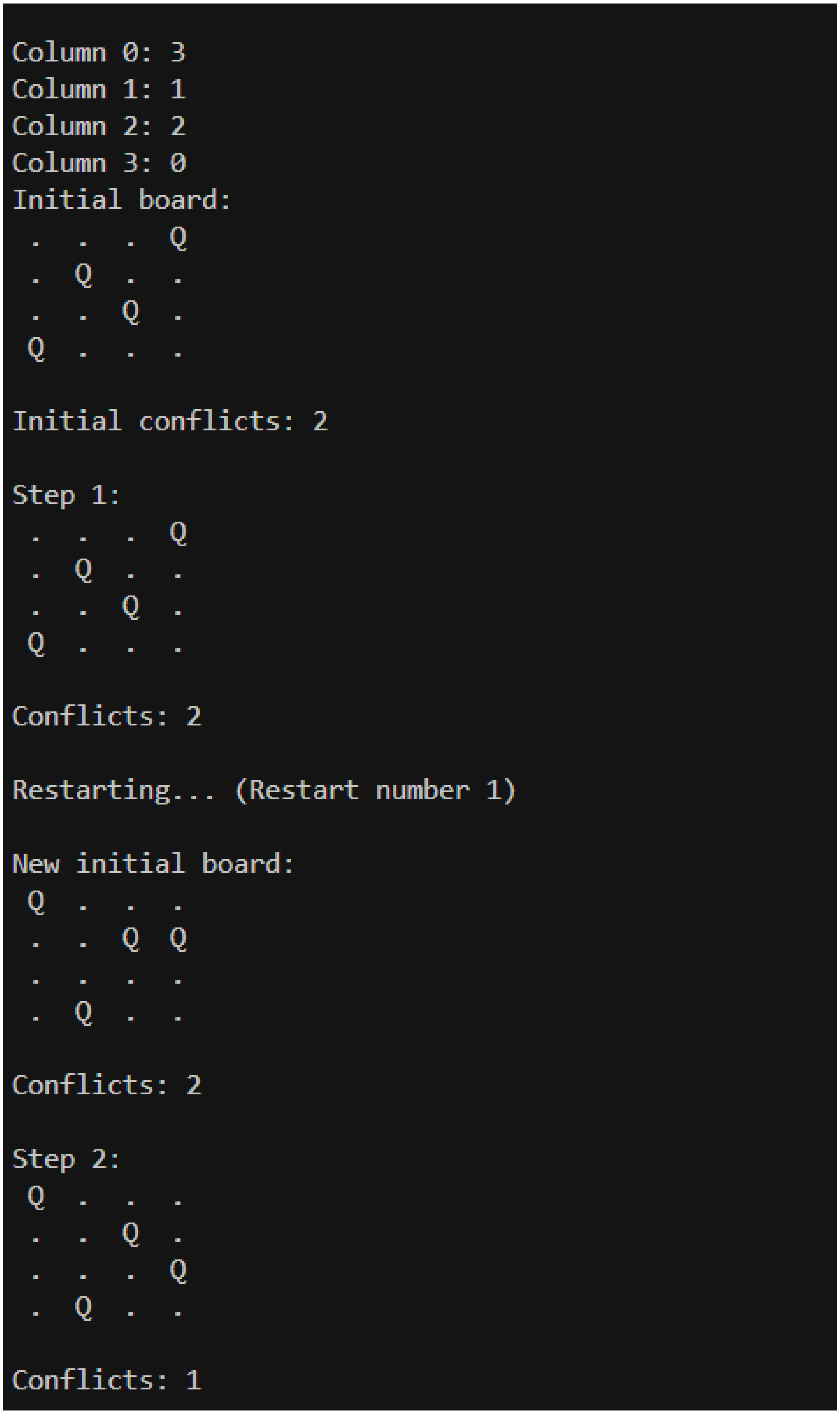
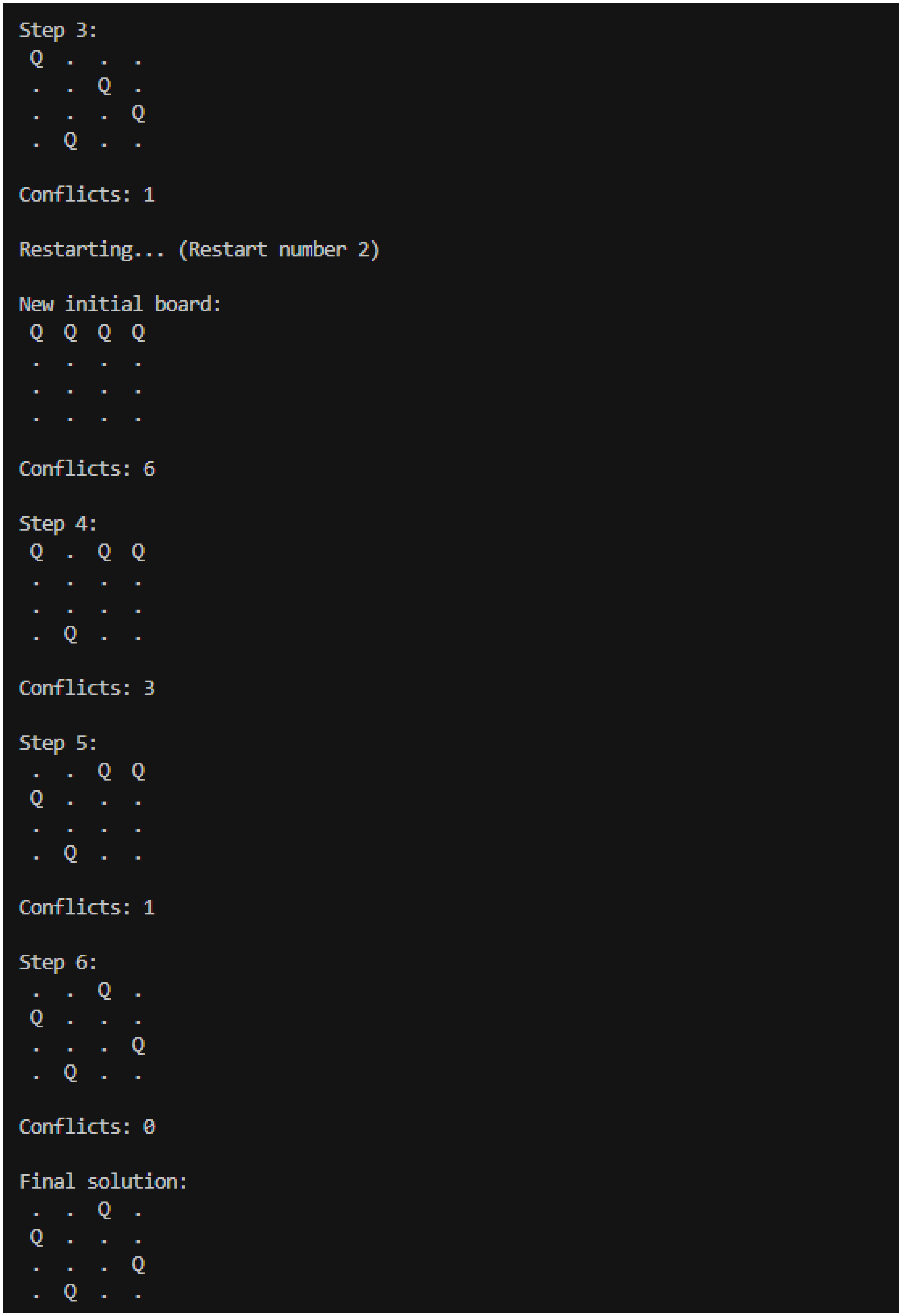
N QUEENS USING HILL CLIMBING SEARCHING

```
import random

def print_board(board, n):
    """Prints the current state of the board."""
    for row in range(n):
        line = ""
        for col in range(n):
            if board[col] == row:
                line += " Q "
            else:
                line += " . "
        print(line)
    print()

def calculate_conflicts(board, n):
    """Calculates the number of conflicts (attacks) between queens."""
    conflicts = 0
    for i in range(n):
        for j in range(i + 1, n):
            # Check if queens are in the same row or diagonal
            if board[i] == board[j] or abs(board[i] - board[j]) == abs(i - j):
                conflicts += 1
    return conflicts

def get_best_neighbor(board, n):
    """
    Finds the best neighboring board with the fewest conflicts.
    Returns the best board and its conflict count.
    """
    current_conflicts = calculate_conflicts(board, n)
    best_board = board[:]
    best_conflicts = current_conflicts
    neighbors = []

    for col in range(n):
        original_row = board[col]
        for row in range(n):
            if row == original_row:
                continue
            # Move queen to a new row and calculate conflicts
            board[col] = row
            new_conflicts = calculate_conflicts(board, n)
            neighbors.append((board[:], new_conflicts))
        # Restore the original row before moving to the next column
        board[col] = original_row

    # Sort neighbors by the number of conflicts (ascending)
    neighbors.sort(key=lambda x: x[1])
    if neighbors:
        best_neighbor = neighbors[0]
        if best_neighbor[1] < best_conflicts:
            return best_neighbor
    return board, current_conflicts

def hill_climbing_with_restarts(n, initial_board, max_restarts=100):
    """
    Performs Hill Climbing with random restarts to solve the N-Queens
problem.
    Returns the final board configuration and its conflict count.
    """
    current_board = initial_board[:]
    current_conflicts = calculate_conflicts(current_board, n)

    print("Initial board:")
    print_board(current_board, n)
    print(f"Initial conflicts: {current_conflicts}\n")

    steps = 0
    restarts = 0

    while current_conflicts > 0 and restarts < max_restarts:
        new_board, new_conflicts = get_best_neighbor(current_board, n)

        steps += 1
        print(f"Step {steps}:")
        print_board(new_board, n)
        print(f"Conflicts: {new_conflicts}\n")

        if new_conflicts < current_conflicts:
            current_board = new_board
            current_conflicts = new_conflicts
        else:
            # If no better neighbor is found, perform a random restart
            restarts += 1
            print(f"Restarting... (Restart number {restarts})\n")
            current_board = [random.randint(0, n-1) for _ in range(n)]
            current_conflicts = calculate_conflicts(current_board, n)
            print("New initial board:")
            print_board(current_board, n)
            print(f"Conflicts: {current_conflicts}\n")

    return current_board, current_conflicts

# Main function
def main():
    n = 4
    print("Enter the initial positions of queens (row numbers from 0 to 3 for
each column):")
    initial_board = []
    for i in range(n):
        while True:
            try:
                row = int(input(f"Column {i}: "))
                if 0 <= row < n:
                    initial_board.append(row)
                    break
                else:
                    print(f"Please enter a number between 0 and {n-1}.")
            except ValueError:
                print("Invalid input. Please enter an integer.")

    solution, conflicts = hill_climbing_with_restarts(n, initial_board)

    print("Final solution:")
    print_board(solution, n)
    if conflicts == 0:
        print("A solution was found with no conflicts!")
    else:
        print(f"No solution was found after {100} restarts. Final number of
conflicts: {conflicts}")

if __name__ == "__main__":
    main()
```

OUPUT :



```
Column 0: 3
Column 1: 1
Column 2: 2
Column 3: 0
Initial board:
 .  .  .  Q
 .  Q  .  .
 .  .  Q  .
 Q  .  .  .

Initial conflicts: 2

Step 1:
 .  .  .  Q
 .  Q  .  .
 .  .  Q  .
 Q  .  .  .

Conflicts: 2

Restarting... (Restart number 1)

New initial board:
 Q  .  .  .
 .  .  Q  Q
 .  .  .  .
 .  Q  .  .

Conflicts: 2

Step 2:
 Q  .  .  .
 .  .  Q  .
 .  .  .  Q
 .  Q  .  .

Conflicts: 1
```



```
Step 3:
 Q  .  .  .
 .  .  Q  .
 .  .  .  Q
 .  Q  .  .

Conflicts: 1

Restarting... (Restart number 2)

New initial board:
 Q  Q  Q  Q
 .  .  .  .
 .  .  .  .
 .  .  .  .

Conflicts: 6

Step 4:
 Q  .  Q  Q
 .  .  .  .
 .  .  .  .
 .  Q  .  .

Conflicts: 3

Step 5:
 .  .  Q  Q
 Q  .  .  .
 .  .  .  .
 .  Q  .  .

Conflicts: 1

Step 6:
 .  .  Q  .
 Q  .  .  .
 .  .  .  Q
 .  Q  .  .

Conflicts: 0

Final solution:
 .  .  Q  .
 Q  .  .  .
 .  .  .  Q
 .  Q  .  .
```