AI PRACTICAL NO. 10

Name:Girish Nhavkar

Roll no:9560

Div:TE COMPS A (BATCH B)

PART 1:

```
import numpy as np
import random

# Function to calculate the distance between two cities
def distance(city1, city2):
    return np.linalg.norm(city1 - city2)

# Function to calculate the total distance of a route
def total_distance(route, cities):
    total = 0
    for i in range(len(route) - 1):
        total += distance(cities[route[i]], cities[route[i+1]])
    total += distance(cities[route[-1]], cities[route[0]])  # Return to the
starting city
    return total

# Function to initialize the population
def initialize_population(num_routes, num_cities):
    population = []
    for _ in range(num_routes):
        route = list(range(num_cities))
        random.shuffle(route)
        population.append(route)
    return population

# Function to perform tournament selection
def tournament_selection(population, fitness_values, tournament_size):
    selected_parents = []
    for _ in range(len(population)):
        tournament_indices = random.sample(range(len(population)),
tournament_size)
        tournament_fitness = [fitness_values[i] for i in tournament_indices]
        winner_index = tournament_indices[np.argmin(tournament_fitness)]
        selected_parents.append(population[winner_index])
    return selected_parents

# Function to perform ordered crossover
def ordered_crossover(parent1, parent2):
```

```python
    size = len(parent1)
    start = random.randint(0, size - 1)
    end = random.randint(start + 1, size)
    offspring = [None] * size
    for i in range(start, end):
        offspring[i] = parent1[i]
    remaining = [item for item in parent2 if item not in offspring]
    j = 0
    for i in range(size):
        if offspring[i] is None:
            offspring[i] = remaining[j]
            j += 1
    return offspring

# Function to perform mutation
def mutate(route, mutation_rate):
    for i in range(len(route)):
        if random.random() < mutation_rate:
            j = random.randint(0, len(route) - 1)
            route[i], route[j] = route[j], route[i]
    return route

# Function to replace the current population with the offspring
def replace(population, offspring, fitness_values):
    combined_population = list(zip(population, fitness_values))
    combined_population.sort(key=lambda x: x[1])
    combined_population[:len(offspring)] = zip(offspring,
[total_distance(route, cities) for route in offspring])
    combined_population.sort(key=lambda x: x[1])
    new_population, _ = zip(*combined_population)
    return new_population

# Main genetic algorithm function
def genetic_algorithm_TSP(cities, num_routes, max_generations,
tournament_size, mutation_rate):
    num_cities = len(cities)
    population = initialize_population(num_routes, num_cities)
    for generation in range(max_generations):
        fitness_values = [total_distance(route, cities) for route in
population]
        selected_parents = tournament_selection(population, fitness_values,
tournament_size)
        offspring = [ordered_crossover(parent1, parent2) for parent1, parent2
in zip(selected_parents[::2], selected_parents[1::2])]
        offspring = [mutate(route, mutation_rate) for route in offspring]
        population = replace(population, offspring, fitness_values)
    best_route = population[0]
    return best_route
```

```python
# Input from the user
def get_city_coordinates(num_cities):
    cities = []
    for i in range(num_cities):
        x, y = map(int, input(f"Enter coordinates for city {i+1} (format: x
y): ").split())
        cities.append([x, y])
    return np.array(cities)

# Example usage
if __name__ == "__main__":
    # Input number of cities from the user
    num_cities = int(input("Enter the number of cities: "))
    cities = get_city_coordinates(num_cities)

    # Parameters
    num_routes = 50
    max_generations = 1000
    tournament_size = 3
    mutation_rate = 0.01

    # Run genetic algorithm
    best_route = genetic_algorithm_TSP(cities, num_routes, max_generations,
tournament_size, mutation_rate)
    print("Best Route:", best_route)
    print("Total Distance:", total_distance(best_route, cities))
```

op:

```
PS C:\Girish\TE\AI> & "C:/Users/Girish Nhavkar/AppData/Local/Programs/Python/Python312/python.exe" c:/Girish/TE/AI/exp10/exp10.py
Enter the number of cities: 2
Enter coordinates for city 1 (format: x y): 2 3
Enter coordinates for city 2 (format: x y): 4 6
Best Route: [1, 0]
Total Distance: 7.211102550927978
PS C:\Girish\TE\AI>
```

Part 2 :

```
import random
import numpy as np


cities = {
    'A': (4,9),
    'B': (6,9),
    'C': (3,7),
    'D': (70,5),
    'E': (75,3)
}

# Genetic Algorithm parameters
population_size = 50
generations = 1000
mutation_rate = 0.01

def calculate_distance(route):
    total_distance = 0
    for i in range(len(route) - 1):
        city1, city2 = route[i], route[i + 1]
        total_distance += np.linalg.norm(np.array(cities[city1]) -
np.array(cities[city2]))
    return total_distance

def generate_initial_population():
    population = []
    cities_list = list(cities.keys())

    for _ in range(population_size):
        route = random.sample(cities_list, len(cities_list))
        population.append(route)

    return population

def crossover(parent1, parent2):
    crossover_point = random.randint(1, len(parent1) - 1)
    child1 = parent1[:crossover_point] + [city for city in parent2 if city not
in parent1[:crossover_point]]
    child2 = parent2[:crossover_point] + [city for city in parent1 if city not
in parent2[:crossover_point]]

    return child1, child2

def mutate(route):
    if random.random() < mutation_rate:
        mutation_point1, mutation_point2 = random.sample(range(len(route)), 2)
```

```python
        route[mutation_point1], route[mutation_point2] = route[mutation_point2], route[mutation_point1]

    return route

def genetic_algorithm():
    population = generate_initial_population()

    for generation in range(generations):
        population = sorted(population, key=lambda x: calculate_distance(x))
        new_population = []

        for _ in range(int(population_size/2)):
            parent1, parent2 = random.sample(population[:10], 2)  # Select top 10 as parents
            child1, child2 = crossover(parent1, parent2)
            child1 = mutate(child1)
            child2 = mutate(child2)
            new_population.extend([child1, child2])

        population = new_population

    best_route = min(population, key=lambda x: calculate_distance(x))
    min_distance = calculate_distance(best_route)

    print(f"Best Route: {best_route}")
    print(f"Total Distance: {min_distance}")

if __name__ == "__main__":
    genetic_algorithm()
```

op:

```
PS C:\Girish\TE\AI> & "C:/Users/Girish Nhavkar/AppData/Local/Programs/Python/Python312/python.exe" c:/Girish/TE/AI/exp10/exppp10.py
Best Route: ['E', 'D', 'B', 'A', 'C']
Total Distance: 73.74611095215988
PS C:\Girish\TE\AI>
```