```
import random

# Function to be maximized (x^2)
def fitness(x):
    return x**2

# Create a random individual: binary string of length 5 (represents numbers 0 to 31)
def create_individual():
    return [random.randint(0, 1) for _ in range(5)]

# Convert binary list to integer
def binary_to_int(binary):
    return int("".join(map(str, binary)), 2)

# Selection: tournament selection
def select(population):
    tournament_size = 3
    tournament = random.sample(population, tournament_size)
    tournament.sort(key=lambda x: x[1], reverse=True)
    return tournament[0]

# Crossover: single-point crossover
def crossover(parent1, parent2):
    crossover_point = random.randint(1, len(parent1)-1)
    child1 = parent1[:crossover_point] + parent2[crossover_point:]
    child2 = parent2[:crossover_point] + parent1[crossover_point:]
    return child1, child2

# Mutation: flip a bit with a small probability
def mutate(individual, mutation_rate=0.01):
    return [1 - bit if random.random() < mutation_rate else bit for bit in individual]

# Main Genetic Algorithm
def genetic_algorithm(population_size=10, generations=100, mutation_rate=0.01):
    # Step 1: Create initial population
    population = [(create_individual(), None) for _ in range(population_size)]

    # Step 2: Evaluate the population
    population = [(ind, fitness(binary_to_int(ind))) for ind, _ in population]

    # Main loop (evolving the population)
    for gen in range(generations):
        # Step 3: Create the next generation
        new_population = []
        while len(new_population) < population_size:
            # Select parents
            parent1 = select(population)
            parent2 = select(population)

            # Perform crossover
            child1, child2 = crossover(parent1[0], parent2[0])

            # Perform mutation
            child1 = mutate(child1, mutation_rate)
            child2 = mutate(child2, mutation_rate)

            # Add the children to the new population
            new_population.append((child1, None))
            new_population.append((child2, None))

        # Step 4: Evaluate the new population
        population = [(ind, fitness(binary_to_int(ind))) for ind, _ in new_population]

        # Print the best solution of each generation
        best_individual = max(population, key=lambda x: x[1])
        print(f"Generation {gen}: Best fitness = {best_individual[1]}, Best individual = {best_individual[0]}")

        # Optional: Stop early if solution is found (fitness == 1024, which is 31^2)
        if best_individual[1] == 1024:
            print("Optimal solution found!")
            break

    # Return the best solution found
    best_solution = max(population, key=lambda x: x[1])
    return best_solution
```

```
# Run the Genetic Algorithm
best_solution = genetic_algorithm(population_size=20, generations=50)
print(f"Best solution: {best_solution[0]} with fitness {best_solution[1]}")
```

```
Generation 0: Best fitness = 841, Best individual = [1, 1, 1, 0, 1]
Generation 1: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Generation 2: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Generation 3: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Generation 4: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Generation 5: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Generation 6: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Generation 7: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Generation 8: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Generation 9: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Generation 10: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Generation 11: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Generation 12: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Generation 13: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Generation 14: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Generation 15: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Generation 16: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Generation 17: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Generation 18: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Generation 19: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Generation 20: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Generation 21: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Generation 22: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Generation 23: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Generation 24: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Generation 25: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Generation 26: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Generation 27: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Generation 28: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Generation 29: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Generation 30: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Generation 31: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Generation 32: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Generation 33: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Generation 34: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Generation 35: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Generation 36: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Generation 37: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Generation 38: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Generation 39: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Generation 40: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Generation 41: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Generation 42: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Generation 43: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Generation 44: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Generation 45: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Generation 46: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Generation 47: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Generation 48: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Generation 49: Best fitness = 961, Best individual = [1, 1, 1, 1, 1]
Best solution: [1, 1, 1, 1, 1] with fitness 961
```