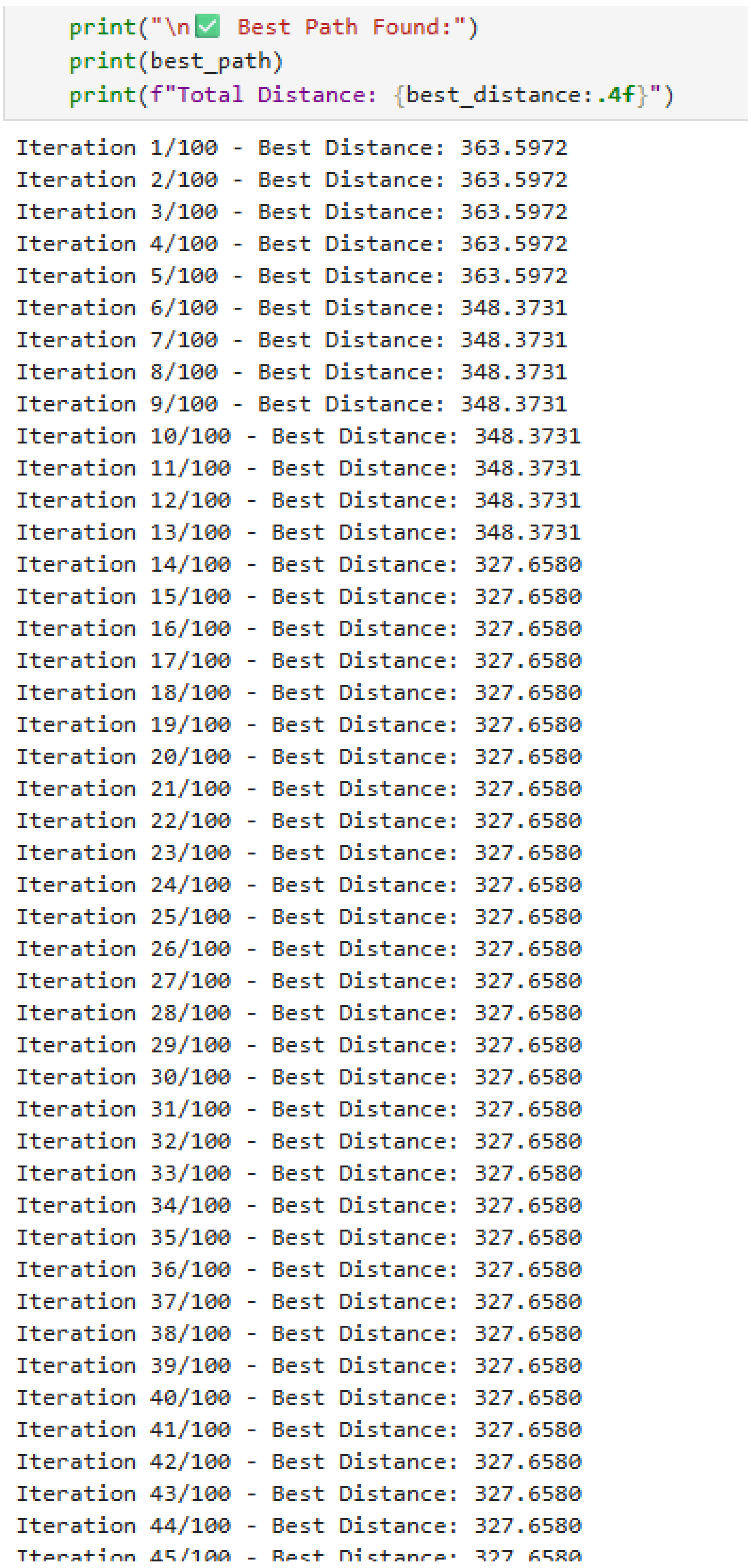
Lab 5

Cuckoo search

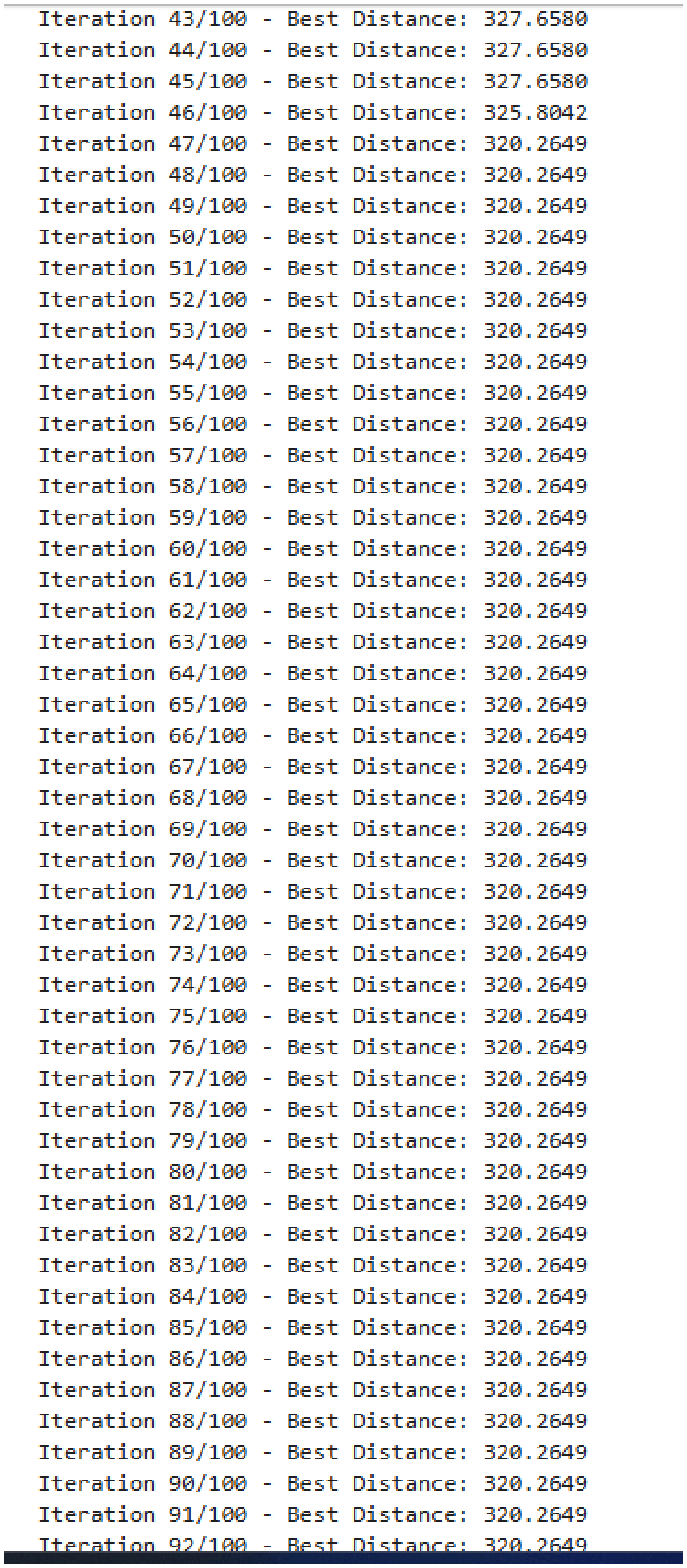
```python
import random
import math

# ------------------------------
# Helper functions
# ------------------------------

def distance(city1, city2):
    """Euclidean distance between two cities"""
    return math.sqrt((city1[0] - city2[0])**2 + (city1[1] - city2[1])**2)

def total_distance(path, cities):
    """Compute total distance of the path"""
    dist = 0
    for i in range(len(path)):
        city1 = cities[path[i]]
        city2 = cities[path[(i + 1) % len(path)]]
        dist += distance(city1, city2)
    return dist

def levy_flight(Lambda):
    """Generate step size using Lévy distribution"""
    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 = random.gauss(0, 1)
    step = u / abs(v)**(1 / Lambda)
    return step

def get_new_solution(path):
    """Generate a new random permutation by swapping two cities"""
    new_path = path[:]
    i, j = random.sample(range(len(new_path)), 2)
    new_path[i], new_path[j] = new_path[j], new_path[i]
    return new_path

# ------------------------------
# Cuckoo Search Algorithm
# ------------------------------

def cuckoo_search_tsp(cities, n_nests=15, pa=0.25, alpha=1, Lambda=1.5, max_iter=100):
    # Initialize nests (random permutations)
    return new_path

# -------------------------------
# Cuckoo Search Algorithm
# -------------------------------

def cuckoo_search_tsp(cities, n_nests=15, pa=0.25, alpha=1, Lambda=1.5, max_iter=100):
    # Initialize nests (random permutations)
    nests = [random.sample(range(len(cities)), len(cities)) for _ in range(n_nests)]
    fitness = [total_distance(nest, cities) for nest in nests]

    best_nest = nests[fitness.index(min(fitness))]
    best_fitness = min(fitness)

    for iteration in range(max_iter):
        for i in range(n_nests):
            # Lévy flight (small random move)
            step_size = levy_flight(Lambda)
            new_nest = get_new_solution(nests[i])

            new_fitness = total_distance(new_nest, cities)

            # If better, replace
            if new_fitness < fitness[i]:
                nests[i] = new_nest
                fitness[i] = new_fitness

            # Update best
            if new_fitness < best_fitness:
                best_nest = new_nest[:]
                best_fitness = new_fitness

        # Abandon some nests with probability pa
        for i in range(n_nests):
            if random.random() < pa:
                nests[i] = random.sample(range(len(cities)), len(cities))
                fitness[i] = total_distance(nests[i], cities)

        print(f"Iteration {iteration+1}/{max_iter} - Best Distance: {best_fitness:.4f}")

    return best_nest, best_fitness

# -------------------------------
# Example Run
# -------------------------------
```

```python
            if new_fitness < best_fitness:
                best_nest = new_nest[:]
                best_fitness = new_fitness

        # Abandon some nests with probability pa
        for i in range(n_nests):
            if random.random() < pa:
                nests[i] = random.sample(range(len(cities)), len(cities))
                fitness[i] = total_distance(nests[i], cities)

        print(f"Iteration {iteration+1}/{max_iter} - Best Distance: {best_fitness:.4f}")

    return best_nest, best_fitness

# ------------------------------
# Example Run
# ------------------------------

if __name__ == "__main__":
    # Example: 10 cities with random coordinates
    cities = [(random.uniform(0, 100), random.uniform(0, 100)) for _ in range(10)]

    best_path, best_distance = cuckoo_search_tsp(cities, n_nests=20, max_iter=100)

    print("\n✅ Best Path Found:")
    print(best_path)
    print(f"Total Distance: {best_distance:.4f}")
```



```
    print("\n✅ Best Path Found:")
    print(best_path)
    print(f"Total Distance: {best_distance:.4f}")
```

```
Iteration 1/100 - Best Distance: 363.5972
Iteration 2/100 - Best Distance: 363.5972
Iteration 3/100 - Best Distance: 363.5972
Iteration 4/100 - Best Distance: 363.5972
Iteration 5/100 - Best Distance: 363.5972
Iteration 6/100 - Best Distance: 348.3731
Iteration 7/100 - Best Distance: 348.3731
Iteration 8/100 - Best Distance: 348.3731
Iteration 9/100 - Best Distance: 348.3731
Iteration 10/100 - Best Distance: 348.3731
Iteration 11/100 - Best Distance: 348.3731
Iteration 12/100 - Best Distance: 348.3731
Iteration 13/100 - Best Distance: 348.3731
Iteration 14/100 - Best Distance: 327.6580
Iteration 15/100 - Best Distance: 327.6580
Iteration 16/100 - Best Distance: 327.6580
Iteration 17/100 - Best Distance: 327.6580
Iteration 18/100 - Best Distance: 327.6580
Iteration 19/100 - Best Distance: 327.6580
Iteration 20/100 - Best Distance: 327.6580
Iteration 21/100 - Best Distance: 327.6580
Iteration 22/100 - Best Distance: 327.6580
Iteration 23/100 - Best Distance: 327.6580
Iteration 24/100 - Best Distance: 327.6580
Iteration 25/100 - Best Distance: 327.6580
Iteration 26/100 - Best Distance: 327.6580
Iteration 27/100 - Best Distance: 327.6580
Iteration 28/100 - Best Distance: 327.6580
Iteration 29/100 - Best Distance: 327.6580
Iteration 30/100 - Best Distance: 327.6580
Iteration 31/100 - Best Distance: 327.6580
Iteration 32/100 - Best Distance: 327.6580
Iteration 33/100 - Best Distance: 327.6580
Iteration 34/100 - Best Distance: 327.6580
Iteration 35/100 - Best Distance: 327.6580
Iteration 36/100 - Best Distance: 327.6580
Iteration 37/100 - Best Distance: 327.6580
Iteration 38/100 - Best Distance: 327.6580
Iteration 39/100 - Best Distance: 327.6580
Iteration 40/100 - Best Distance: 327.6580
Iteration 41/100 - Best Distance: 327.6580
Iteration 42/100 - Best Distance: 327.6580
Iteration 43/100 - Best Distance: 327.6580
Iteration 44/100 - Best Distance: 327.6580
Iteration 45/100 - Best Distance: 327.6580
```



```
Iteration 43/100 - Best Distance: 327.6580
Iteration 44/100 - Best Distance: 327.6580
Iteration 45/100 - Best Distance: 327.6580
Iteration 46/100 - Best Distance: 325.8042
Iteration 47/100 - Best Distance: 320.2649
Iteration 48/100 - Best Distance: 320.2649
Iteration 49/100 - Best Distance: 320.2649
Iteration 50/100 - Best Distance: 320.2649
Iteration 51/100 - Best Distance: 320.2649
Iteration 52/100 - Best Distance: 320.2649
Iteration 53/100 - Best Distance: 320.2649
Iteration 54/100 - Best Distance: 320.2649
Iteration 55/100 - Best Distance: 320.2649
Iteration 56/100 - Best Distance: 320.2649
Iteration 57/100 - Best Distance: 320.2649
Iteration 58/100 - Best Distance: 320.2649
Iteration 59/100 - Best Distance: 320.2649
Iteration 60/100 - Best Distance: 320.2649
Iteration 61/100 - Best Distance: 320.2649
Iteration 62/100 - Best Distance: 320.2649
Iteration 63/100 - Best Distance: 320.2649
Iteration 64/100 - Best Distance: 320.2649
Iteration 65/100 - Best Distance: 320.2649
Iteration 66/100 - Best Distance: 320.2649
Iteration 67/100 - Best Distance: 320.2649
Iteration 68/100 - Best Distance: 320.2649
Iteration 69/100 - Best Distance: 320.2649
Iteration 70/100 - Best Distance: 320.2649
Iteration 71/100 - Best Distance: 320.2649
Iteration 72/100 - Best Distance: 320.2649
Iteration 73/100 - Best Distance: 320.2649
Iteration 74/100 - Best Distance: 320.2649
Iteration 75/100 - Best Distance: 320.2649
Iteration 76/100 - Best Distance: 320.2649
Iteration 77/100 - Best Distance: 320.2649
Iteration 78/100 - Best Distance: 320.2649
Iteration 79/100 - Best Distance: 320.2649
Iteration 80/100 - Best Distance: 320.2649
Iteration 81/100 - Best Distance: 320.2649
Iteration 82/100 - Best Distance: 320.2649
Iteration 83/100 - Best Distance: 320.2649
Iteration 84/100 - Best Distance: 320.2649
Iteration 85/100 - Best Distance: 320.2649
Iteration 86/100 - Best Distance: 320.2649
Iteration 87/100 - Best Distance: 320.2649
Iteration 88/100 - Best Distance: 320.2649
Iteration 89/100 - Best Distance: 320.2649
Iteration 90/100 - Best Distance: 320.2649
Iteration 91/100 - Best Distance: 320.2649
Iteration 92/100 - Best Distance: 320.2649
Iteration 61/100 - Best Distance: 320.2649
Iteration 62/100 - Best Distance: 320.2649
Iteration 63/100 - Best Distance: 320.2649
Iteration 64/100 - Best Distance: 320.2649
Iteration 65/100 - Best Distance: 320.2649
Iteration 66/100 - Best Distance: 320.2649
Iteration 67/100 - Best Distance: 320.2649
Iteration 68/100 - Best Distance: 320.2649
Iteration 69/100 - Best Distance: 320.2649
Iteration 70/100 - Best Distance: 320.2649
Iteration 71/100 - Best Distance: 320.2649
Iteration 72/100 - Best Distance: 320.2649
Iteration 73/100 - Best Distance: 320.2649
Iteration 74/100 - Best Distance: 320.2649
Iteration 75/100 - Best Distance: 320.2649
Iteration 76/100 - Best Distance: 320.2649
Iteration 77/100 - Best Distance: 320.2649
Iteration 78/100 - Best Distance: 320.2649
Iteration 79/100 - Best Distance: 320.2649
Iteration 80/100 - Best Distance: 320.2649
Iteration 81/100 - Best Distance: 320.2649
Iteration 82/100 - Best Distance: 320.2649
Iteration 83/100 - Best Distance: 320.2649
Iteration 84/100 - Best Distance: 320.2649
Iteration 85/100 - Best Distance: 320.2649
Iteration 86/100 - Best Distance: 320.2649
Iteration 87/100 - Best Distance: 320.2649
Iteration 88/100 - Best Distance: 320.2649
Iteration 89/100 - Best Distance: 320.2649
Iteration 90/100 - Best Distance: 320.2649
Iteration 91/100 - Best Distance: 320.2649
Iteration 92/100 - Best Distance: 320.2649
Iteration 93/100 - Best Distance: 320.2649
Iteration 94/100 - Best Distance: 320.2649
Iteration 95/100 - Best Distance: 320.2649
Iteration 96/100 - Best Distance: 320.2649
Iteration 97/100 - Best Distance: 320.2649
Iteration 98/100 - Best Distance: 320.2649
Iteration 99/100 - Best Distance: 320.2649
Iteration 100/100 - Best Distance: 320.2649

✅ Best Path Found:
[1, 8, 3, 5, 4, 0, 6, 9, 7, 2]
Total Distance: 320.2649
```