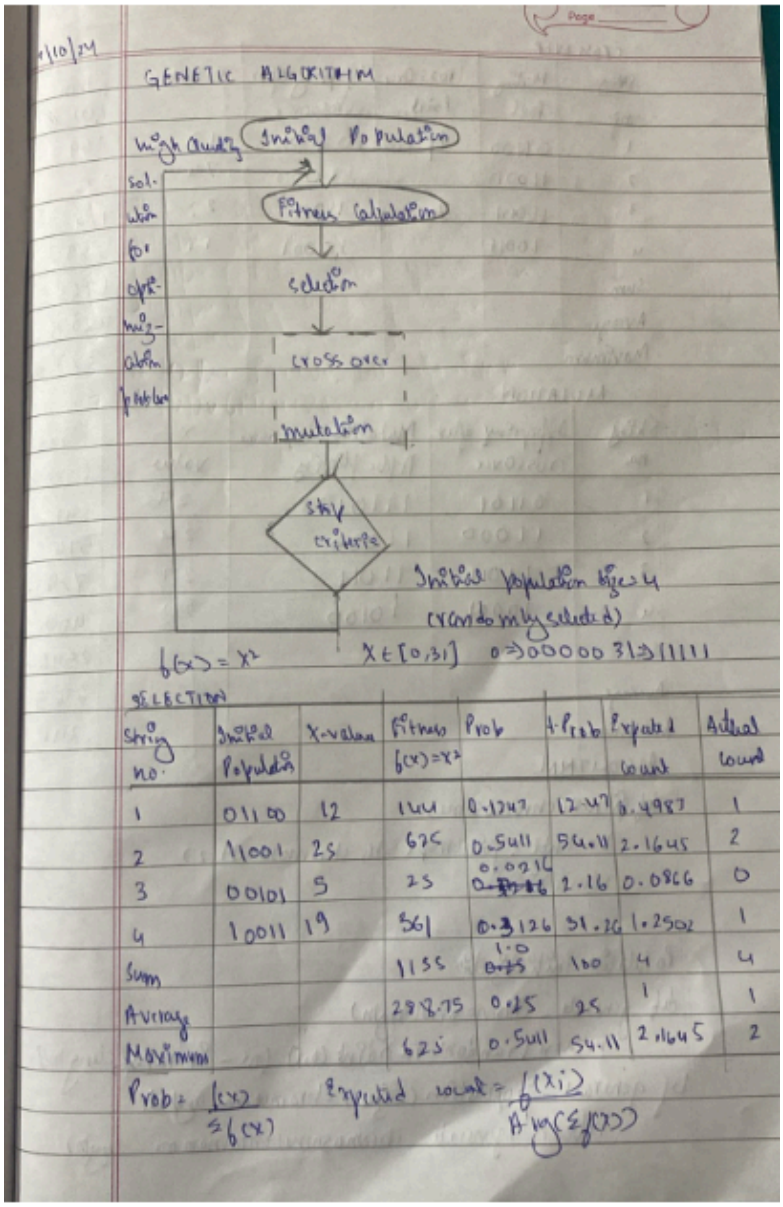
Program 1

Problem statement:Genetic Algorithm for Optimization Problems

Algorithm:



GENETIC ALGORITHM

Initial Population

Fitness calculation

Selection

Cross over

Mutation

Stop criteria

Initial population size = 4 (randomly selected)

$f(x) = x^2$ $x \in [0, 31]$ $0 \Rightarrow 00000$ $31 \Rightarrow 11111$

SELECTION

String no.	Initial Population	X-value	Fitness $f(x)=x^2$	Prob	f. Prob	Expected count	Actual count
1	01100	12	144	0.1247	12.47	0.4987	1
2	11001	25	625	0.5411	54.11	2.1645	2
3	00101	5	25	0.0216	2.16	0.0866	0
4	10011	19	361	0.3126	31.26	1.2502	1
Sum			1155	1.0	100	4	4
Average			288.75	0.25	25	1	1
Maximum			625	0.5411	54.11	2.1645	2

Prob = $\frac{f(x)}{\sum f(x)}$ Expected count = $\frac{f(x_i)}{\text{Avg}(\sum f(x))}$

CROSSOVER

String no.	Mating Pool	Cross Over Point	Offspring 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					1763
Average					440.5
Maximum					729

MUTATION

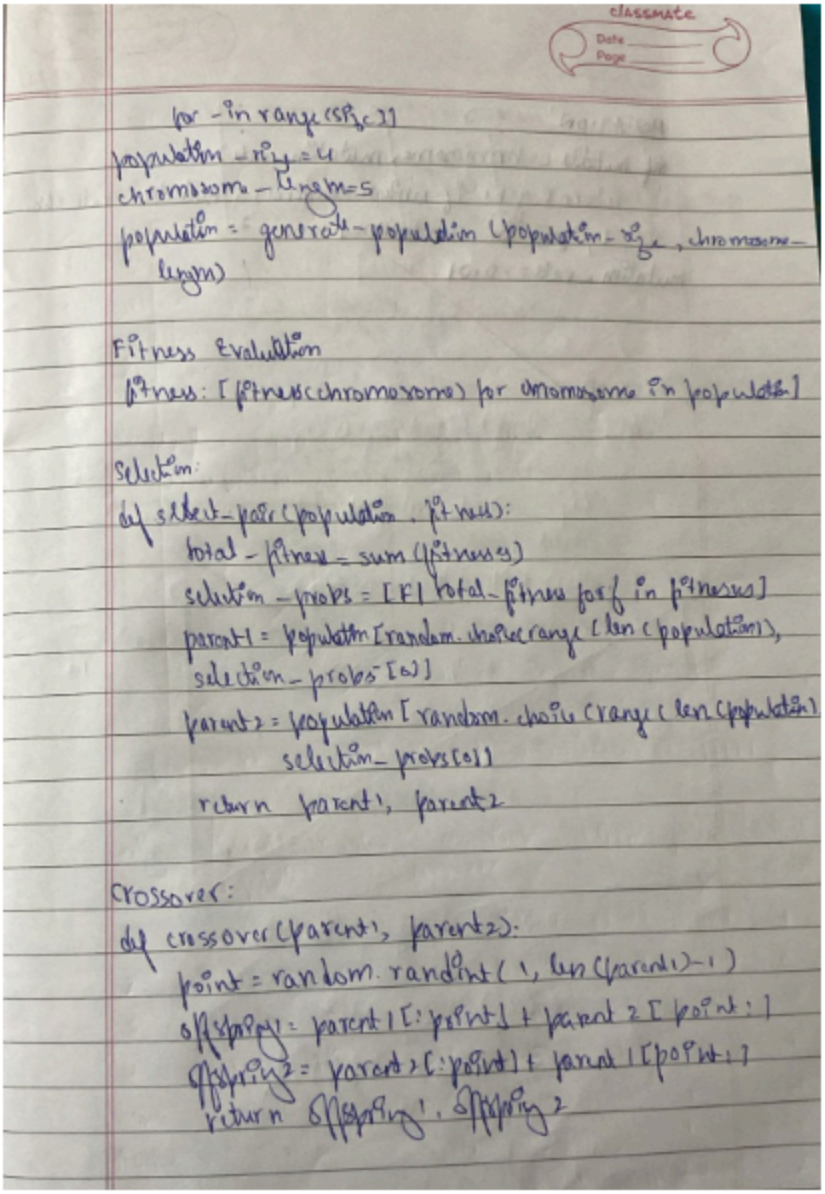
String no.	Offspring after crossover	Mutation Chromosome After flipping	x value	Fitness $f(x)=x^2$
1	01101	11101	29	841
2	11000	11000	24	576
3	11011	11011	27	729
4	10001	10100	20	400
Sum				2546
Average				636.5
Maximum				841

ALGORITHM:

```
def fitness(chromosome):
    x = int(''.join(map(str, chromosome)), 2)
    return x**2
```

Population initialization

```
def generate_chromosome(length):
    return [random.randint(0,1) for _ in range(length)]
def generate_population(size, chromosome_length):
    return [generate_chromosome(chromosome_length)
```

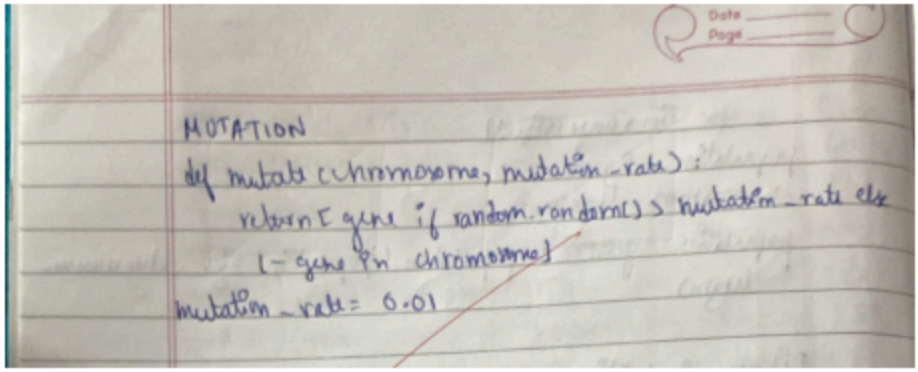


for _ in range(50):
population_size = 4
chromosome_length = 5
population = generate_population(population_size, chromosome_length)

Fitness Evaluation
fitness: [fitness(chromosome) for chromosome in population]

Selection:
def select_pair(population, fitness):
    total_fitness = sum(fitnesses)
    selection_probs = [f/ total_fitness for f in fitnesses]
    parent1 = population[random.choice(range(len(population)), selection_probs)]
    parent2 = population[random.choice(range(len(population)) selection_probs)]
    return parent1, parent2

Crossover:
def crossover(parent1, parent2):
    point = random.randint(1, len(parent1)-1)
    offspring1 = parent1[:point] + parent2[point:]
    offspring2 = parent2[:point] + parent1[point:]
    return offspring1, offspring2



MUTATION
def mutate(chromosome, mutation_rate):
    return [gene if random.random() > mutation_rate else
        1-gene for gene in chromosome]
mutation_rate = 0.01

Code:
import random
def fitness(x):

```
    return x**2
def create_population(pop_size):
    population = []
    for _ in range(pop_size):
        individual = random.randint(0, 31)
        population.append(individual)
    return population
def select_parents(population):
    population.sort(key=lambda x: fitness(x), reverse=True)
    return population[0], population[1]
def crossover(parent1, parent2):
    crossover_point = random.randint(0, 4)
    mask1 = parent1 >> crossover_point
    mask2 = parent2 & ((1 << crossover_point) - 1)
    child = (mask1 << crossover_point) | mask2
    return child
def mutate(individual, mutation_rate=0.3):
    if random.random() < mutation_rate:
        bit_to_flip = random.randint(0, 4)
        individual ^= (1 << bit_to_flip)
    return individual
def genetic_algorithm(pop_size, generations, mutation_rate):
    population = create_population(pop_size)
    for generation in range(generations):
        parent1, parent2 = select_parents(population)
        new_population = [parent1]

        for _ in range((pop_size - 1) // 2):
            child1 = crossover(parent1, parent2)
            child2 = crossover(parent2, parent1)
            child1 = mutate(child1, mutation_rate)
            child2 = mutate(child2, mutation_rate)
            new_population.append(child1)
            new_population.append(child2)
        population = new_population
        best_individual = max(population, key=lambda x: fitness(x))
        print(f"Generation {generation + 1}: Best individual = {best_individual}, Fitness =
{fitness(best_individual)}")
    return best_individual
pop_size = 6
generations = 20
mutation_rate = 0.6
best = genetic_algorithm(pop_size, generations, mutation_rate)
print(f"Best solution found: x = {best}, f(x) = {fitness(best)}")
```

```python
import random
def fitness(x):
    return x**2
def create_population(pop_size):
    population = []
    for _ in range(pop_size):
        individual = random.randint(0, 31)
        population.append(individual)
    return population
def select_parents(population):
    population.sort(key=lambda x: fitness(x), reverse=True)
    return population[0], population[1]
def crossover(parent1, parent2):
    crossover_point = random.randint(0, 4)
    mask1 = parent1 >> crossover_point
    mask2 = parent2 & ((1 << crossover_point) - 1)
    child = (mask1 << crossover_point) | mask2
    return child
def mutate(individual, mutation_rate=0.3):
    if random.random() < mutation_rate:
        bit_to_flip = random.randint(0, 4)
        individual ^= (1 << bit_to_flip)
    return individual
def genetic_algorithm(pop_size, generations, mutation_rate):
    population = create_population(pop_size)
    for generation in range(generations):
        parent1, parent2 = select_parents(population)
        new_population = [parent1]

        for _ in range((pop_size - 1) // 2):
            child1 = crossover(parent1, parent2)
            child2 = crossover(parent2, parent1)
            child1 = mutate(child1, mutation_rate)
            child2 = mutate(child2, mutation_rate)
            new_population.append(child1)
            new_population.append(child2)
        population = new_population
        best_individual = max(population, key=lambda x: fitness(x))
        print(f"Generation {generation + 1}: Best individual = {best_individual}, Fitness = {fitness(best_individual)}")
    return best_individual
pop_size = 6
generations = 20
mutation_rate = 0.6
best = genetic_algorithm(pop_size, generations, mutation_rate)
print(f"Best solution found: x = {best}, f(x) = {fitness(best)}")
```

Output:

```
Generation 1: Best individual = 28, Fitness = 784
Generation 2: Best individual = 30, Fitness = 900
Generation 3: Best individual = 31, Fitness = 961
Generation 4: Best individual = 31, Fitness = 961
Generation 5: Best individual = 31, Fitness = 961
Generation 6: Best individual = 31, Fitness = 961
Generation 7: Best individual = 31, Fitness = 961
Generation 8: Best individual = 31, Fitness = 961
Generation 9: Best individual = 31, Fitness = 961
Generation 10: Best individual = 31, Fitness = 961
Generation 11: Best individual = 31, Fitness = 961
Generation 12: Best individual = 31, Fitness = 961
Generation 13: Best individual = 31, Fitness = 961
Generation 14: Best individual = 31, Fitness = 961
Generation 15: Best individual = 31, Fitness = 961
Generation 16: Best individual = 31, Fitness = 961
Generation 17: Best individual = 31, Fitness = 961
Generation 18: Best individual = 31, Fitness = 961
Generation 19: Best individual = 31, Fitness = 961
Generation 20: Best individual = 31, Fitness = 961
Best solution found: x = 31, f(x) = 961
```