1.MAXIMUM AND MINIMUM

```
a=[1,3,5,7,9,11,13,15,17]
def find_max_min(a):
    min_val=float('inf')
    max_val=float('-inf')
    for num in a:
        if num<min_val:
            min_val=num
        if num>max_val:
            max_val=num
    return min_val,max_val
min_val,max_val=find_max_min(a)
print(f"Max={max_val},Min={min_val}")
```

```
2. array = [2, 4, 6, 8, 10, 12, 14, 18]
min_value = min(array)
max_value = max(array)
print(f"Min = {min_value}, Max = {max_value}")
```

3.MERGE SORT

```
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        L = arr[:mid]
        R = arr[mid:]
        merge_sort(L)
        merge_sort(R)
        i = j = k = 0
        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
        while i < len(L):
            arr[k] = L[i]
            i += 1
            k += 1

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

def print_list(arr):
    for i in range(len(arr)):
        print(arr[i], end=" ")
    print()
if __name__ == "__main__":
    arr = [31, 23, 35, 27, 11, 21, 15, 28]
    print("Given array is", end="\n")
    print_list(arr)
    merge_sort(arr)
    print("Sorted array is", end="\n")
    print_list(arr)
```

4.
```
def merge_sort(arr):
    comparisons = 0  # Initialize comparison counter

    def merge_sort_recursive(arr):
        nonlocal comparisons
        if len(arr) > 1:
```

```
        mid = len(arr) // 2
        L = arr[:mid]
        R = arr[mid:]
        comparisons += merge_sort_recursive(L)
        comparisons += merge_sort_recursive(R)

        i = j = k = 0
        while i < len(L) and j < len(R):
            comparisons += 1
            if L[i] < R[j]:
                arr[k] = L[i]
                i += 1
            else:
                arr[k] = R[j]
                j += 1
            k += 1
        while i < len(L):
            arr[k] = L[i]
            i += 1
            k += 1

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

        return comparisons

    total_comparisons = merge_sort_recursive(arr)
    return total_comparisons

def print_list(arr):
```

```
    for i in range(len(arr)):
        print(arr[i], end=" ")
    print()
if __name__ == "__main__":
    arr = [12, 4, 78, 23, 45, 67, 89, 1]
    print("Given array is")
    print_list(arr)
    total_comparisons = merge_sort(arr)
    print("Sorted array is")
    print_list(arr)
    print(f"Total number of comparisons: {total_comparisons}")
```

5. All Pairs Shortest Paths: Floyd's Algorithm

```
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
        dist[v][u] = w
    print("Initial distance matrix:")
    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 Floyd-Warshall algorithm:")
    for row in dist:
        print(row)

    return dist
```

```
def print_shortest_path(dist):
    shortest_path_length = float('inf')
    shortest_path = None
    for i in range(len(dist)):
        for j in range(len(dist)):
            if i != j and dist[i][j] < shortest_path_length:
                shortest_path_length = dist[i][j]
                shortest_path = (i, j)
    print(f"\nShortest path is between node {shortest_path[0]} and node {shortest_path[1]} with a distance of {shortest_path_length}")
if __name__ == "__main__":
    n = 4
    edges = [[0, 1, 3], [1, 2, 1], [1, 3, 4], [2, 3, 1]]
    dist = floyd_warshall(n, edges)
    print_shortest_path(dist)
```

6.
```
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("Initial distance matrix:")
    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 Floyd-Warshall algorithm:")
    for row in dist:
        print(row)
```

```
    return dist
def print_shortest_path(dist):
    source = 0  # City 1 (index 0)
    destination = 2  # City 3 (index 2)
    print(f"\nShortest path from City 1 to City 3: {dist[source][destination]}")
if __name__ == "__main__":
    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 = floyd_warshall(n, edges)
    print_shortest_path(dist)
```

7.
```
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("Initial distance matrix:")
    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 Floyd-Warshall algorithm:")
    for row in dist:
        print(row)

    return dist
def print_shortest_path(dist):
    source = 0
    destination = 5
    print(f"\nShortest path from Router A to Router F: {dist[source][destination]}")
if __name__ == "__main__":
    n = 6
    edges = [
        [0, 1, 1],  # Router A to Router B: 1
        [0, 2, 5],  # Router A to Router C: 5
        [1, 2, 2],
        [1, 3, 1],
        [2, 4, 3],
        [3, 4, 1],
        [3, 5, 6],
        [4, 5, 2]
    ]
    dist = floyd_warshall(n, edges)
    print_shortest_path(dist)
    edges = [
        [0, 1, 1],
        [0, 2, 5],
        [1, 2, 2],
```

```
        [2, 4, 3],
        [3, 4, 1],
        [3, 5, 6],
        [4, 5, 2]
    ]
    dist = floyd_warshall(n, edges)
    print_shortest_path(dist)
8. def floyd_warshall_with_threshold(n, edges, distanceThreshold):
    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
        dist[v][u] = w  # Assuming it's an undirected graph
    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

def find_neighbors_within_threshold(dist, distanceThreshold):
    neighbors = {}
    for i in range(len(dist)):
        neighbors[i] = [j for j in range(len(dist)) if i != j and dist[i][j] <= distanceThreshold]
    return neighbors
if __name__ == "__main__":
    n = 5
    edges = [
        [0, 1, 2],
        [0, 4, 8],
```

```
    [1, 2, 3],
    [1, 4, 2],
    [2, 3, 1],
    [3, 4, 1]
]
distanceThreshold = 2
dist = floyd_warshall_with_threshold(n, edges, distanceThreshold)

print("\nDistance matrix after Floyd-Warshall algorithm:")
for row in dist:
    print(row)

neighbors = find_neighbors_within_threshold(dist, distanceThreshold)
for city in neighbors:
    print(f'City {city} -> {neighbors[city]}")
```