```python
        population = new_population

        best_solution = max(population, key=fitness)
        print(f"Generation {generation + 1}: Best solution = {best_solution}, Fitness = {fitness(best_solution)}")

    return max(population, key=fitness)


pop_size = 5
generations = 4
mutation_rate = 0.01
lower_bound = 0
upper_bound = 31

best_solution = genetic_algorithm(pop_size, generations, mutation_rate, lower_bound, upper_bound)
print(f"\nBest solution found: {best_solution}, Fitness = {fitness(best_solution)}")
```

```
        for _ in range(pop_size // 2):
            parent1 = selection(population)
            parent2 = selection(population)

            child1, child2 = crossover(parent1, parent2)
            child1 = mutation(child1, mutation_rate, lower_bound, upper_bound)
            child2 = mutation(child2, mutation_rate, lower_bound, upper_bound)

            new_population.extend([child1, child2])

        population = new_population

        best_solution = max(population, key=fitness)
        print(f"Generation {generation + 1}: Best solution = {best_solution}, Fitness = {fitness(best_solution)}")

    return max(population, key=fitness)


pop_size = 5
generations = 4
mutation_rate = 0.01
```

```
    return child1, child2


def mutation(child, mutation_rate, lower_bound, upper_bound):
    if random.random() < mutation_rate:
        binary_child = to_binary_string(child)
        # Avoid mutating the sign bit
        mutation_point = random.randint(1, len(binary_child) - 1) if binary_child.startswith('-') else random.randint(0, len(
        mutated_child_list = list(binary_child)
        mutated_child_list[mutation_point] = '1' if mutated_child_list[mutation_point] == '0' else '0'
        mutated_child = ''.join(mutated_child_list)
        child = from_binary_string(mutated_child)

    return max(lower_bound, min(child, upper_bound))


def genetic_algorithm(pop_size, generations, mutation_rate, lower_bound, upper_bound):
    population = create_population(pop_size, lower_bound, upper_bound)

    for generation in range(generations):
        new_population = []
```

```python
def from_binary_string(binary_string):
    """Converts a binary string representation back to an integer, handling negative numbers."""
    if binary_string.startswith('-'):
        return -int(binary_string[1:], 2)
    else:
        return int(binary_string, 2)

def crossover(parent1, parent2):
    binary_parent1 = to_binary_string(parent1)
    binary_parent2 = to_binary_string(parent2)

    # Ensure crossover point is at least 1 and not beyond the length of the binary string
    crossover_point = random.randint(1, max(1, len(binary_parent1.lstrip('-')) - 1))


    child1_binary = binary_parent1[:crossover_point] + binary_parent2[crossover_point:]
    child2_binary = binary_parent2[:crossover_point] + binary_parent1[crossover_point:]

    child1 = from_binary_string(child1_binary)
    child2 = from_binary_string(child2_binary)
```

```python
import random

def fitness(x):
    return x**2

def create_population(pop_size, lower_bound, upper_bound):
    population = [random.randint(lower_bound, upper_bound) for _ in range(pop_size)]
    return population

def selection(population):
    tournament_size = 3
    selected = random.sample(population, tournament_size)
    selected = sorted(selected, key=fitness, reverse=True)
    return selected[0]

def to_binary_string(number, bits=32):
    """Converts an integer to its binary string representation, handling negative numbers."""
    if number < 0:
        return '-' + bin(abs(number))[2:].zfill(bits)
    else:
        return bin(number)[2:].zfill(bits)

def from_binary_string(binary_string):
```

```
print(f"\nBest solution found: {best_solution}, Fitness = {fi
```

```
Generation 1: Best solution = 29, Fitness = 841
Generation 2: Best solution = 29, Fitness = 841
Generation 3: Best solution = 29, Fitness = 841
Generation 4: Best solution = 29, Fitness = 841

Best solution found: 29, Fitness = 841
```