```python
import random
import math

# Function to calculate the number of conflicts (energy function)
def calculate_conflicts(board):
    n = len(board)
    conflicts = 0
    # Count conflicts between queens
    for i in range(n):
        for j in range(i + 1, n):
            if board[i] == board[j] or abs(board[i] - board[j]) == abs(i - j):
                conflicts += 1
    return conflicts

# Function to generate a random initial state
def random_board(n):
    return [random.randint(0, n-1) for _ in range(n)]

# Function to generate a neighboring state
def get_neighbor(board):
    neighbor = board[:]
    i = random.randint(0, len(board) - 1)
    neighbor[i] = random.randint(0, len(board) - 1)
    return neighbor

# Simulated Annealing algorithm
def simulated_annealing(n, initial_temp=1000, cooling_rate=0.95, max_iter=10000):
    current_board = random_board(n)
    current_energy = calculate_conflicts(current_board)
    temp = initial_temp
```

```
    for _ in range(max_iter):
        if current_energy == 0:
            return current_board  # Solution found

        # Generate a neighboring state
        neighbor = get_neighbor(current_board)
        neighbor_energy = calculate_conflicts(neighbor)

        # Calculate the energy difference
        energy_diff = current_energy - neighbor_energy

        # Accept the neighbor with probability based on temperature
        if energy_diff > 0 or random.random() < math.exp(energy_diff / temp):
            current_board = neighbor
            current_energy = neighbor_energy

        # Decrease temperature
        temp *= cooling_rate

    return current_board  # Return the best solution found

# Example usage
n = 4  # N-Queens problem size
solution = simulated_annealing(n)
print("Solution:", solution)
print("Conflicts:", calculate_conflicts(solution))
```

```
Solution: [1, 3, 0, 2]
Conflicts: 0

=== Code Execution Successful ===
```