```
import random
import numpy as np

# Function to initialize the population of antibodies (solutions)
def initialize_population(population_size, solution_size):
    # Randomly initialize antibodies (solutions)
    population = []
    for _ in range(population_size):
        antibody = np.random.uniform(low=-10, high=10, size=solution_size)  # Example:
solutions in range [-10, 10]
        population.append(antibody)
    return population

# Function to calculate the affinity of an antibody based on a fitness function
def calculate_affinity(population):
    affinities = []
    for antibody in population:
        # Example fitness function: sum of squares of the solution values (to
minimize)
        fitness = np.sum(antibody ** 2)
        affinity = 1 / (1 + fitness)  # Higher fitness -> higher affinity
        affinities.append(affinity)
    return affinities

# Function to select antibodies for cloning based on affinity
def select_antibodies_for_cloning(population, affinities):
    # Normalize affinities
    total_affinity = sum(affinities)
    probabilities = [affinity / total_affinity for affinity in affinities]

    # Fix: Instead of using np.random.choice on the population directly,
    # select indices and then use those to index the population
    selected_indices = np.random.choice(
        range(len(population)),
        size=len(population),
        p=probabilities,
        replace=True
    )

    selected_antibodies = [population[i] for i in selected_indices]
    return selected_antibodies

# Function to clone selected antibodies (just copy them)
def clone_antibodies(selected_antibodies):
    return selected_antibodies.copy()

# Function to mutate antibodies (introduce diversity)
def mutate_antibodies(cloned_antibodies, mutation_rate):
    mutated_antibodies = []
    for antibody in cloned_antibodies:
        # Mutate with a random small value if mutation rate is met
        if random.random() < mutation_rate:
            mutation = np.random.uniform(low=-1, high=1, size=antibody.shape)
            antibody = antibody + mutation  # Add mutation to the antibody
        mutated_antibodies.append(antibody)
    return mutated_antibodies

# Function to select the next generation of antibodies based on affinity
```

```
def select_next_generation(population, mutated_antibodies, affinities):
    # Combine original and mutated antibodies
    combined_population = population + mutated_antibodies
    # Calculate new affinities for the combined population
    combined_affinities = calculate_affinity(combined_population)

    # Sort antibodies by affinity (select top half)
    sorted_indices = np.argsort(combined_affinities)[::-1]  # Sort descending by
affinity
    next_generation = [combined_population[i] for i in
sorted_indices[:len(population)]]
    return next_generation

# Function to get the best antibody in the population
def best_antibody(population, affinities):
    best_index = np.argmax(affinities)
    return population[best_index]

# Main function to execute the Clonal Selection Algorithm
def clonal_selection_algorithm(population_size, solution_size, mutation_rate,
num_iterations):
    # Initialize population of antibodies (solutions)
    antibodies = initialize_population(population_size, solution_size)

    for iteration in range(num_iterations):
        # Calculate affinity of antibodies
        affinities = calculate_affinity(antibodies)

        # Select antibodies for cloning
        selected_antibodies = select_antibodies_for_cloning(antibodies, affinities)

        # Clone selected antibodies
        cloned_antibodies = clone_antibodies(selected_antibodies)

        # Mutate cloned antibodies
        mutated_antibodies = mutate_antibodies(cloned_antibodies, mutation_rate)

        # Select antibodies for the next generation
        antibodies = select_next_generation(antibodies, mutated_antibodies,
affinities)

        # Optional: Print progress for longer runs
        if (iteration + 1) % 10 == 0:
            best = best_antibody(antibodies, calculate_affinity(antibodies))
            print(f"Iteration {iteration + 1}/{num_iterations}, Best fitness:
{np.sum(best ** 2):.6f}")

    # Return best antibody (solution)
    best = best_antibody(antibodies, calculate_affinity(antibodies))
    return best

# Example usage
population_size = 50  # Number of antibodies in the population
solution_size = 5  # Size of each solution (e.g., number of parameters)
mutation_rate = 0.1  # Probability of mutation
num_iterations = 100  # Number of iterations (generations)

# Run Clonal Selection Algorithm
```

```python
best_solution = clonal_selection_algorithm(population_size, solution_size,
mutation_rate, num_iterations)
print("Best solution:", best_solution)
print("Fitness value:", np.sum(best_solution ** 2))
```

```
Iteration 10/100, Best fitness: 23.855810
Iteration 20/100, Best fitness: 5.160971
Iteration 30/100, Best fitness: 0.712264
Iteration 40/100, Best fitness: 0.322100
Iteration 50/100, Best fitness: 0.265101
Iteration 60/100, Best fitness: 0.265101
Iteration 70/100, Best fitness: 0.265101
Iteration 80/100, Best fitness: 0.265101
Iteration 90/100, Best fitness: 0.225927
Iteration 100/100, Best fitness: 0.225927
Best solution: [-0.22832568 -0.05301184 -0.05875036 -0.07173226  0.40297251]
Fitness value: 0.22592683526565888
```