Program 1
Problem statement:Genetic Algorithm for Optimization Problems
Algorithm:

					6	Pege_)	
110/24		***			1	VA MAY			
44	GENE	10 A16	KITHM	Land Corp.	and H	100	47		
11 (0)		de Jui	S-1 W- W	mistalian)	150%		in the second	323	
1100		The same	2		100 17 0	10000	F		
-	162	(50)	reus calo	100	10011	No. of Contract of	. 6	100 / A	
		(10	THEIS CON	mig), VII	1,60.5	1000			
-45	61		diden	CALLS O	6 9/3	1000	M	1000	
	OVE-	,	CITOLIN	10000	1		AND THE RESERVE TO THE PERSON OF THE PERSON	1000	
100,000	M2-								
	Olm 105 over								
187	for the bear	7 1	water	100	200	4	100	46	
1000	wex	170	mount	100	MANAGE	1	48	880	
4834	1 100						-		
102	200	1	VAS		30037		2700		
ALC:		-	criture						
ALPRI	1 1		Y		as popul				
038		0.55359	9010		ndo my				
1163313	600	= Xx	, X	£ [0,31]	0=)00	3000	111111111111111111111111111111111111111	1	
27	9ELECTIT			.0.		0 0		. 1.	
108	String		X-value	Fitness		1-Prob 8		Ade	
	no.	Population		P(x)=x>		1MEAG	ORGER SECTION	100	
	1	0110	12		0-1242	12-11	1884.6	1	
	2 6	11001	25	625	0.021			2	
7300	3	00101	5	25	0-17-16	2.16	0.0866	0	
1,398	4	10011	19	561	0.3126	51.20	1.2502	1	
	Sugar		139	1155	0.45	100 6	100000		
		THE REAL PROPERTY.		298.15	AND ASSESSED.		1	1	
	Average				- 11	_	2,1645	100	
(pull	Movimum	-10) (10 410	625	_	_	_	100	
1999	Prob =	CXD	1 5 xhey	ناط المد	me = 60	CC 1022	3		
	The same of	= (x)	cine and a	11 111	HV	(5/0)	"		

				C. inte		
CE055 3	188		slehen ator	x Volum	Ctrox	
Short	Mating		er offspring after		P. LUN = M	
ano.	8008	loint		13	169	-
1	01100	4	11000	24	576	-
2	11001		11011	27	729	1
3	11001	2		17	289	-
ч	LODII		10001	B 2 8 1		1
Sum	1000	9	4.5.012	100000	1763	-
Avery	1770				440.3	-
Marina	2	130.00	(NO (F 97)	1	729	10
	MOITATI			1	46 -	
Sking	O Mayron	y star	Mutatim Chrimo	Som X	16the	0
NO.	CLOSEDI		Mer Happey	value	(0):	Ly 3
1	0110			29	841	
	110		11000	24	576	
2	-			27	72	9
3	3011		11011	20		
4 66	10	001	10100		40	
Sum	0000	0 110	0738	00000	25	
	14186	755		100 15 55	63.	6.5
Average		1.9		1000	8	41
	17/	SYS	(0)	Selected 1	Total I	
ALGON	TMM:		-			
	W (CNYS			02/1/0		
X	= PW(: Porna	op (SH, chrow	(2, [[unolou	- 51	
	duin xt	434	25 1 23	1010.0	83	
-			133 100 100			
	16 20		1		313	
Po lulati	w into	disals	1511		1000	
del aun	wate .	bromo So	millygh):	Transfer of	MINNE	
0	notes to t	van den	1. randint co,	11 (ns - in	range cles	19 m)
- 10	MINIM I	, taneor	12 114 114 (0)	13 40	- lula	0
by gen	11097 - 1	nopulati	m (size, chi	to most the - w	3	
10	return	Tannera	h - chromoson	EMPORTULIN	one - lyn	10.7

	Poge
9	for - in range (spc.)
100	problem = 12y = 4
ck	10 Missoria - whether
be	pulitin = goneral - population (population - 25, chromosor
	(Lynn)
F?	thus Evaluation
16	thus: [formuse chromoxomo) for anomosomo en populati
su	utem:
de	silet-pos (population, pt mus):
	botal - Khrey - sum (fitners)
	solution - props = [FI total fitness for fin perness)
7 1	paront = population Francism wherecrange (len chapulation)
81	saledian-propostoll
	parent = propulation I random chose crange clan chapter
	schitim-provision
	reburn parent, parents
	the state of the s
CYt	ossover:
di	enssover (parent), parentes.
19	Lout - random. randint (1, un yarener)
1	Mil On mount I I well + locked 2 [point]
	eltabula = house school + loury 1 choling 1
	return Stapping, Stapping,
	Love a ottobring . Her. a

1	Pogs Date
	HOTATION
	de mutate consumosome, mutation rate): return t give if random rondernes s mutation rate else
marrant	1- gine in chromosomes
	metation rate = 0.01
-money and	1- gens in chromomos

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 \stackrel{\wedge}{=} (1 \ll 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)\}")
```

```
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)</pre>
    child = (mask1 << crossover_point) | mask2</pre>
    return child
def mutate(individual, mutation_rate=0.3):
    if random.random() < mutation_rate:</pre>
        bit_to_flip = random.randint(0, 4)
        individual ^= (1 << bit_to_flip)</pre>
    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:

```
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
```