WEEK 4

Implement Hill Climbing search algorithm to solve N-Queens problem

CODE:

```
import random

def calculate_heuristic(board, n):
    """Calculates the number of pairs of queens attacking each other."""
    heuristic_value = 0
    for i in range(n):
        for j in range(i + 1, n):
            if board[i] == board[j] or abs(board[i] - board[j]) == abs(i -
j):
                heuristic_value += 1
    return heuristic_value

def hill_climbing_nqueens(n, initial_board):
    """Solves the N-Queens problem using hill climbing starting from an
initial board."""
    current_board = initial_board
    current_heuristic = calculate_heuristic(current_board, n)

    while True:
        neighbors = []
        for row in range(n):
            for col in range(n):
                if current_board[row] != col:
                    # Generate a neighbor by changing the queen's position
in the row
                    neighbor_board = current_board[:]
                    neighbor_board[row] = col
                    neighbors.append((neighbor_board,
calculate_heuristic(neighbor_board, n)))

        # Select the neighbor with the lowest heuristic value
        best_neighbor, best_neighbor_heuristic = min(neighbors, key=lambda
x: x[1])
```

```
        # If no improvement is found, break (local optimum)
        if best_neighbor_heuristic >= current_heuristic:
            break

        # Move to the best neighbor
        current_board = best_neighbor
        current_heuristic = best_neighbor_heuristic

    return current_board, current_heuristic

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

# Take user input for the board size (n) and initial state
try:
    n = int(input("Enter the number of queens (n): "))
    if n < 4:
        print("No solution exists for N less than 4.")
    else:
        # Take user input for the initial state of the board
        initial_board = []
        print("Enter the initial board state (row positions for each column
from 0 to n-1):")
        for i in range(n):
            row_position = int(input(f"Position of queen in row {i} (0 to
{n-1}): "))
            initial_board.append(row_position)

        solution, heuristic = hill_climbing_nqueens(n, initial_board)
```

```
        # Goal state is defined as a board with a heuristic of 0 (no
attacking queens)
        if heuristic == 0:
            print("Solution found:")
            print_board(solution)
        else:
            print("Local optima reached, not a perfect solution.")
            print("Heuristic value:", heuristic)
            print_board(solution)
except ValueError:
    print("Please enter valid integers for board size and initial
positions.")
```

OUTPUT:

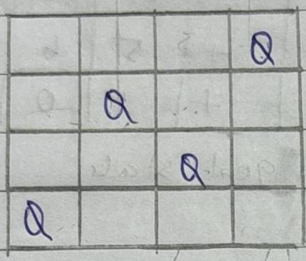
```
Enter the number of queens (n): 4
Enter the initial board state (row positions for each column from 0 to n-1):
Position of queen in row 0 (0 to 3): 4
Position of queen in row 1 (0 to 3): 2
Position of queen in row 2 (0 to 3): 3
Position of queen in row 3 (0 to 3): 1
Local optima reached, not a perfect solution.
Heuristic value: 1
Q . . .
. . Q .
. . . Q
. Q . .
```

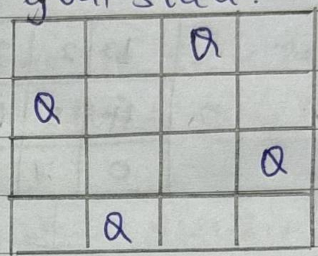
22/10/24 Week-4

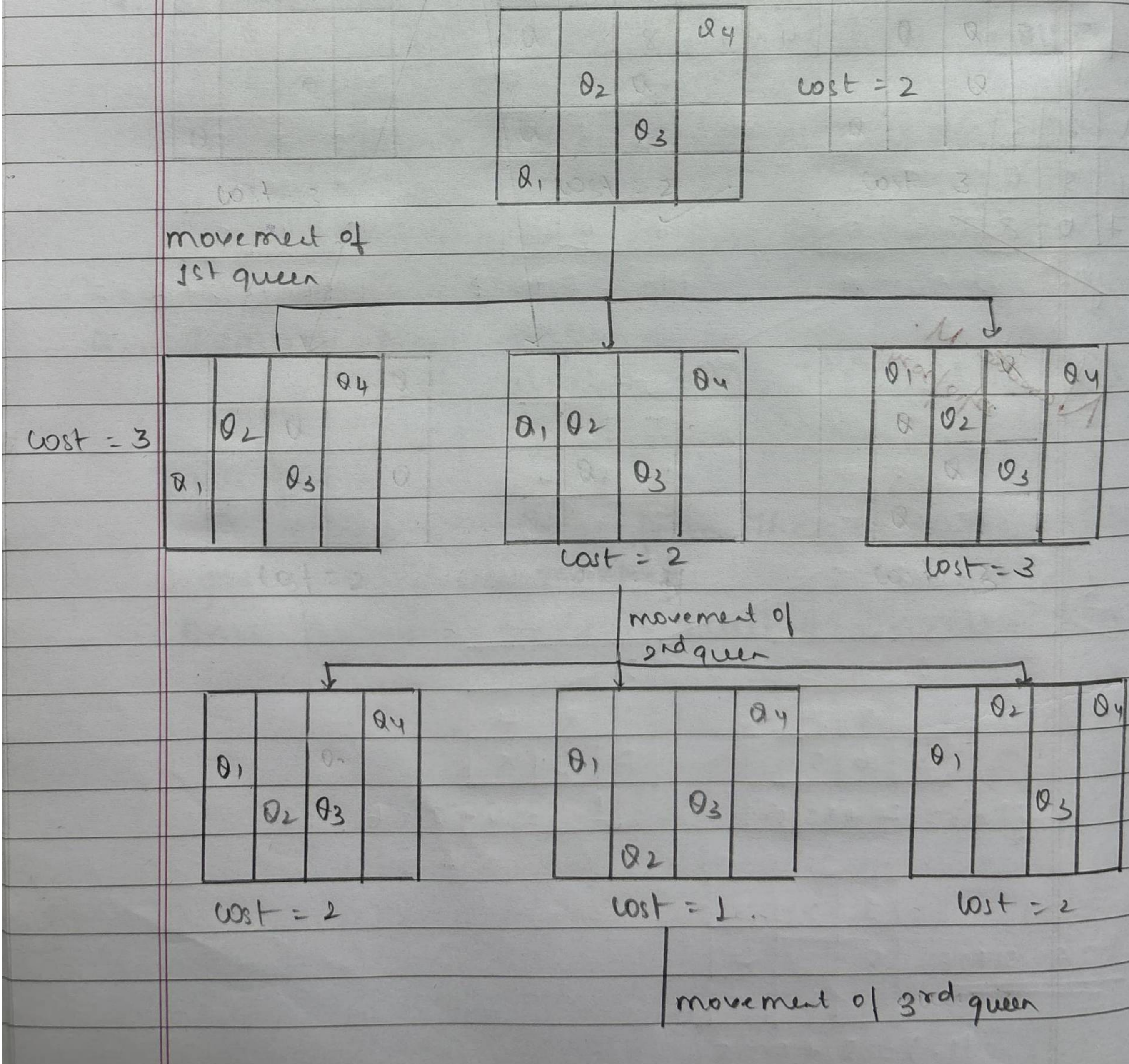
Implement Hill Climbing search algorithm to solve 4 queens problem

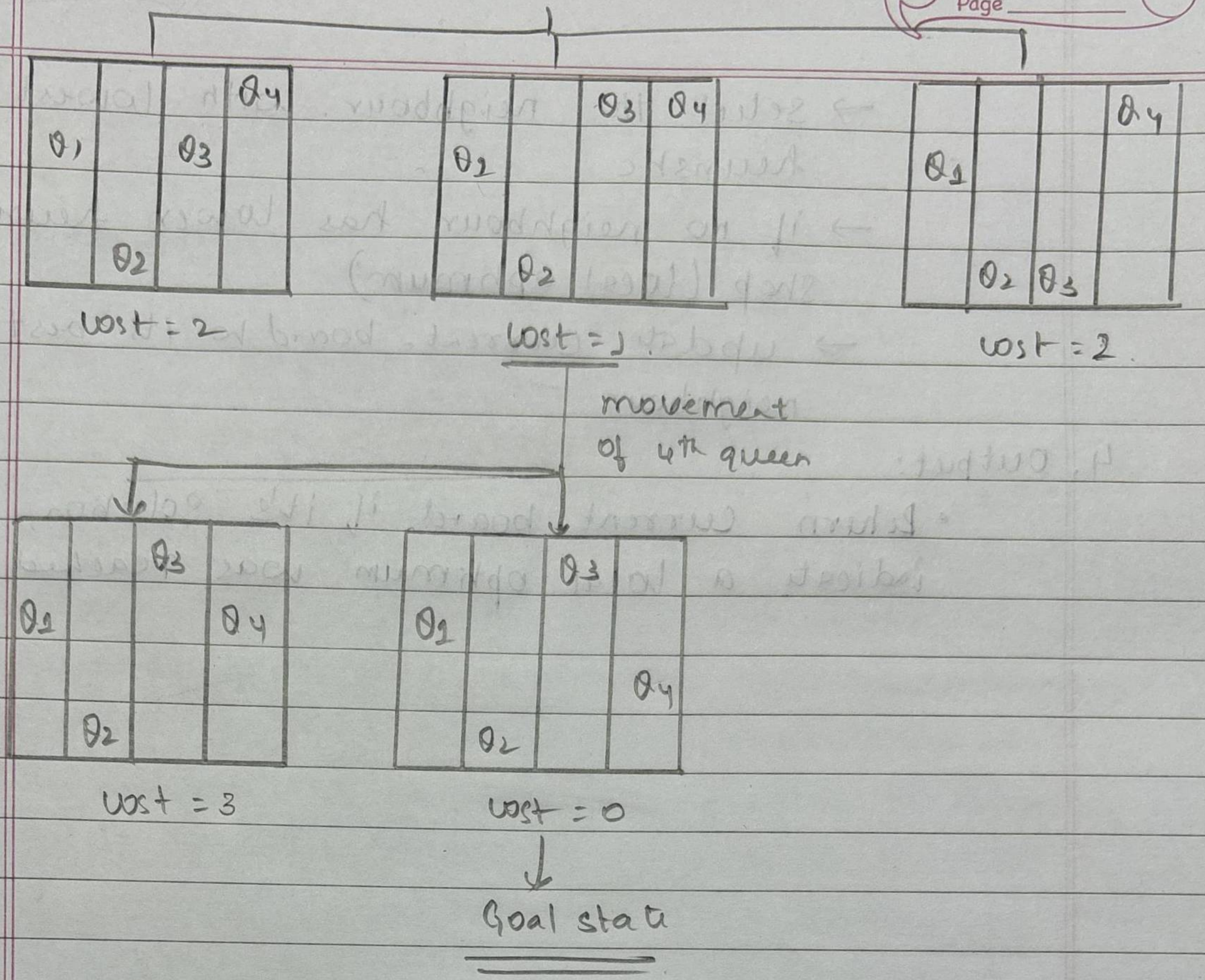
Initial state
$x_0 = 3 \quad x_1 = 1 \quad x_2 = 2 \quad x_3 = 0$

goal state:









Algorithm:

1. Input:
   - n: Board size (number of queens)
   - Initial_board: inital queen position in each row.

2. Heuristic calculation:
   - count pairs of queens attacking each other (same column or diagonal)

3. Hill climbing search
   - set current_board to initial board and calculate its heuristic
   - Repeat:
     → generate neighbouring boards by moving each queen to other columns in its row

→ Select the neighbour with lowest heuristic

→ if no neighbour has lower heuristic stop (local optimum)

→ update current_board to its best neighbour

4. Output:

- Return current_board if it's solution, other indicate a local optimum was reached.