```
import random

# Define constants
N_QUEENS = 8
POPULATION_SIZE = 100
GENERATIONS = 1000
MUTATION_RATE = 0.1

# Step 1: Fitness Function
def calculate_fitness(chromosome):
    conflicts = 0
    n = len(chromosome)  # Number of queens (4 in this case)

    # Check each pair of queens
    for i in range(n):
        for j in range(i + 1, n):
            # Same row conflict
            if chromosome[i] == chromosome[j]:
                conflicts += 1

            # Diagonal conflict
            if abs(chromosome[i] - chromosome[j]) == abs(i - j):
                conflicts += 1

    return conflicts  # Lower conflicts means better solution

# Step 2: Generate Random Chromosome
def generate_chromosome():
    # Chromosome: a random permutation of row positions
    return [random.randint(0, N_QUEENS - 1) for _ in range(N_QUEENS)]

# Step 3: Initialize Population
def initialize_population():
    return [generate_chromosome() for _ in range(POPULATION_SIZE)]

# Step 4: Selection
def select_parents(population):
    # Select 2 parents using tournament selection
    tournament_size = 5
    tournament = random.sample(population, tournament_size)
    tournament.sort(key=lambda chromosome: calculate_fitness(chromosome))
    return tournament[0], tournament[1]  # Two parents with the best fitness

# Step 5: Crossover (One-point crossover)
def crossover(parent1, parent2):
    crossover_point = random.randint(0, N_QUEENS - 1)
    child1 = parent1[:crossover_point] + parent2[crossover_point:]
    child2 = parent2[:crossover_point] + parent1[crossover_point:]
    return child1, child2

# Step 6: Mutation
def mutate(chromosome):
    for i in range(len(chromosome)):
        if random.random() < MUTATION_RATE:
            # Mutate by changing the queen's row position randomly
            chromosome[i] = random.randint(0, N_QUEENS - 1)
    return chromosome

def display_board(chromosome):
    n = len(chromosome)  # Number of queens (size of the board)

    # Create an empty board (n x n) filled with dots (.)
    board = [["x" for _ in range(n)] for _ in range(n)]

    # Place queens on the board according to the chromosome
    for col in range(n):
        row = chromosome[col]
        board[row][col] = "Q"  # Place a queen in the correct position

    # Print the board row by row
    for row in board:
        print(" | ".join(row))
        print("-" * ((n * 4)-2))
    print("\n")

# Step 7: Genetic Algorithm
def genetic_algorithm():
    # Step 1: Initialize population
    population = initialize_population()

    for generation in range(GENERATIONS):
```

```
            # Step 2: Evaluate fitness
            population.sort(key=lambda chromosome: calculate_fitness(chromosome))

            # Check if the best solution is perfect (fitness = 0)
            best_fitness = calculate_fitness(population[0])
            if best_fitness == 0:
                print(f"Solution found in generation {generation}: {population[0]}")
                return population[0]

            # Step 3: Selection, Crossover, Mutation
            new_population = []

            while len(new_population) < POPULATION_SIZE:
                # Select parents
                parent1, parent2 = select_parents(population)

                # Crossover
                child1, child2 = crossover(parent1, parent2)

                # Mutation
                child1 = mutate(child1)
                child2 = mutate(child2)

                # Add children to the new population
                new_population.extend([child1, child2])

            # Replace old population with the new population
            population = new_population[:POPULATION_SIZE]

        # If no solution is found in the given generations
        print("No solution found.")
        return None

    # Step 8: Run the Genetic Algorithm
    solution = genetic_algorithm()

    if solution:
        print("Final solution:", solution)
        print("Fitness:", calculate_fitness(solution))
        display_board(solution)
    else:
        print("No valid solution found.")
```

```
Solution found in generation 19: [4, 1, 7, 0, 3, 6, 2, 5]
Final solution: [4, 1, 7, 0, 3, 6, 2, 5]
Fitness: 0
x | x | x | Q | x | x | x | x
------------------------------
x | Q | x | x | x | x | x | x
------------------------------
x | x | x | x | x | x | Q | x
------------------------------
x | x | x | x | Q | x | x | x
------------------------------
Q | x | x | x | x | x | x | x
------------------------------
x | x | x | x | x | x | x | Q
------------------------------
x | x | x | x | x | Q | x | x
------------------------------
x | x | Q | x | x | x | x | x
------------------------------
```