LAB 3:

ANT COLONY OPTIMIZATION:

```python
#LAB 3
#Ant Colony Optimization

import numpy as np
import random
```

```python
class SimpleACO:
    def __init__(self, cities, n_ants=10, n_iterations=50, decay=0.1, alpha=1, beta=2):
        self.cities = cities                        # List of city coordinates
        self.n_ants = n_ants                        # Number of ants
        self.n_iterations = n_iterations            # Number of iterations
        self.decay = decay                          # Pheromone evaporation rate
        self.alpha = alpha                          # Importance of pheromone
        self.beta = beta                            # Importance of distance (heuristic)
        self.n_cities = len(cities)                 # Number of cities
        self.distances = self.calculate_distances() # Distance matrix between cities
        self.pheromones = np.ones((self.n_cities, self.n_cities)) # Pheromone levels
```

```python
        # Print input parameters
        print("ACO Algorithm Parameters:")
        print(f"Cities (coordinates): {self.cities}")
        print(f"Number of Ants: {self.n_ants}")
        print(f"Number of Iterations: {self.n_iterations}")
        print(f"Pheromone Decay Rate: {self.decay}")
        print(f"Alpha (Pheromone Importance): {self.alpha}")
        print(f"Beta (Distance Importance): {self.beta}")
        print()
```

```python
    def calculate_distances(self):
        # Calculate the Euclidean distance between each pair of cities
        distances = np.zeros((self.n_cities, self.n_cities))
        for i in range(self.n_cities):
            for j in range(i + 1, self.n_cities):
                distances[i][j] = distances[j][i] = np.linalg.norm(np.array(self.cities[i]) - 
np.array(self.cities[j]))
        return distances
```

```python
    def run(self):
        best_route = None
        best_distance = float('inf')
```

```python
        for _ in range(self.n_iterations):
            all_routes = []
            all_distances = []
```

```python
            # Each ant constructs a route
            for _ in range(self.n_ants):
                route = self.construct_route()
                distance = self.calculate_route_distance(route)
                all_routes.append(route)
                all_distances.append(distance)
```

```python
                # Track the best route found
                if distance < best_distance:
                    best_route = route
                    best_distance = distance
```

```python
            # Update pheromones based on all routes
            self.update_pheromones(all_routes, all_distances)

        return best_route, best_distance

    def construct_route(self):
        # Start from a random city and build the route
        route = [random.randint(0, self.n_cities - 1)]
        while len(route) < self.n_cities:
            current_city = route[-1]
            next_city = self.choose_next_city(current_city, route)
            route.append(next_city)
        return route

    def choose_next_city(self, current_city, route):
        # Calculate the probability of moving to each unvisited city
        probabilities = []
        for city in range(self.n_cities):
            if city not in route:
                pheromone = self.pheromones[current_city][city] ** self.alpha
                heuristic = (1 / self.distances[current_city][city]) ** self.beta
                probabilities.append((city, pheromone * heuristic))

        # Normalize probabilities and choose the next city
        total = sum(prob for _, prob in probabilities)
        probabilities = [(city, prob / total) for city, prob in probabilities]
        r = random.random()
        cumulative_prob = 0
        for city, prob in probabilities:
            cumulative_prob += prob
            if r <= cumulative_prob:
                return city

    def calculate_route_distance(self, route):
        # Calculate the total distance for the route
        distance = sum(self.distances[route[i]][route[i + 1]] for i in range(len(route) - 1))
        distance += self.distances[route[-1]][route[0]]  # Return to start
        return distance

    def update_pheromones(self, all_routes, all_distances):
        # Evaporate pheromones
        self.pheromones *= (1 - self.decay)

        # Add new pheromones based on the routes taken
        for route, distance in zip(all_routes, all_distances):
            pheromone_contribution = 1 / distance
            for i in range(len(route) - 1):
                self.pheromones[route[i]][route[i + 1]] += pheromone_contribution
                self.pheromones[route[i + 1]][route[i]] += pheromone_contribution
            self.pheromones[route[-1]][route[0]] += pheromone_contribution
            self.pheromones[route[0]][route[-1]] += pheromone_contribution

# Example usage
cities = [(0, 0), (2, 3), (5, 5), (8, 2), (7, 7)]
aco = SimpleACO(cities)
best_route, best_distance = aco.run()
print("Best Route:", best_route)
print("Best Distance:", best_distance)
```

OUTPUT:

```
ACO Algorithm Parameters:
Cities (coordinates): [(0, 0), (2, 3), (5, 5), (8, 2), (7, 7)]
Number of Ants: 10
Number of Iterations: 50
Pheromone Decay Rate: 0.1
Alpha (Pheromone Importance): 1
Beta (Distance Importance): 2

Best Route: [2, 4, 3, 0, 1]
Best Distance: 23.38476044050227
```