~$ ./a.out
1 2 2 3 3 3 4 5
```

12. NON RECURSIVE MERGE SORT

Code-

```
#include <bits/stdc++.h>
using namespace std;

/* Function to merge the two haves arr[l..m] and arr[m+1..r] of array arr[] */
void merge(int arr[], int l, int m, int r);

// Utility function to find minimum of two integers
int min(int x, int y) { return (x<y)? x :y; }


/* Iterative mergesort function to sort arr[0...n-1] */
void mergeSort(int arr[], int n)
{
   int curr_size;  // For current size of subarrays to be merged
                   // curr_size varies from 1 to n/2
   int left_start; // For picking starting index of left subarray
                   // to be merged

   // Merge subarrays in bottom up manner.  First merge subarrays of
   // size 1 to create sorted subarrays of size 2, then merge subarrays
   // of size 2 to create sorted subarrays of size 4, and so on.
   for (curr_size=1; curr_size<=n-1; curr_size = 2*curr_size)
   {
       // Pick starting point of different subarrays of current size
       for (left_start=0; left_start<n-1; left_start += 2*curr_size)
       {
           // Find ending point of left subarray. mid+1 is starting
           // point of right
           int mid = min(left_start + curr_size - 1, n-1);

           int right_end = min(left_start + 2*curr_size - 1, n-1);

           // Merge Subarrays arr[left_start...mid] & arr[mid+1...right_end]
           merge(arr, left_start, mid, right_end);
       }
   }
}

/* Function to merge the two haves arr[l..m] and arr[m+1..r] of array arr[] */
void merge(int arr[], int l, int m, int r)
{
    int i, j, k;
    int n1 = m - l + 1;
    int n2 =  r - m;

    /* create temp arrays */
    int L[n1], R[n2];

    /* Copy data to temp arrays L[] and R[] */
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1+ j];

    /* Merge the temp arrays back into arr[l..r]*/
    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2)
    {
        if (L[i] <= R[j])
        {
            arr[k] = L[i];
            i++;
        }
        else
        {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    /* Copy the remaining elements of L[], if there are any */
    while (i < n1)
    {
        arr[k] = L[i];
        i++;
        k++;
    }

    /* Copy the remaining elements of R[], if there are any */
    while (j < n2)
    {
        arr[k] = R[j];
        j++;
        k++;
    }
}

/* Function to print an array */
void printArray(int A[], int size)
{
    int i;
    for (i=0; i < size; i++)
        cout <<" "<< A[i];
    cout <<"\n";
}

/* Driver program to test above functions */
int main()
{
    int arr[] = {12, 11, 13, 5, 6, 7};
    int n = sizeof(arr)/sizeof(arr[0]);

    cout <<"Given array is \n ";
    printArray(arr, n);

    mergeSort(arr, n);

    cout <<"\nSorted array is \n ";
    printArray(arr, n);
    return 0;
}
```

OUTPUT:

```
swapnil-ghosh@swapnil-ghosh-HP-Laptop-15s-eq2xxx:~$ g++ nonrecursivemerge.cpp
swapnil-ghosh@swapnil-ghosh-HP-Laptop-15s-eq2xxx:~$ ./a.out
Given array is
  12 11 13 5 6 7

Sorted array is
  5 6 7 11 12 13
```