C++ Test code 1

```
// C++ program for Merge Sort
#include <iostream>
using namespace std;

// Merges two subarrays of array[].
// First subarray is arr[begin..mid]
// Second subarray is arr[mid+1..end]
void merge(int array[], int const left, int const mid,
           int const right)
{
    auto const subArrayOne = mid - left + 1;
    auto const subArrayTwo = right - mid;

    // Create temp arrays
    auto *leftArray = new int[subArrayOne],
        *rightArray = new int[subArrayTwo];

    // Copy data to temp arrays leftArray[] and rightArray[]
    for (auto i = 0; i < subArrayOne; i++)
        leftArray[i] = array[left + i];
    for (auto j = 0; j < subArrayTwo; j++)
        rightArray[j] = array[mid + 1 + j];

    auto indexOfSubArrayOne
        = 0, // Initial index of first sub-array
        indexOfSubArrayTwo
        = 0; // Initial index of second sub-array
    int indexOfMergedArray
        = left; // Initial index of merged array

    // Merge the temp arrays back into array[left..right]
    while (indexOfSubArrayOne < subArrayOne
        && indexOfSubArrayTwo < subArrayTwo) {
        if (leftArray[indexOfSubArrayOne]
            <= rightArray[indexOfSubArrayTwo]) {
            array[indexOfMergedArray]
                = leftArray[indexOfSubArrayOne];
            indexOfSubArrayOne++;
        }
        else {
            array[indexOfMergedArray]
                = rightArray[indexOfSubArrayTwo];
            indexOfSubArrayTwo++;
```

```
                }
                indexOfMergedArray++;
        }
        // Copy the remaining elements of
        // left[], if there are any
        while (indexOfSubArrayOne < subArrayOne) {
                array[indexOfMergedArray]
                        = leftArray[indexOfSubArrayOne];
                indexOfSubArrayOne++;
                indexOfMergedArray++;
        }
        // Copy the remaining elements of
        // right[], if there are any
        while (indexOfSubArrayTwo < subArrayTwo) {
                array[indexOfMergedArray]
                        = rightArray[indexOfSubArrayTwo];
                indexOfSubArrayTwo++;
                indexOfMergedArray++;
        }
        delete[] leftArray;
        delete[] rightArray;
}

// begin is for left index and end is
// right index of the sub-array
// of arr to be sorted */
void mergeSort(int array[], int const begin, int const end)
{
        if (begin >= end)
                return; // Returns recursively

        auto mid = begin + (end - begin) / 2;
        mergeSort(array, begin, mid);
        mergeSort(array, mid + 1, end);
        merge(array, begin, mid, end);
}

// UTILITY FUNCTIONS
// Function to print an array
void printArray(int A[], int size)
{
        for (auto i = 0; i < size; i++)
                cout << A[i] << " ";
}
```

```
// Driver code
int main()
{
	int arr[] = { 12, 11, 13, 5, 6, 7 };
	auto arr_size = sizeof(arr) / sizeof(arr[0]);

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

	mergeSort(arr, 0, arr_size - 1);

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

// This code is contributed by Mayank Tyagi
// This code was revised by Joshua Estes
```

C++ Test code 2

```
// C++ Program for Floyd Warshall Algorithm
#include <bits/stdc++.h>
using namespace std;

// Number of vertices in the graph
#define V 4

/* Define Infinite as a large enough
value.This value will be used for
vertices not connected to each other */
#define INF 99999

// A function to print the solution matrix
void printSolution(int dist[][V]);

// Solves the all-pairs shortest path
// problem using Floyd Warshall algorithm
void floydWarshall(int dist[][V])
{

	int i, j, k;

	/* Add all vertices one by one to
	the set of intermediate vertices.
	---> Before start of an iteration,
	we have shortest distances between all
	pairs of vertices such that the
	shortest distances consider only the
	vertices in set {0, 1, 2, .. k-1} as
	intermediate vertices.
	----> After the end of an iteration,
	vertex no. k is added to the set of
	intermediate vertices and the set becomes {0, 1, 2, ..
	k} */
	for (k = 0; k < V; k++) {
		// Pick all vertices as source one by one
		for (i = 0; i < V; i++) {
			// Pick all vertices as destination for the
			// above picked source
			for (j = 0; j < V; j++) {
				// If vertex k is on the shortest path from
				// i to j, then update the value of
				// dist[i][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];
                        }
                }
        }

        // Print the shortest distance matrix
        printSolution(dist);
}

/* A utility function to print solution */
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;
        }
}

// Driver's code
int main()
{
        /* Let us create the following weighted graph
                        10
        (0)------->(3)
                |           /|\
        5 |         |
                |           | 1
        \|/         |
        (1)------->(2)
                        3           */
        int graph[V][V] = { { 0, 5, INF, 10 },
                                                { INF, 0, 3, INF },
```

```
                                                { INF, INF, 0, 1 },
                                                { INF, INF, INF, 0 } };

        // Function call
        floydWarshall(graph);
        return 0;
}

// This code is contributed by Mythri J L
```

Java Test code 1

```
/* Java program for Merge Sort */
class MergeSort {
	// Merges two subarrays of arr[].
	// First subarray is arr[l..m]
	// Second subarray is arr[m+1..r]
	void merge(int arr[], int l, int m, int r)
	{
		// Find sizes of two subarrays to be merged
		int n1 = m - l + 1;
		int n2 = r - m;

		/* Create temp arrays */
		int L[] = new int[n1];
		int R[] = new int[n2];

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

		/* Merge the temp arrays */

		// Initial indexes of first and second subarrays
		int i = 0, j = 0;

		// Initial index of merged subarray array
		int 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 remaining elements of L[] if any */
		while (i < n1) {
```

```
                arr[k] = L[i];
                i++;
                k++;
        }

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

// Main function that sorts arr[l..r] using
// merge()
void sort(int arr[], int l, int r)
{
        if (l < r) {
                // Find the middle point
                int m = l + (r - l) / 2;

                // Sort first and second halves
                sort(arr, l, m);
                sort(arr, m + 1, r);

                // Merge the sorted halves
                merge(arr, l, m, r);
        }
}

/* A utility function to print array of size n */
static void printArray(int arr[])
{
        int n = arr.length;
        for (int i = 0; i < n; ++i)
                System.out.print(arr[i] + " ");
        System.out.println();
}

// Driver code
public static void main(String args[])
{
        int arr[] = { 12, 11, 13, 5, 6, 7 };
```

```
                System.out.println("Given Array");
                printArray(arr);

                MergeSort ob = new MergeSort();
                ob.sort(arr, 0, arr.length - 1);

                System.out.println("\nSorted array");
                printArray(arr);
        }
}
/* This code is contributed by Rajat Mishra */
```

Java Test code 2

```
// Java program for Floyd Warshall All Pairs Shortest
// Path algorithm.
import java.io.*;
import java.lang.*;
import java.util.*;

class AllPairShortestPath {
	final static int INF = 99999, V = 4;

	void floydWarshall(int dist[][])
	{

		int i, j, k;



		/* Add all vertices one by one
		to the set of intermediate
		vertices.
		---> Before start of an iteration,
			we have shortest
			distances between all pairs
			of vertices such that
			the shortest distances consider
			only the vertices in
			set {0, 1, 2, .. k-1} as
			intermediate vertices.
		----> After the end of an iteration,
				vertex no. k is added
				to the set of intermediate
				vertices and the set
				becomes {0, 1, 2, .. k} */
		for (k = 0; k < V; k++) {
			// Pick all vertices as source one by one
			for (i = 0; i < V; i++) {
				// Pick all vertices as destination for the
				// above picked source
				for (j = 0; j < V; j++) {
					// If vertex k is on the shortest path
					// from i to j, then update the value of
					// dist[i][j]
```

```
                                if (dist[i][k] + dist[k][j]
                                        < dist[i][j])
                                        dist[i][j]
                                                = dist[i][k] + dist[k][j];
                        }
                }
        }

        // Print the shortest distance matrix
        printSolution(dist);
}

void printSolution(int dist[][])
{
        System.out.println(
                "The following matrix shows the shortest "
                + "distances between every pair of vertices");
        for (int i = 0; i < V; ++i) {
                for (int j = 0; j < V; ++j) {
                        if (dist[i][j] == INF)
                                System.out.print("INF ");
                        else
                                System.out.print(dist[i][j] + " ");
                }
                System.out.println();
        }
}

// Driver's code
public static void main(String[] args)
{
        /* Let us create the following weighted graph
        10
        (0)------->(3)
        |                /|\
        5 |              |
        |                | 1
        \|/              |
        (1)------->(2)
        3                */
        int graph[][] = { { 0, 5, INF, 10 },
                                        { INF, 0, 3, INF },
                                        { INF, INF, 0, 1 },
                                        { INF, INF, INF, 0 } };
```

```
                AllPairShortestPath a = new AllPairShortestPath();

                // Function call
                a.floydWarshall(graph);
        }
}

// Contributed by Aakash Hasija
```

Python Test code 1

```
# Python program for implementation of MergeSort
def mergeSort(arr):
    if len(arr) > 1:

        # Finding the mid of the array
        mid = len(arr)//2

        # Dividing the array elements
        L = arr[:mid]

        # into 2 halves
        R = arr[mid:]

        # Sorting the first half
        mergeSort(L)

        # Sorting the second half
        mergeSort(R)

        i = j = k = 0

        # Copy data to temp arrays L[] and R[]
        while i < len(L) and j < len(R):
            if L[i] <= R[j]:
                arr[k] = L[i]
                i += 1
            else:
                arr[k] = R[j]
                j += 1
            k += 1

        # Checking if any element was left
        while i < len(L):
            arr[k] = L[i]
            i += 1
            k += 1

        while j < len(R):
            arr[k] = R[j]
            j += 1
            k += 1

# Code to print the list
```

```
def printList(arr):
        for i in range(len(arr)):
                print(arr[i], end=" ")
        print()


# Driver Code
if __name__ == '__main__':
        arr = [12, 11, 13, 5, 6, 7]
        print("Given array is", end="\n")
        printList(arr)
        mergeSort(arr)
        print("Sorted array is: ", end="\n")
        printList(arr)

# This code is contributed by Mayank Khanna
```

Python Test code 2

```
# Python3 Program for Floyd Warshall Algorithm

# Number of vertices in the graph
V = 4

# Define infinity as the large
# enough value. This value will be
# used for vertices not connected to each other
INF = 99999

# Solves all pair shortest path
# via Floyd Warshall Algorithm


def floydWarshall(graph):
        """ dist[][] will be the output
        matrix that will finally
                have the shortest distances
                between every pair of vertices """
        """ initializing the solution matrix
        same as input graph matrix
        OR we can say that the initial
        values of shortest distances
        are based on shortest paths considering no
        intermediate vertices """

        dist = list(map(lambda i: list(map(lambda j: j, i)), graph))

        """ Add all vertices one by one
        to the set of intermediate
        vertices.
        ---> Before start of an iteration,
        we have shortest distances
        between all pairs of vertices
        such that the shortest
        distances consider only the
        vertices in the set
        {0, 1, 2, .. k-1} as intermediate vertices.
        ----> After the end of a
        iteration, vertex no. k is
        added to the set of intermediate
        vertices and the
        set becomes {0, 1, 2, .. k}
```

```
        """
        for k in range(V):

                # pick all vertices as source one by one
                for i in range(V):

                        # Pick all vertices as destination for the
                        # above picked source
                        for j in range(V):

                                # If vertex k is on the shortest path from
                                # i to j, then update the value of dist[i][j]
                                dist[i][j] = min(dist[i][j],
                                                        dist[i][k] + dist[k][j]
                                                        )
        printSolution(dist)


# A utility function to print the solution
def printSolution(dist):
        print("Following matrix shows the shortest distances\
between every pair of vertices")
        for i in range(V):
                for j in range(V):
                        if(dist[i][j] == INF):
                                print("%7s" % ("INF"), end=" ")
                        else:
                                print("%7d\t" % (dist[i][j]), end=' ')
                        if j == V-1:
                                print()


# Driver's code
if __name__ == "__main__":
"""
                        10
                (0)------->(3)
                |                  /|\
                5 |                 |
                |                  | 1
                \|/                |
                (1)------->(2)
                        3                  """
graph = [[0, 5, INF, 10],
```

```
                [INF, 0, 3, INF],
                [INF, INF, 0, 1],
                [INF, INF, INF, 0]
                ]
# Function call
floydWarshall(graph)
# This code is contributed by Mythri J L
```