1. **Implement Hill Climbing search algorithm to solve N-Queens problem.**

```
import random
def calculate_cost(state):
    attacking_pairs = 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):
                attacking_pairs += 1
    return attacking_pairs

def generate_neighbours(state):
    neighbours = []
    n = len(state)
    for i in range(n):
        for j in range(i + 1, n):
            neighbour = state[:]
            neighbour[i], neighbour[j] = neighbour[j], neighbour[i]
            neighbours.append(neighbour)
    return neighbours

def print_board(state):
    n = len(state)
    for row in range(n):
        line = ""
        for col in range(n):
            if state[col] == row:
                line += "Q "
            else:
                line += ". "
        print(line)
```

```
    print()

def hill_climbing(initial_state):
    current_state = initial_state
    current_cost = calculate_cost(current_state)
    step = 0

    print(f"Step {step}: Initial state")
    print_board(current_state)
    print(f"Cost = {current_cost}\n")

    while True:
        step += 1
        neighbours = generate_neighbours(current_state)
        neighbour_costs = [calculate_cost(neighbour) for neighbour in
neighbours]
        min_cost = min(neighbour_costs)

        if min_cost >= current_cost:
            print(f"Step {step}: Reached local minimum")
            print(f"Final state:")
            print_board(current_state)
            print(f"Final cost = {current_cost}\n")
            return current_state, current_cost
        else:
            best_neighbour = neighbours[neighbour_costs.index(min_cost)]
            print(f"Step {step}: Move to better neighbour")
            print_board(best_neighbour)
            print(f"Cost = {min_cost}\n")

            current_state = best_neighbour
            current_cost = min_cost
```

```
initial_state = [3, 1, 2, 0]


final_state, final_cost = hill_climbing(initial_state)
```

OUTPUT:

```
Step 0: Initial state
. . . Q
. Q . .
. . Q .
Q . . .

Cost = 2

Step 1: Move to better neighbour
. . . Q
Q . . .
. . Q .
. Q . .

Cost = 1

Step 2: Move to better neighbour
. . Q .
Q . . .
. . . Q
. Q . .

Cost = 0

Step 3: Reached local minimum
Final state:
. . Q .
Q . . .
. . . Q
. Q . .

Final cost = 0
```