**LAB – 2**

**GENES ALGORITHM**

**(APPLICATION USED OPTIMIZING THE FUNCTION)**

**CODE :**

import random

CHROMOSOME\_LENGTH = 5

POPULATION\_SIZE = 6

MAX\_GEN = 20

MUTATION\_RATE = 0.1

def decode(chromosome):

return int("".join(str(bit) for bit in chromosome), 2)

def fitness(chromosome):

x = decode(chromosome)

return x \*\* 2

def random\_chromosome():

return [random.randint(0, 1) for \_ in range(CHROMOSOME\_LENGTH)]

def selection(population):

fitness\_values = [fitness(ind) for ind in population]

total\_fit = sum(fitness\_values)

pick = random.uniform(0, total\_fit)

current = 0

for ind, fit in zip(population, fitness\_values):

current += fit

if current > pick:

return ind

return population[-1] # This line should be unreachable if the logic is correct

def crossover(p1, p2):

point = random.randint(1, CHROMOSOME\_LENGTH - 1)

child1 = p1[:point] + p2[point:]

child2 = p2[:point] + p1[point:]

return child1, child2

def mutate(chromosome):

for i in range(CHROMOSOME\_LENGTH):

if random.random() < MUTATION\_RATE:

chromosome[i] = 1 - chromosome[i]

return chromosome

def gene\_algorithm():

population = [random\_chromosome() for \_ in range(POPULATION\_SIZE)]

for gen in range(MAX\_GEN):

new\_population = []

while len(new\_population) < POPULATION\_SIZE:

parent1 = selection(population)

parent2 = selection(population)

child1, child2 = crossover(parent1, parent2)

new\_population.append(mutate(child1))

new\_population.append(mutate(child2))

population = new\_population

best = max(population, key=fitness)

best\_x = decode(best)

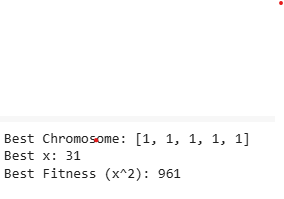
print("Best Chromosome:", best)

print("Best x:", best\_x)

print("Best Fitness (x^2):", fitness(best))

gene\_algorithm()

**OUTPUT :**

****