Finding Maximum and Minimum

Problem 1

Write a program to find both the maximum and minimum values in the array.

```
python
Copy code
def find_min_max(arr):
    min_val = float('inf')
    max_val = float('-inf')

    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
arr1 = [5, 7, 3, 4, 9, 12, 6, 2]
print(find_min_max(arr1))  # Output: (2, 12)

arr2 = [1, 3, 5, 7, 9, 11, 13, 15, 17]
print(find_min_max(arr2))  # Output: (1, 17)

arr3 = [22, 34, 35, 36, 43, 67, 12, 13, 15, 17]
print(find_min_max(arr3))  # Output: (12, 67)
```

Problem 2

Consider an array of integers sorted in ascending order. Write a program to find both the maximum and minimum values in the array.

```
python
Copy code
def find_min_max_sorted(arr):
    return arr[0], arr[-1]

# Test Cases
arr1 = [2, 4, 6, 8, 10, 12, 14, 18]
print(find_min_max_sorted(arr1))  # Output: (2, 18)

arr2 = [11, 13, 15, 17, 19, 21, 23, 35, 37]
print(find_min_max_sorted(arr2))  # Output: (11, 37)

arr3 = [22, 34, 35, 36, 43, 67, 12, 13, 15, 17]
print(find_min_max_sorted(arr3))  # Output: (12, 67)
```

Merge Sort

Problem 1

Implement Merge Sort.

```
python
Copy code
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

# Test Cases
arr1 = [31, 23, 35, 27, 11, 21, 15, 28]
merge_sort(arr1)
print(arr1)  # Output: [11, 15, 21, 23, 27, 28, 31, 35]

arr2 = [22, 34, 25, 36, 43, 67, 52, 13, 65, 17]
merge_sort(arr2)
print(arr2)  # Output: [13, 17, 22, 25, 34, 36, 43, 52, 65, 67]
```

Problem 2

Implement Merge Sort and count the number of comparisons.

```
python
Copy code
def merge_sort_with_count(arr):
    count = [0]

    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):
                count[0] += 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

    merge_sort(arr)
    return count[0]

# Test Cases
arr1 = [12, 4, 78, 23, 45, 67, 89, 1]
comparisons = merge_sort_with_count(arr1)
print(arr1, comparisons)  # Output: [1, 4, 12, 23, 45, 67, 78, 89],
comparisons count

arr2 = [38, 27, 43, 3, 9, 82, 10]
comparisons = merge_sort_with_count(arr2)
print(arr2, comparisons)  # Output: [3, 9, 10, 27, 38, 43, 82], comparisons
count
```

Floyd's Algorithm

Problem 1

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

```
python
Copy code
def floyd_warshall(n, edges):
    dist = [[float('inf')] * n for _ in range(n)]
    for i in range(n):
        dist[i][i] = 0

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

    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]

    return dist
```

```
# Test Case
n = 4
edges = [[0, 1, 3], [1, 2, 1], [1, 3, 4], [2, 3, 1]]
dist_matrix = floyd_warshall(n, edges)
for row in dist_matrix:
    print(row)
```

Example: City 1 to City 3

```
python
Copy code
# Additional test case data
n = 4
edges = [[0, 1, 3], [0, 2, 8], [0, 3, -4], [1, 3, 1], [1, 2, 4], [2, 0, 2],
[
```

4o
next
ChatGPT

Sure! Let's continue with the implementation for the remaining tasks.

Floyd's Algorithm (continued)

Problem 1 (continued)

Example: City 1 to City 3 with the provided edges

```
python
Copy code
# Example: City 1 to City 3
def floyd_warshall_example():
    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 = [[float('inf')] * 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("Distance matrix before applying Floyd's Algorithm:")
    for row in dist:
        print(row)

    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("\nDistance matrix after applying Floyd's Algorithm:")
    for row in dist:
        print(row)

    # Print shortest path from City 1 to City 3 (0-based index)
    print("\nShortest path from City 1 to City 3:", dist[0][2])
```

```
floyd_warshall_example()
```

Problem 1 (continued)

Example: Network with 6 routers

python
Copy code

```
# Example: Network with 6 routers
def floyd_warshall_routers():
    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 = [[float('inf')] * 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("Distance matrix before applying Floyd's Algorithm:")
    for row in dist:
        print(row)

    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("\nDistance matrix after applying Floyd's Algorithm:")
    for row in dist:
        print(row)

    print("\nShortest path from Router A to Router F:", dist[0][5])

    # Simulate the failure of the link between Router B and Router D
    dist[1][3] = float('inf')

    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("\nDistance matrix after the link failure between Router B and Router D:")
    for row in dist:
        print(row)

    print("\nShortest path from Router A to Router F after link failure:", dist[0][5])

floyd_warshall_routers()
```

Problem 2

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

```
python
Copy code
def floyd_warshall(n, edges):
    dist = [[float('inf')] * 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("Distance matrix before applying Floyd's Algorithm:")
    for row in dist:
        print(row)

    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("\nDistance matrix after applying Floyd's Algorithm:")
    for row in dist:
        print(row)

    return dist

# Test Case: Find shortest path from C to A and E to C
edges1 = [[1, 0, 2], [0, 2, 3], [2, 3, 1], [3, 0, 6], [2, 1, 7]]
dist_matrix1 = floyd_warshall(4, edges1)
print("\nShortest path from C to A:", dist_matrix1[2][0])

edges2 = [[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]]
dist_matrix2 = floyd_warshall(5, edges2)
print("\nShortest path from E to C:", dist_matrix2[4][2])
```