```python
import random

N = 8  # Size of the chessboard (8x8)

# Function to calculate the number of conflicts (attacking pairs of queens)
def calculate_conflicts(state):
    conflicts = 0
    for i in range(N):
        for j in range(i + 1, N):
            if state[i] == state[j] or abs(state[i] - state[j]) == abs(i - j):
                conflicts += 1
    return conflicts

# Function to generate a random state (a possible solution)
def generate_initial_state():
    state = list(range(N))  # Generate a list of column positions
    random.shuffle(state)    # Shuffle to get a random placement
    return state

# Function to generate a neighbor state by randomly moving one queen
def generate_neighbor(state):
    neighbor = state[:]
    i = random.randint(0, N - 1)  # Randomly choose a queen
    j = random.randint(0, N - 1)  # Randomly choose a new row for this queen
    neighbor[i] = j
    return neighbor
```

```python
def simulated_annealing():
    # Initial state
    current_state = generate_initial_state()
    current_conflicts = calculate_conflicts(current_state)
    temperature = 10000  # Initial temperature
    cooling_rate = 0.995  # Cooling rate
    min_temperature = 1  # Minimum temperature

    # Repeat the process until a solution is found or temperature drops
    while temperature > min_temperature:
        # Generate a neighbor state
        neighbor_state = generate_neighbor(current_state)
        neighbor_conflicts = calculate_conflicts(neighbor_state)

        # Calculate the change in conflicts
        delta = current_conflicts - neighbor_conflicts

        # Always accept the neighbor to increase attacks
        if delta >= 0:
            current_state = neighbor_state
            current_conflicts = neighbor_conflicts

        # Cool the system down
        temperature *= cooling_rate

        # Check if a valid solution has been found
        if current_conflicts == 0:
            break  # No conflicts means a valid solution is found

    return current_state
```

```python
# Function to print the solution in 1's and 0's
def print_solution(state):
    board = [[0 for _ in range(N)] for _ in range(N)]
    for row in range(N):
        board[state[row]][row] = 1
    for row in board:
        print(" ".join(map(str, row)))

# Driver function
if __name__ == "__main__":
    print("Name : Vyom Gupta")
    print("USN  : 1BM22CS333\n")

    # Run Simulated Annealing to solve the N-Queens problem
    solution = simulated_annealing()

    # Print the solution once it's found (no conflicts)
    print("Solution (Queens do not attack each other):")
    print_solution(solution)
```

```
Name : Vyom Gupta
USN  : 1BM22CS333

Solution (Queens do not attack each other):
0 0 0 0 1 0 0 0
0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 1 0 0 0 0
0 0 0 0 0 0 1 0
1 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0
0 1 0 0 0 0 0 0
```