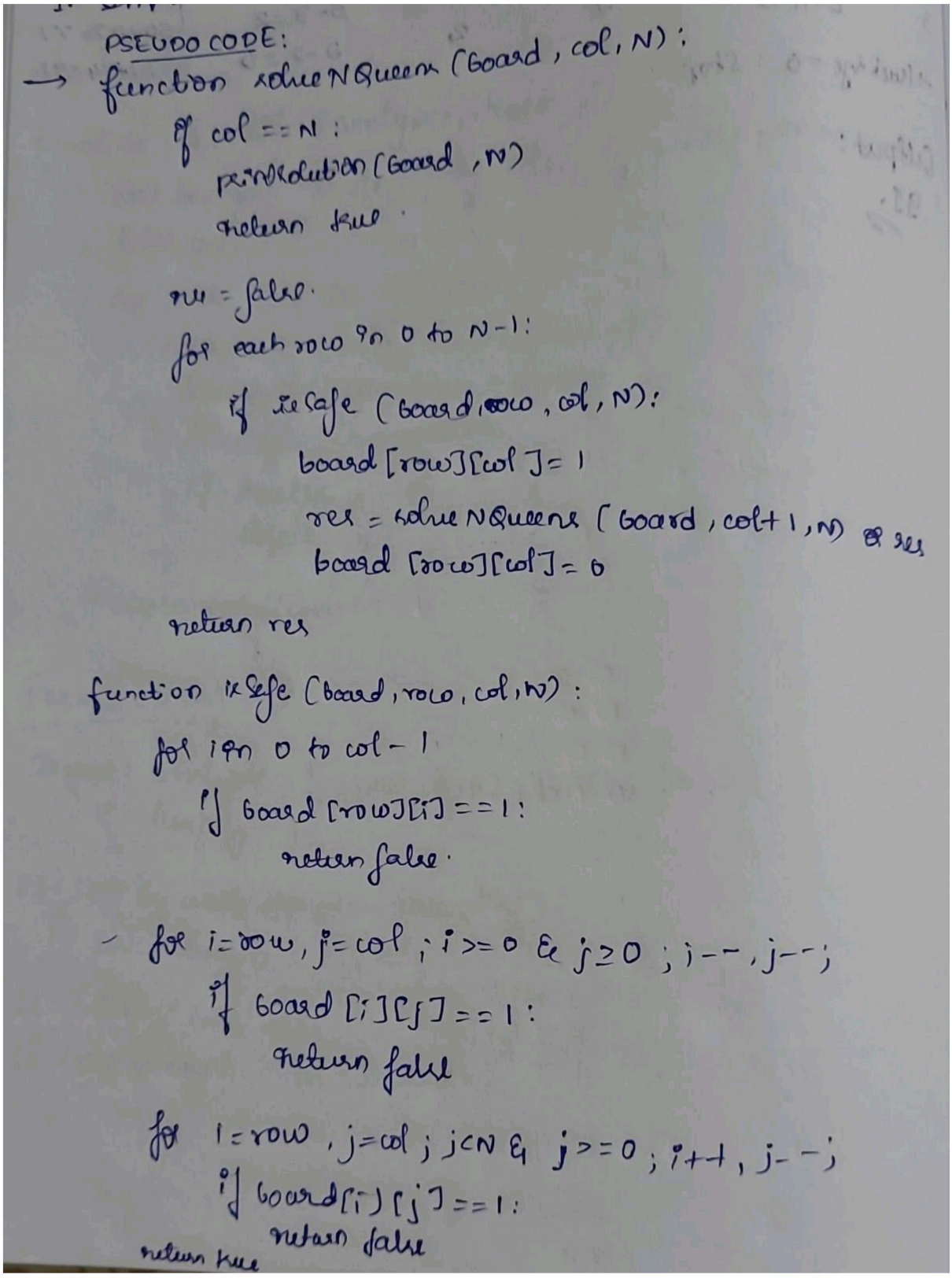
PROGRAM 9

Implement "N-Queens Problem" using Backtracking.

ALGORITHM



```
PSEUDO CODE:
function solveNQueens(board, col, N):
    if col == N:
        printSolution(board, N)
        return true

    res = false
    for each row in 0 to N-1:
        if isSafe(board, row, col, N):
            board[row][col] = 1
            res = solveNQueens(board, col+1, N) || res
            board[row][col] = 0

    return res

function isSafe(board, row, col, N):
    for i in 0 to col-1
        if board[row][i] == 1:
            return false

    for i=row, j=col; i>=0 && j>=0; i--, j--;
        if board[i][j] == 1:
            return false

    for i=row, j=col; j<N && j>=0; i++, j--;
        if board[i][j] == 1:
            return false
return true
```

CODE

```
#include <stdio.h>
#include <stdlib.h>

#define MAX 20

int board[MAX];
int N;

int isSafe(int row, int col) {
    for (int i = 0; i < row; i++) {
        if (board[i] == col || abs(board[i] - col) == abs(i - row))
            return 0;
    }
    return 1;
}

void printSolution() {
    printf("\nSolution:\n");
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            if (board[i] == j)
                printf("Q ");
            else
                printf(". ");
        }
        printf("\n");
    }
}
```

```
int solveNQueens(int row) {
    if (row == N) {
        printSolution();
        return 1;
    }

    int found = 0;
    for (int col = 0; col < N; col++) {
        if (isSafe(row, col)) {
            board[row] = col;
            found |= solveNQueens(row + 1);  // Try next row
        }
    }

    return found;
}

int main() {
    printf("Enter number of queens (N): ");
    scanf("%d", &N);

    if (N < 1 || N > MAX) {
        printf("N should be between 1 and %d\n", MAX);
        return 1;
    }

    if (!solveNQueens(0))
        printf("No solution exists for N = %d\n", N);

    return 0;}
```

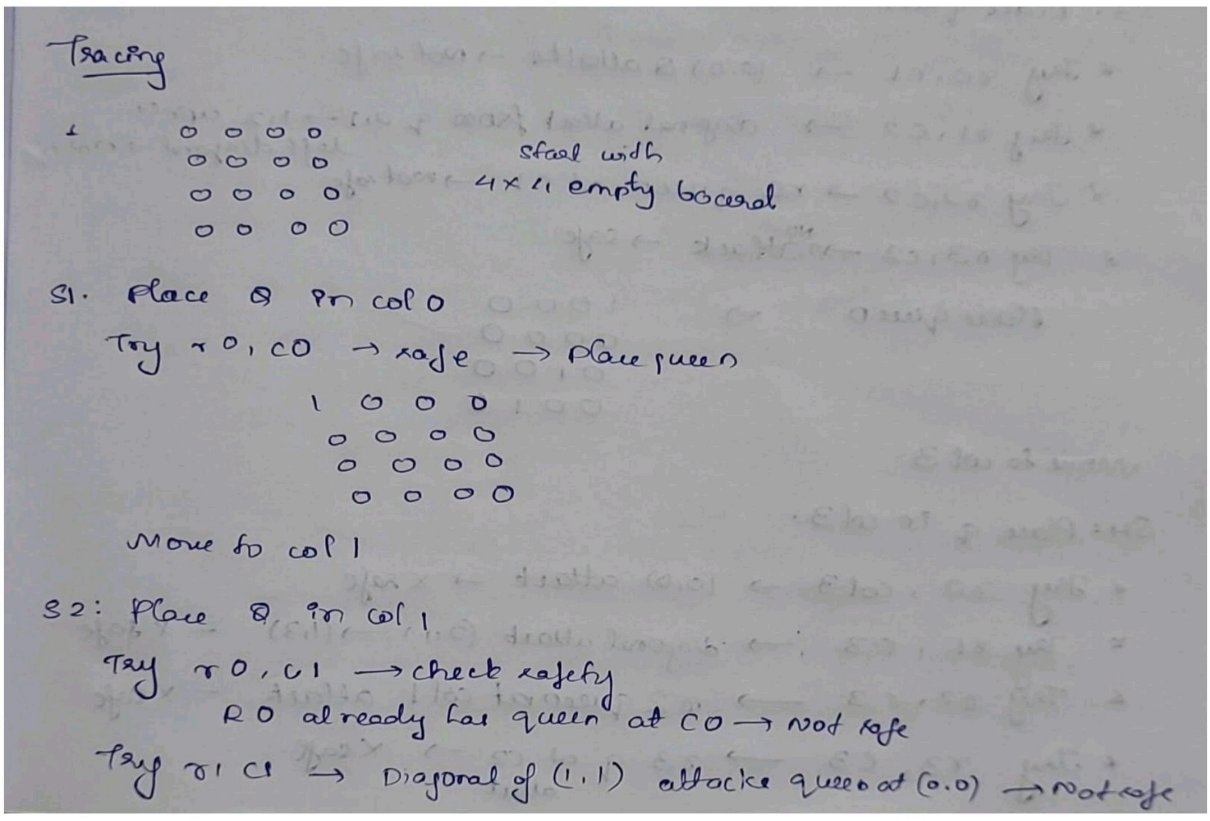
OUTPUT

```
Enter number of queens (N): 4

Solution:
. Q . .
. . . Q
Q . . .
. . Q .

Solution:
. . Q .
Q . . .
. . . Q
. Q . .
```

TRACING



* Try r0, c2
  * No q in same row, no diagonal attack → safe
    Place queen

1 0 0 0
0 0 0 0
0 1 0 0
0 0 0 0

move to col 2.

S3: Place queen in col 2.

* Try r0, c1 → (0,0) Q attacks → Not safe.
* Try r1, c2 → diagonal attack from q 2,1 → 1,2 upper left diagonal → not safe
* Try r2, c2 → r2 queen at c1 → not safe
* Try r3, c2 → No attack → safe

Place queen ⇒

1 0 0 0
0 0 0 0
0 1 0 0
0 0 1 0

Move to col 3

S4: Place q in col 3.

* Try r0, col 3 → (0,0) attack → x safe
* Try r1, c3 → diagonal attack (2,1) → (1,3) — x safe
* Try r2, c3 → r2 queen at col 1 attack — x safe
* Try r3, c3 → r3 q at c2 → x safe (attack)

No valid pos in col 3 → Backtrack

Backtrack S1: Remove q from (3,2)

1 0 0 0
0 0 0 0
0 1 0 0
0 0 0 0

S3 continued: Try next rows in col 2

* No more rows in col 2 — Backtrack again.

Backtrack S2: Remove q from (2,1)

⇒

1 0 0 0
0 0 0 0
0 0 0 0
0 0 0 0

S2 → continued: Try next rows in col 1

* Try r3, c1 → No attack safe → Place queen

1 0 0 0
0 0 0 0
0 0 0 0
0 1 0 0

S3 → continued: Place queen in col 2.

* Try r0, c2 → Q(0,0) attacks — x safe
* Try r1, c2 → Diagonal attack from (3,1) → x safe.
* Try r2, c2 → No attack — safe. → Place queen

1 0 0 0
0 0 0 0
0 0 1 0
0 1 0 0

S4: Place Q in col 3.

- Try (0,3) → % (0,0) attack – ✗ Not safe
- Try (1,3) → No attack – ✓ safe

Place queen.

1 0 0 0
0 0 0 1
0 0 1 0
0 1 0 0

Final step: All queens placed

1 0 0 0
0 0 0 1
0 0 1 0
0 1 0 0