EXPERIMENT 11

AIM: WAP to implement genetic algorithm.

THEORY : A **Genetic Algorithm (GA)** is a heuristic search algorithm inspired by the principles of natural selection and genetics. It is used to find approximate solutions to optimization and search problems. GA belongs to a class of algorithms known as **evolutionary algorithms**, which simulate the process of natural evolution.

CODE :

```
import random


# Parameters
POPULATION_SIZE = 6
CHROMOSOME_LENGTH = 5  # To represent 0–31 in binary
MUTATION_RATE = 0.1
GENERATIONS = 10


# Fitness function: f(x) = x^2
def fitness(chromosome):
    x = int(chromosome, 2)
    return x ** 2


# Generate a random chromosome
def random_chromosome():
    return ''.join(random.choice(['0', '1']) for _ in range(CHROMOSOME_LENGTH))


# Selection: Tournament selection
def selection(population):
    selected = random.sample(population, 2)
    return max(selected, key=fitness)


# Crossover: Single point crossover
```

```
def crossover(parent1, parent2):
    point = random.randint(1, CHROMOSOME_LENGTH - 1)
    child1 = parent1[:point] + parent2[point:]
    child2 = parent2[:point] + parent1[point:]
    return child1, child2

# Mutation: Flip bits with some probability
def mutate(chromosome):
    return ''.join(
        bit if random.random() > MUTATION_RATE else ('1' if bit == '0' else '0')
        for bit in chromosome
    )

# Genetic Algorithm
def genetic_algorithm():
    # Step 1: Initialize population
    population = [random_chromosome() for _ in range(POPULATION_SIZE)]

    for generation in range(GENERATIONS):
        print(f"\nGeneration {generation + 1}:")
        population = sorted(population, key=fitness, reverse=True)

        # Display best in current generation
        best = population[0]
        print(f"Best: {best} -> x={int(best, 2)} fitness={fitness(best)}")

        # Step 2: Create new generation
        new_population = population[:2]  # Elitism: carry forward best 2

        while len(new_population) < POPULATION_SIZE:
```

```
            parent1 = selection(population)
            parent2 = selection(population)
            child1, child2 = crossover(parent1, parent2)
            new_population.append(mutate(child1))
            if len(new_population) < POPULATION_SIZE:
                new_population.append(mutate(child2))

        population = new_population

    # Final result
    best = max(population, key=fitness)
    print(f"\nBest solution after {GENERATIONS} generations:")
    print(f"Chromosome: {best} -> x={int(best, 2)}, fitness={fitness(best)}")

# Run the GA
genetic_algorithm()
```

```
Generation 1:
Best: 11010 -> x=26 fitness=676

Generation 2:
Best: 11010 -> x=26 fitness=676

Generation 3:
Best: 11010 -> x=26 fitness=676

Generation 4:
Best: 11100 -> x=28 fitness=784

Generation 5:
Best: 11110 -> x=30 fitness=900

Generation 6:
Best: 11110 -> x=30 fitness=900

Generation 7:
Best: 11110 -> x=30 fitness=900

Generation 8:
Best: 11110 -> x=30 fitness=900

Generation 9:
Best: 11111 -> x=31 fitness=961

Generation 10:
Best: 11111 -> x=31 fitness=961

Best solution after 10 generations:
Chromosome: 11111 -> x=31, fitness=961
```