8. Optimize the flow of the goods through a supply chain, minimizing transportation costs, inventory, holding costs, and lead timers while ensuring timely delivery and resource utilization

Solution:

```
import math, random

# ---------- Distance Matrix ----------

def compute_distance_matrix(cities):

    n = len(cities)

    dist = [[0]*n for _ in range(n)]

    for i in range(n):

        for j in range(n):

            if i != j:

                dist[i][j] = math.dist(cities[i], cities[j])

    return dist


# ---------- Route Length ----------

def route_length(route, dist):

    n = len(route)

    return sum(dist[route[i]][route[(i+1)%n]] for i in range(n))


# ---------- Lévy Flight ----------

def levy_flight(Lambda):

    # Mantegna's algorithm for Lévy flight

    sigma = (math.gamma(1 + Lambda) * math.sin(math.pi * Lambda / 2) /

             (math.gamma((1 + Lambda) / 2) * Lambda * 2 ** ((Lambda - 1) / 2))) ** (1 /
Lambda)

    u = random.gauss(0, sigma)

    v = abs(random.gauss(0, 1)) ** (1 / Lambda)

    step = u / v

    return step
```

```
# ---------- Cuckoo Search for TSP ----------
def Cuckoo_TSP(cities, n_nests=15, n_iterations=100, pa=0.25, Lambda=1.5):
    n = len(cities)
    dist = compute_distance_matrix(cities)

    # Initialize nests with random routes
    nests = [random.sample(range(n), n) for _ in range(n_nests)]
    fitness = [route_length(route, dist) for route in nests]

    best_index = min(range(n_nests), key=lambda i: fitness[i])
    best_route = nests[best_index][:]
    best_length = fitness[best_index]

    for _ in range(n_iterations):
        for i in range(n_nests):
            # Generate new route using Lévy flight-style random swaps
            new_route = nests[i][:]
            step_size = int(abs(levy_flight(Lambda)) * n) % n
            for _ in range(step_size):
                a, b = random.sample(range(n), 2)
                new_route[a], new_route[b] = new_route[b], new_route[a]

            new_fitness = route_length(new_route, dist)

            # Replace if better
            if new_fitness < fitness[i]:
                nests[i], fitness[i] = new_route, new_fitness

        # Abandon a fraction of worse nests (discovery probability pa)
        n_abandon = int(pa * n_nests)
```

```
        worst_indices = sorted(range(n_nests), key=lambda i: fitness[i],
reverse=True)[:n_abandon]
        for i in worst_indices:
            nests[i] = random.sample(range(n), n)
            fitness[i] = route_length(nests[i], dist)

        # Update best
        current_best_index = min(range(n_nests), key=lambda i: fitness[i])
        if fitness[current_best_index] < best_length:
            best_length = fitness[current_best_index]
            best_route = nests[current_best_index][:]

    return best_route, best_length
# ---------- Example ----------
cities = [(0,0), (1,5), (5,2), (6,6), (8,3)]
best_route, best_length = Cuckoo_TSP(cities)

print("Best Route (city indices):", best_route)
print("Best Route Length:", best_length)
```

Output:

```
Best Route (city indices): [4, 3, 1, 0, 2]
Best Route Length: 22.35103276995244
```