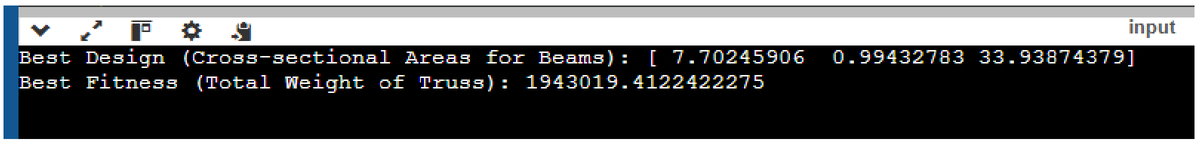
```
import numpy as np
import random

# Define the fitness function (objective function)
# This is a simplified objective function that calculates the weight of a truss
# For this example, we assume linear relationships and basic mechanical
constraints
def fitness_function(design):
    # Example parameters for a truss with 3 beams (for simplicity)
    beam_lengths = [5.0, 7.0, 6.0]  # lengths of beams (in meters)
    beam_materials = [7800, 7800, 7800]  # density of steel in kg/m^3
    # Max stress for steel (simplified)
    max_stress = 250e6

    # Design is the cross-sectional areas of the truss beams
    # Calculate the weight (mass) of the truss
    total_weight = sum(beam_lengths[i] * beam_materials[i] * design[i] for i in
range(len(design)))

    # Apply a basic constraint: if any member's cross-sectional area results in
stress exceeding max_stress, penalize
    for i in range(len(design)):
        stress = design[i] * beam_materials[i] * beam_lengths[i]  # Simplified
stress calculation
        if stress > max_stress:
            total_weight += 100000  # Apply penalty to the fitness function

    return total_weight

# Cuckoo Search Algorithm
def cuckoo_search(nests, max_iter, pa, alpha=0.01):
    # Initialize nests with random solutions
    best_solution = None
    best_fitness = float('inf')

    for iteration in range(max_iter):
        # Evaluate fitness of the current nests
        fitness = [fitness_function(nest) for nest in nests]

        # Find the best nest
        min_fitness = min(fitness)
        min_index = fitness.index(min_fitness)

        if min_fitness < best_fitness:
            best_fitness = min_fitness
            best_solution = nests[min_index]

        # Generate new solutions using Lévy flights
        new_nests = []
        for nest in nests:
            new_nest = []
            for i in range(len(nest)):
                # Lévy flight - random walk
                new_value = nest[i] + alpha * random.uniform(-1, 1) * (max(nest)
- min(nest))
                new_nest.append(new_value)

            # Apply boundary constraints (e.g., cross-sectional area should be
positive)
            new_nest = np.clip(new_nest, 0.1, 100.0)  # Assume area range [0.1,
100] cm^2

            # Apply discovery probability to abandon some nests
            if random.random() < pa:
                # Generate a random new solution
                new_nests.append([random.uniform(0.1, 100.0) for _ in
range(len(nest))])
            else:
                new_nests.append(new_nest)

        # Replace the worst nests with the best ones
        nests = sorted(zip(nests, fitness), key=lambda x: x[1])
        nests[:len(nests)//2] = zip(new_nests[:len(nests)//2],
fitness[:len(nests)//2])

    return best_solution, best_fitness

# Parameters
num_nests = 10  # Number of nests (candidate solutions)
max_iter = 100  # Maximum number of iterations
pa = 0.25  # Probability of discovering a new nest

# Initialize nests with random values for cross-sectional areas (between 0.1 and
100 cm^2)
initial_nests = [np.random.uniform(0.1, 100.0, 3) for _ in range(num_nests)]  #
3 design variables for simplicity

# Run the Cuckoo Search algorithm
best_design, best_weight = cuckoo_search(initial_nests, max_iter, pa)

print("Best Design (Cross-sectional Areas for Beams):", best_design)
print("Best Fitness (Total Weight of Truss):", best_weight)
```



```
Best Design (Cross-sectional Areas for Beams): [ 7.70245906  0.99432783 33.93874379]
Best Fitness (Total Weight of Truss): 1943019.4122422275
```