HILL CLIMB SEARCH AGGORITHM FOR N-QUEEN

```
import random
def calculate_attacks(board):
    """Calculate the number of attacking pairs of queens."""
    n = len(board)
    attacks = 0
    for i in range(n):
        for j in range(i + 1, n):
            if board[i] == board[j]:  # Same row
                attacks += 1
            if abs(board[i] - board[j]) == abs(i - j):  # Same diagonal
                attacks += 1
    return attacks

def cost_function(board):
    """Cost function is equivalent to the number of attacking pairs of queens."""
    return calculate_attacks(board)

def hill_climbing_with_initial_state(n, initial_board):
    """Solve the N-Queens problem using Hill Climbing with an initial board configuration."""
    board = initial_board[:]
    current_cost = cost_function(board)

    while True:
        # If no attacking pairs, we've found the solution
        if current_cost == 0:
            return board, current_cost
```

```
        # Generate neighbors (by moving one queen in each column)
        neighbors = []
        for col in range(n):
            for row in range(n):
                if row != board[col]:  # Don't move to the current position
                    new_board = board[:]
                    new_board[col] = row
                    neighbors.append((new_board, cost_function(new_board)))

        # Find the neighbor with the lowest cost
        neighbors.sort(key=lambda x: x[1])  # Sort by cost (attacking pairs)
        best_neighbor = neighbors[0]

        # If no better neighbor (i.e., the best neighbor has the same cost), we're stuck
        if best_neighbor[1] >= current_cost:
            return board, current_cost  # Local maxima reached, no solution found

        # Otherwise, move to the best neighbor
        board, current_cost = best_neighbor

# Example usage:
if __name__ == "__main__":
    # Take user input for board size
    n = int(input("Enter the size of the board (N): "))

    # Take user input for the initial configuration of queens (one queen per row)
    print("Enter the initial configuration of queens (one queen per row):")
    initial_board = []
    for i in range(n):
        column = int(input(f"Row {i+1}: Enter the column index for queen (0 to {n-1}): "))
        initial_board.append(column)
```

```
    # Apply hill climbing with the initial board
    solution, cost = hill_climbing_with_initial_state(n, initial_board)

    # Print the result
    if cost == 0:
        print("\nSolution found:", solution)
    else:
        print("\nNo solution found, local maxima reached.")
    print("Number of attacks (cost):", cost)
```

Output:

```
Enter the size of the board (N): 4
Enter the initial configuration of queens (one queen per row):
Row 1: Enter the column index for queen (0 to 3): 3
Row 2: Enter the column index for queen (0 to 3): 3
Row 3: Enter the column index for queen (0 to 3): 3
Row 4: Enter the column index for queen (0 to 3): 3

Solution found: [2, 0, 3, 1]
Number of attacks (cost): 0
```