1) maximum and minimum values in the array

def find\_min\_max(arr):

min\_value = min(arr)

max\_value = max(arr)

return min\_value, max\_value

array1 = [5, 7, 3, 4, 9, 12, 6, 2]

min1, max1 = find\_min\_max(array1)

print(f"Input: {array1} => Min = {min1}, Max = {max1}")

array2 = [1, 3, 5, 7, 9, 11, 13, 15, 17]

min2, max2 = find\_min\_max(array2)

print(f"Input: {array2} => Min = {min2}, Max = {max2}")

array3 = [22, 34, 35, 36, 43, 67, 12, 13, 15, 17]

min3, max3 = find\_min\_max(array3)

print(f"Input: {array3} => Min = {min3}, Max = {max3}")

Input : N= 8, a[] = {5,7,3,4,9,12,6,2}

Output : Min = 2, Max = 12

Input :N= 9, a[] = {1,3,5,7,9,11,13,15,17}

Output : Min = 1, Max = 17

Input : N= 10, a[] = {22,34,35,36,43,67, 12,13,15,17}

Output: Min 12, Max 67

2) Program to find the array's maximum and minimum values.

def find\_min\_max(arr):

minimum = min(arr)

maximum = max(arr)

return minimum, maximum

array1 = [2, 4, 6, 8, 10, 12, 14, 18]

min1, max1 = find\_min\_max(array1)

print(f"Input: N=8, a[] = {array1} => Min = {min1}, Max = {max1}")

array2 = [11, 13, 15, 17, 19, 21, 23, 35, 37]

min2, max2 = find\_min\_max(array2)

print(f"Input: N=9, a[] = {array2} => Min = {min2}, Max = {max2}")

array3 = [22, 34, 35, 36, 43, 67, 12, 13, 15, 17]

min3, max3 = find\_min\_max(array3)

print(f"Input: N=10, a[] = {array3} => Min = {min3}, Max = {max3}")

Input:N=8, 2,4,6,8,10,12,14,18.

Output : Min = 2, Max =18

Input : N= 9, a[] = {11,13,15,17,19,21,23,35,37}

Output : Min = 11, Max = 37

Input : N= 10, a[] = {22,34,35,36,43,67, 12,13,15,17}

Output: Min 12, Max 67

3) You are given an unsorted array of 31,23,35,27,11,21,15,28. Write a program for Merge Sort and implement it using any programming language you choose.

def merge\_sort(arr):

if len(arr) > 1:

mid = len(arr) // 2

left\_half = arr[:mid]

right\_half = arr[mid:]

merge\_sort(left\_half)

merge\_sort(right\_half)

i = j = k = 0

while i < len(left\_half) and j < len(right\_half):

if left\_half[i] < right\_half[j]:

arr[k] = left\_half[i]

i += 1

else:

arr[k] = right\_half[j]

j += 1

k += 1

while i < len(left\_half):

arr[k] = left\_half[i]

i += 1

k += 1

while j < len(right\_half):

arr[k] = right\_half[j]

j += 1

k += 1

arr1 = [31, 23, 35, 27, 11, 21, 15, 28]

merge\_sort(arr1)

print("Sorted array is:", arr1)

arr2 = [22, 34, 25, 36, 43, 67, 52, 13, 65, 17]

merge\_sort(arr2)

print("Sorted array is:", arr2)

Input :N= 8, a[] = {31,23,35,27,11,21,15,28}

Output:11,15,21,23,27,28,31,35

Input : N= 10, a[] = {22,34,25,36,43,67, 52,13,65,17}

Output: 13,17,22,25,34,36,43,52,65,67

4) Implement the Merge Sort algorithm in a programming language of your choice and test it on the array 12,4,78,23,45,67,89,1. Modify your implementation to count the number of comparisons made during the sorting process. Print this count along with the sorted array.

def merge\_sort(arr):

if len(arr) > 1:

mid = len(arr) // 2

left\_half = arr[:mid]

right\_half = arr[mid:]

merge\_sort(left\_half)

merge\_sort(right\_half)

i = j = k = 0

comparisons = 0

while i < len(left\_half) and j < len(right\_half):

comparisons += 1

if left\_half[i] < right\_half[j]:

arr[k] = left\_half[i]

i += 1

else:

arr[k] = right\_half[j]

j += 1

k += 1

while i < len(left\_half):

arr[k] = left\_half[i]

i += 1

k += 1

while j < len(right\_half):

arr[k] = right\_half[j]

j += 1

k += 1

return comparisons

return 0

array1 = [12, 4, 78, 23, 45, 67, 89, 1]

comparisons1 = merge\_sort(array1)

print(f"Sorted Array: {array1}, Comparisons: {comparisons1}")

array2 = [38, 27, 43, 3, 9, 82, 10]

comparisons2 = merge\_sort(array2)

print(f"Sorted Array: {array2}, Comparisons: {comparisons2}")

Input : N= 8, a[] = {12,4,78,23,45,67,89,1}

Output :1,4,12,23,45,67,78,89

Input : N= 7, a[] = {38,27,43,3,9,82,10}

Output :3,9,10,27,38,43,82.

5) Implement Floyd's Algorithm to find the shortest path between all pairs of cities

import numpy as np

def floyd\_warshall(n, edges):

# Initialize the distance matrix

dist = np.full((n, n), float('inf'))

for i in range(n):

dist[i][i] = 0

for u, v, w in edges:

dist[u][v] = min(dist[u][v], w)

print("Distance matrix before applying Floyd's Algorithm:")

print(dist)

for k in range(n):

for i in range(n):

for j in range(n):

if dist[i][j] > dist[i][k] + dist[k][j]:

dist[i][j] = dist[i][k] + dist[k][j]

print("Distance matrix after applying Floyd's Algorithm:")

print(dist)

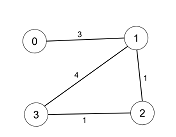
return dist

n = 4

edges = [[0, 1, 3], [1, 2, 1], [1, 3, 4], [2, 3, 1]]

distance\_matrix = floyd\_warshall(n, edges)

Input: n = 4, edges = [[0,1,3],[1,2,1],[1,3,4],[2,3,1]], distanceThreshold = 4

Output: 3