**FIND MAXIMUM AND MINIMUM:**

**1.Write a Program to find both the maximum and minimum values in the array. Implement using any programming language of your choice. Execute your code and provide the maximum and minimum values found.**

**Input : N= 8, a[] = {5,7,3,4,9,12,6,2} Output : Min = 2, Max = 12 Test Cases :**

**Input : N= 9, a[] = {1,3,5,7,9,11,13,15,17} Output : Min = 1, Max = 17 Test Cases :**

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

**Output : Min 12, Max 67 PROGRAM:** def find\_min\_max(arr): if len(arr) == 0:

return None, None

min\_val = arr[0] max\_val = arr[0]

for num in arr: if num < min\_val: min\_val = num if num > max\_val: max\_val = num

return min\_val, max\_val

test\_cases = [

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

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

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

]

results = [find\_min\_max(case) for case in test\_cases] print(results)

**2. Consider an array of integers sorted in ascending order: 2,4,6,8,10,12,14,18.**

**Write a Program to find both the maximum and minimum values in the array. Implement using any programming language of your choice. Execute your code and provide the maximum and minimum values found.**

**Input : N=8, 2,4,6,8,10,12,14,18. Output : Min = 2, Max =18 Test Cases :**

**Input : N= 9, a[] = {11,13,15,17,19,21,23,35,37} Output : Min = 11, Max = 37 Test Cases :**

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

def find\_min\_max\_sorted(arr): if len(arr) == 0:

return None, None

min\_val = arr[0] max\_val = arr[-1]

return min\_val, max\_val

test\_cases\_sorted = [

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

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

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

]

results\_sorted = [find\_min\_max\_sorted(case) for case in test\_cases\_sorted] print(results\_sorted)

**MERGE SORT:**

**1.You are given an unsorted array 31,23,35,27,11,21,15,28. Write a program for Merge Sort and implement using any programming language of your choice.**

**Test Cases :**

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

**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 PROGRAM:** 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

return arr

# Test cases test\_cases\_merge\_sort = [

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

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

]

results\_merge\_sort = [merge\_sort(case) for case in test\_cases\_merge\_sort] print(results\_merge\_sort)

**2.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.**

**Test Cases :**

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

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

**Output : 3,9,10,27,38,43,82 Program:**

def merge\_sort\_with\_comparisons(arr):

comparisons = 0

def merge\_sort(arr):

nonlocal comparisons 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):

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 arr

sorted\_array = merge\_sort(arr) return sorted\_array, comparisons

# Test cases test\_cases\_comparisons = [ [12, 4, 78, 23, 45, 67, 89, 1],

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

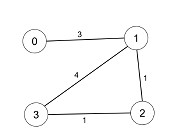
]

for case in test\_cases\_comparisons: sorted\_array, comparison\_count = merge\_sort\_with\_comparisons(case) print(f"Sorted array: {sorted\_array}")

print(f"Number of comparisons: {comparison\_count}")

**ALL PAIR SHORTEST PATH: FLOYDS ALGORITHM**

**1. Implement Floyd's Algorithm to find the shortest path between all pairs of cities. Display the distance matrix before and after applying the algorithm. Identify and print the shortest path**



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

Output: 3

Explanation: The figure above describes the graph.

The neighboring cities at a distanceThreshold = 4 for each city are:

City 0 -> [City 1, City 2]

City 1 -> [City 0, City 2, City 3]

City 2 -> [City 0, City 1, City 3]

City 3 -> [City 1, City 2]

Cities 0 and 3 have 2 neighboring cities at a distanceThreshold = 4, but we have to return city 3 since it has the greatest number.

**Test cases :**

a) You are given a small network of 4 cities connected by roads with the following distances:

City 1 to City 2: 3

City 1 to City 3: 8

City 1 to City 4: -4

City 2 to City 4: 1

City 2 to City 3: 4

City 3 to City 1: 2

City 4 to City 3: -5

City 4 to City 2: 6

Implement Floyd's Algorithm to find the shortest path between all pairs of cities. Display the distance matrix before and after applying the algorithm. Identify and print the shortest path from City 1 to City 3.

Input as above

Output : City 1 to City 3 = -9

b. Consider a network with 6 routers. The initial routing table is as follows:

Router A to Router B: 1 Router A to Router C: 5

Router B to Router C: 2

Router B to Router D: 1

Router C to Router E: 3

Router D to Router E: 1

Router D to Router F: 6

Router E to Router F: 2

Write a Program to implement Floyd's Algorithm to calculate the shortest paths between all pairs of routers. Simulate a change where the link between Router B and Router D fails. Update the distance matrix accordingly. Display the shortest path from Router A to Router F before and after the link failure.

Input as above

Output : Router A to Router F = 5 **Program:** import sys

def print\_matrix(matrix): for row in matrix:

print(" ".join(map(lambda x: f"{x:5}", row))) print()

def floyd\_warshall(n, edges):

dist = [[sys.maxsize] \* n for \_ in range(n)]

for i in range(n): dist[i][i] = 0

for u, v, w in edges:

dist[u][v] = w

print("Initial Distance Matrix:") print\_matrix(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 Floyd-Warshall Algorithm:") print\_matrix(dist)

return dist

def find\_shortest\_path(matrix, start, end):

return matrix[start][end]

# Test Case a)

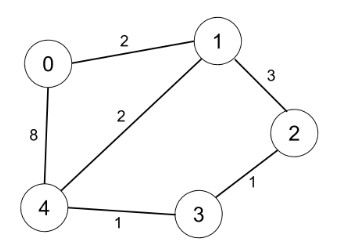
n = 4 edges = [[0, 1, 3], [0, 2, 8], [0, 3, -4], [1, 3, 1], [1, 2, 4], [2, 0, 2], [3, 2, -5], [3, 1, 6]] dist\_matrix = floyd\_warshall(n, edges) shortest\_path\_1\_to\_3 = find\_shortest\_path(dist\_matrix, 0, 2)

print(f"The shortest path from City 1 to City 3 is {shortest\_path\_1\_to\_3}\n")

# Test Case b) n = 6 edges = [[0, 1, 1], [0, 2, 5], [1, 2, 2], [1, 3, 1], [2, 4, 3], [3, 4, 1], [3, 5, 6], [4, 5, 2]] dist\_matrix\_before\_failure = floyd\_warshall(n, edges) shortest\_path\_A\_to\_F\_before = find\_shortest\_path(dist\_matrix\_before\_failure, 0, 5)

|  |  |  |
| --- | --- | --- |
| print(f"The shortest path from Router A to Router F before link  {shortest\_path\_A\_to\_F\_before}\n")    # Simulating link failure between Router B and Router D edges\_with\_failure = [[0, 1, 1], [0, 2, 5], [1, 2, 2], [2, 4, 3], [3, 4, 1], [3, 5, 6], [4, 5, 2]] dist\_matrix\_after\_failure = floyd\_warshall(n, edges\_with\_failure) shortest\_path\_A\_to\_F\_after = find\_shortest\_path(dist\_matrix\_after\_failure, 0, 5) | failure | is |
| print(f"The shortest path from Router A to Router F after link  {shortest\_path\_A\_to\_F\_after}\n") | failure | is |

**2.Implement Floyd's Algorithm to find the shortest path between all pairs of cities. Display the distance matrix before and after applying the algorithm. Identify and print the shortest path**



Input: n = 5, edges = [[0,1,2],[0,4,8],[1,2,3],[1,4,2],[2,3,1],[3,4,1]], distanceThreshold = 2 Output: 0

Explanation: The figure above describes the graph.

The neighboring cities at a distanceThreshold = 2 for each city are:

City 0 -> [City 1]

City 1 -> [City 0, City 4]

City 2 -> [City 3, City 4]

City 3 -> [City 2, City 4]

City 4 -> [City 1, City 2, City 3]

The city 0 has 1 neighboring city at a distanceThreshold = 2.

1. Test cases :
   1. to A 2

A TO C 3

* 1. TO D 1
  2. TO A 6

C TO B 7

Find shortest path from C to A

Output = 7

1. Find shortest path from E to C

C TO A 2 A TO B 4

* 1. TO C 1

B TO E 6

E TO A 1

A TO D 5

* 1. TO E 2
  2. TO D 4

D TO C 1

* 1. TO D 3

Output : E to C = 5

**Program:** import sys

def print\_matrix(matrix): for row in matrix:

print(" ".join(map(lambda x: f"{x:5}", row))) print()

def floyd\_warshall(n, edges):

dist = [[sys.maxsize] \* n for \_ in range(n)]

for i in range(n): dist[i][i] = 0

for u, v, w in edges: dist[u][v] = w

print("Initial Distance Matrix:")

print\_matrix(dist)

for k in range(n): for i in range(n): for j in range(n): if dist[i][k] != sys.maxsize and dist[k][j] != sys.maxsize and dist[i][j] > dist[i][k] + dist[k][j]:

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

print("Distance Matrix after Floyd-Warshall Algorithm:") print\_matrix(dist)

return dist

def find\_shortest\_path(matrix, start, end):

return matrix[start][end]

n = 5 edges = [[0, 1, 2], [0, 4, 8], [1, 2, 3], [1, 4, 2], [2, 3, 1], [3, 4, 1]] dist\_matrix = floyd\_warshall(n, edges)

print(f"The shortest path from City 2 to City 0 is {find\_shortest\_path(dist\_matrix, 2, 0)}") print(f"The shortest path from City 4 to City 2 is {find\_shortest\_path(dist\_matrix, 4, 2)}")

# Additional Test Case a) edges\_a = [[1, 0, 2], [0, 2, 3], [2, 3, 1], [3, 0, 6], [2, 1, 7]] n\_a = 4 dist\_matrix\_a = floyd\_warshall(n\_a, edges\_a) shortest\_path\_C\_to\_A = find\_shortest\_path(dist\_matrix\_a, 2, 0) print(f"The shortest path from City C to City A is {shortest\_path\_C\_to\_A}")

# Additional Test Case b) edges\_b = [[2, 0, 2], [0, 1, 4], [1, 2, 1], [1, 4, 6], [4, 0, 1], [0, 3, 5], [3, 4, 2], [4, 3, 4], [3, 2, 1], [2, 3, 3]] n\_b = 5

dist\_matrix\_b = floyd\_warshall(n\_b, edges\_b) shortest\_path\_E\_to\_C = find\_shortest\_path(dist\_matrix\_b, 4, 2) print(f"The shortest path from City E to City C is {shortest\_path\_E\_to\_C}")