```python
from random import randint

N = 8

# A utility function that configures
# the 2D array "board" and
# array "state" randomly to provide
# a starting point for the algorithm.
def configureRandomly(board, state):

    # Iterating through the
    # column indices
    for i in range(N):
        # Getting a random row index
        state[i] = randint(0, 100000) % N

        # Placing a queen on the
        # obtained place in
        # chessboard.
        board[state[i]][i] = 1
```

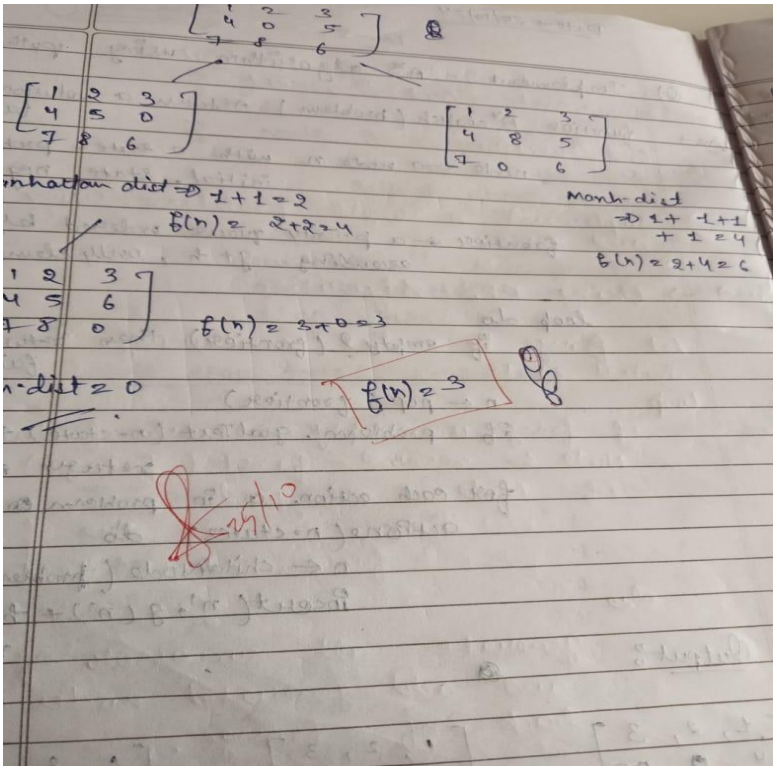
```python
def printBoard(board):
    print("Solution Found!")
    print("Name: Vyom Gupta")
    print("USN: 1BM22CS333")
    print("Board Configuration:")
    for i in range(N):
        print(*board[i])

# A utility function that prints
# the array "state".
def printState(state):
    print(*state)

# A utility function that compares
# two arrays, state1 and state2 and
# returns True if equal
# and False otherwise.
def compareStates(state1, state2):
    for i in range(N):
        if (state1[i] != state2[i]):
            return False
    return True

# A utility function that fills
# the 2D array "board" with
# values "value"
```

```python
def fill(board, value):
    for i in range(N):
        for j in range(N):
            board[i][j] = value

# This function calculates the
# objective value of the
# state(queens attacking each other)
# using the board by the
# following logic.
def calculateObjective(board, state):
    attacking = 0

    # Checking each queen
    for i in range(N):
        row = state[i]
        col = i - 1

        # Check for left attacks (same row)
        while col >= 0 and board[row][col] != 1:
            col -= 1
        if col >= 0 and board[row][col] == 1:
            attacking += 1

        # Check for right attacks (same row)
        row = state[i]
        col = i + 1
        while col < N and board[row][col] != 1:
            col += 1
        if col < N and board[row][col] == 1:
```

```
        # Diagonal left-up
        row = state[i] - 1
        col = i - 1
        while col >= 0 and row >= 0 and board[row][col] != 1:
            col -= 1
            row -= 1
        if col >= 0 and row >= 0 and board[row][col] == 1:
            attacking += 1

        # Diagonal right-down
        row = state[i] + 1
        col = i + 1
        while col < N and row < N and board[row][col] != 1:
            col += 1
            row += 1
        if col < N and row < N and board[row][col] == 1:
            attacking += 1

        # Diagonal left-down
        row = state[i] + 1
        col = i - 1
        while col >= 0 and row < N and board[row][col] != 1:
            col -= 1
            row += 1
        if col >= 0 and row < N and board[row][col] == 1:
            attacking += 1

        # Diagonal right-up
        row = state[i] - 1
        col = i + 1
```

```
    fill(board, 0)
    generateBoard(board, state)

def hillClimbing(board, state):
    neighbourBoard = [[0 for _ in range(N)] for _ in range(N)]
    neighbourState = [0 for _ in range(N)]

    copyState(neighbourState, state)
    generateBoard(neighbourBoard, neighbourState)

    while True:
        copyState(state, neighbourState)
        generateBoard(board, state)

        getNeighbour(neighbourBoard, neighbourState)

        if compareStates(state, neighbourState):
            printBoard(board)
            break

        elif calculateObjective(board, state) == calculateObjective(neighbourBoard, neighbourState):
            neighbourState[randint(0, 100000) % N] = randint(0, 100000) % N
            generateBoard(neighbourBoard, neighbourState)

# Driver code
state = [0] * N
board = [[0 for _ in range(N)] for _ in range(N)]

configureRandomly(board, state)
```

```
Solution Found!
Name: Vyom Gupta
USN: 1BM22CS333
Board Configuration:
0 0 0 0 0 1 0 0
0 0 0 1 0 0 0 0
0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1
0 0 0 0 1 0 0 0
0 0 0 0 0 0 1 0
1 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0
```


Manh. dist
→ 1+1+1+1 = 4
f(n) = 2+4 = 6
f(n) = 2+2 = 4
f(n) = 3+0 = 3
f(n) = 3