Lab 6: Implement Hill ClimbingSearch Algorithm to Solve N - Queens Problem

Code

```
import random

def calculate_cost(state):
    """Calculate the number of conflicts in the current state."""
    cost = 0
    n = len(state)
    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):
                cost += 1
    return cost

def get_neighbors(state):
    """Generate all possible neighbors by moving each queen in its
column."""
    neighbors = []
    n = len(state)
    for col in range(n):
        for row in range(n):
            if state[col] != row:  # Move the queen in column `col` to a
different row
                new_state = list(state)
                new_state[col] = row
                neighbors.append(new_state)
    return neighbors

def hill_climbing(n, max_iterations=1000):
    """Perform hill climbing search to solve the N-Queens problem."""
    current_state = [random.randint(0, n - 1) for _ in range(n)]
    current_cost = calculate_cost(current_state)

    for iteration in range(max_iterations):
        if current_cost == 0:  # Found a solution
            return current_state

        neighbors = get_neighbors(current_state)
```

```
        neighbor_costs = [(neighbor, calculate_cost(neighbor)) for
neighbor in neighbors]
        next_state, next_cost = min(neighbor_costs, key=lambda x: x[1])

        if next_cost >= current_cost:  # No improvement found
            print(f"Local maximum reached at iteration {iteration}.
Restarting...")
            return None  # Restart with a new random state

        current_state, current_cost = next_state, next_cost
        print(f"Iteration {iteration}: Current state: {current_state},
Cost: {current_cost}")

    print(f"Max iterations reached without finding a solution.")
    return None

# Get user-defined input for the number of queens
try:
    n = int(input("Enter the number of queens (N): "))
    if n <= 0:
        raise ValueError("N must be a positive integer.")
except ValueError as e:
    print(e)
    n = 4  # Default to 4 if input is invalid

solution = None

# Keep trying until a solution is found
while solution is None:
    solution = hill_climbing(n)

print(f"Solution found: {solution}")
```

Output

```
Enter the number of queens (N): 4
Iteration 0: Current state: [3, 0, 2, 0], Cost: 2
Iteration 1: Current state: [3, 0, 2, 1], Cost: 1
Local maximum reached at iteration 2. Restarting...
Iteration 0: Current state: [2, 2, 0, 3], Cost: 2
Iteration 1: Current state: [1, 2, 0, 3], Cost: 1
Local maximum reached at iteration 2. Restarting...
Iteration 0: Current state: [1, 3, 0, 2], Cost: 0
Solution found: [1, 3, 0, 2]
```

```
Enter the number of queens (N): 8
Iteration 0: Current state: [1, 3, 0, 6, 1, 7, 2, 4], Cost: 3
Iteration 1: Current state: [1, 5, 0, 6, 1, 7, 2, 4], Cost: 2
Iteration 2: Current state: [1, 5, 0, 6, 3, 7, 2, 4], Cost: 0
Solution found: [1, 5, 0, 6, 3, 7, 2, 4]
```