Lab-1

Genetic Algorithm:

* Genetic Algorithm:

$$f(x) = x^2$$

Steps:

1. selecting encoding techniques → 0 to 31.
2. select initial population: – "4"

| String No | Initial population | x value | fitness $f(x) = x^2$ | Prob. $f(x)/\Sigma f(x)$ | % Prob | Expected count | Actual count |
|---|---|---|---|---|---|---|---|
| 1 | 01100 | 12 | 144 | 0.1247 | 12.47 | 0.49 | 1 |
| 2 | 11001 | 25 | 625 | 0.5411 | 54.11 | 2.164 | 2 |
| 3 | 00101 | 5 | 25 | 0.0216 | 2.16 ~~0.086~~ | 0.086 | 0 |
| 4 | 10011 | 19 | 361 | 0. 3125 | 31.25 | 1.25 | 1 |

Sum 1155

Avg 288.75

Max 625

3. select Mating pool:

| String No. | Matching pool | crossover point | Off spring after crossover | x value | fitness $f(x) = x^2$ |
|---|---|---|---|---|---|
| 1 | 01100 | 4 | 01101 | 13 | 169 |
| 2 | 11001 | | 11000 | 24 | 576 |
| 3 | 11001 | 2 | 11011 | 27 | 729 |
| 4 | 10011 | | 10001 | 17 | 289 |

Sum 1155

Avg 288.75

Max 625

4) Crossover: Random 4 & 2

Max value - 729

5) Mutation :

| String No | Offspring after crossover | Mutation chromosome for flippings | Offspring after mutation | X value | fitness $f(x)=x^2$ |
|---|---|---|---|---|---|
| 1 | 01101 | 10000 | 11101 | 29 | 841 |
| 2 | 11000 | 00000 | 11000 | 24 | 576 |
| 3 | 11011 | 00000 | 11011 | 27 | 729 |
| 4 | 10001 | 00101 | 10100 | 20 | 400 |
| Sum | | | | | 2546 |
| Avg | | | | | 630.5 |
| Max. | | | | | 841. |

~~Pseudocode~~:

Output:

Generation 0: Best fitness = 729, Best individual = 2

Generation 1: Best fitness = 729, Best individual = 27

Generation 2: Best fitness = 729, Best individual = 27

Generation 3: Best fitness = 729, Best individual = 27

⋮

Generation 18: Best fitness = 729, Best individual = 27

Generation 19: Best fitness = 729, Best individual = 27

Best solution : $x = 27$, $f(x) = 729$

29/8/25

Output:

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

```
Generation 0: Best fitness = 729, Best individual = 27
Generation 1: Best fitness = 729, Best individual = 27
Generation 2: Best fitness = 729, Best individual = 27
Generation 3: Best fitness = 729, Best individual = 27
Generation 4: Best fitness = 729, Best individual = 27
Generation 5: Best fitness = 729, Best individual = 27
Generation 6: Best fitness = 729, Best individual = 27
Generation 7: Best fitness = 729, Best individual = 27
Generation 8: Best fitness = 729, Best individual = 27
Generation 9: Best fitness = 729, Best individual = 27
Generation 10: Best fitness = 729, Best individual = 27
Generation 11: Best fitness = 729, Best individual = 27
Generation 12: Best fitness = 729, Best individual = 27
Generation 13: Best fitness = 729, Best individual = 27
Generation 14: Best fitness = 729, Best individual = 27
Generation 15: Best fitness = 729, Best individual = 27
Generation 16: Best fitness = 729, Best individual = 27
Generation 17: Best fitness = 729, Best individual = 27
Generation 18: Best fitness = 729, Best individual = 27
Generation 19: Best fitness = 729, Best individual = 27
Best solution: x = 27, f(x) = 729
```

```
import random
```

Code:

```
import random

# Problem parameters
CHROMOSOME_LENGTH = 5  # 5 bits to represent numbers 0-31
POPULATION_SIZE = 10
GENERATIONS = 20
CROSSOVER_RATE = 0.7
MUTATION_RATE = 0.01

# Decode binary chromosome to integer
def decode(chromosome):
    return int("".join(str(bit) for bit in chromosome), 2)

# Fitness function: maximize x^2
def fitness(chromosome):
    x = decode(chromosome)
    return x ** 2

# Generate initial population
def init_population():
    population = []
    for _ in range(POPULATION_SIZE):
        chromosome = [random.randint(0,1) for _ in range(CHROMOSOME_LENGTH)]
```

```
        population.append(chromosome)
    return population

# Roulette Wheel Selection
def select(population, fitnesses):
    total_fitness = sum(fitnesses)
    pick = random.uniform(0, total_fitness)
    current = 0
    for i, fit in enumerate(fitnesses):
        current += fit
        if current > pick:
            return population[i]

# Single-point Crossover
def crossover(parent1, parent2):
    if random.random() < CROSSOVER_RATE:
        point = random.randint(1, CHROMOSOME_LENGTH-1)
        child1 = parent1[:point] + parent2[point:]
        child2 = parent2[:point] + parent1[point:]
        return child1, child2
    else:
        return parent1[:], parent2[:]

# Mutation: bit flip
def mutate(chromosome):
    for i in range(CHROMOSOME_LENGTH):
        if random.random() < MUTATION_RATE:
            chromosome[i] = 1 - chromosome[i]
    return chromosome

# Main GA loop
def genetic_algorithm():
    population = init_population()

    for generation in range(GENERATIONS):
        fitnesses = [fitness(chromo) for chromo in population]

        print(f"Generation {generation}: Best fitness = {max(fitnesses)}, Best individual = 
{decode(population[fitnesses.index(max(fitnesses))])}")

        new_population = []
```

```
        while len(new_population) < POPULATION_SIZE:
            parent1 = select(population, fitnesses)
            parent2 = select(population, fitnesses)
            child1, child2 = crossover(parent1, parent2)
            child1 = mutate(child1)
            child2 = mutate(child2)
            new_population.extend([child1, child2])

        population = new_population[:POPULATION_SIZE]

    # Final result
    fitnesses = [fitness(chromo) for chromo in population]
    best_index = fitnesses.index(max(fitnesses))
    best_chromosome = population[best_index]
    best_value = decode(best_chromosome)
    print(f"Best solution: x = {best_value}, f(x) = {fitness(best_chromosome)}")

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