LAB 4 Task

Name : Saad Masood

Roll Number: SU92-BSAIM-F23-039

Department: BS Artificial Intelligence (AI)-4A

Subject: Programming for AI (Lab)

Submitted : Sir Rasikh Ali

N-QUEEN

CODE

```
n=int(input("Enter the board size: "))
board=[[0 for _ in range(n)] for _ in range(n)]

def check_column(board,row,column):
    for i in range(row):
        if board[i][column]==1:
            return False
    return True

def check_diagonal(board,row,column,n):
    i,j=row,column
    while i>=0 and j>=0:
        if board[i][j]==1:
            return False
        i-=1
        j-=1
    i,j=row,column
    while i>=0 and j<n:
        if board[i][j]==1:
            return False
        i-=1
        j+=1
    return True
def nqn(board, row):
    if row==n:
        for b in board:
            print(b)
        print()
        return True
    for i in range(n):
        if check_column(board,row,i) and check_diagonal(board,row,i,n):
            board[row][i]=1
            if nqn(board, row + 1):
                return True
            board[row][i]=0
    return False

if not nqn(board, 0):
    print("No solution exists.")
```

CODE EXPLAINANTION

1. Input and Board Initialization

```
n=int(input("Enter the board size: "))
board=[[0 for _ in range(n)] for _ in range(n)]
```

- The user inputs the board size n, which represents the number of queens and the board dimensions.
- A 2D list board is initialized with zeros, where:
  - 0 represents an empty cell.
  - 1 represents a queen.

2. Column Safety Check

```
def check_column(board,row,column):
        for i in range(row):
            if board[i][column]==1:
                return False
        return True
```

- This function ensures no other queen is placed in the same column above the current row.
- Since queens are placed row by row from top to bottom, we only check previous rows.

3. Diagonal Check

```
def check_diagonal(board,row,column,n):
    i,j=row,column
    while i>=0 and j>=0:
        if board[i][j]==1:
            return False
        i-=1
        j-=1
```

- This part checks the **upper-left diagonal** to ensure no other queen is placed there.

```
i,j=row,column
while i>=0 and j<n:
    if board[i][j]==1:
        return False
    i-=1
    j+=1
return True
```

- This part checks the **upper-right diagonal** for the same condition.

4. Backtracking Algorithm:

```
def nqn(board, row):
    if row==n:
        for b in board:
            print(b)
        print()
        return True
```

If all n rows are successfully filled, the board is printed as a solution.

```
for i in range(n):
        if check_column(board,row,i) and check_diagonal(board,row,i,n):
            board[row][i]=1
            if nqn(board, row + 1):
                return True
            board[row][i]=0
    return False
```

- Tries placing a queen in each column of the current row.
- If a valid position is found, the function calls itself for the next row.
- If no solution is found in the next row, **backtracking** occurs (resets the position to 0).

5. No Solution Case

```
if not nqn(board, 0):
    print("No solution exists.")
```

- If no valid arrangement is found, it prints **"No solution exists."**

Why This Works (Backtracking Approach)

- The algorithm **places queens row by row**, ensuring safety in columns and diagonals.
- If a dead end is reached, it **backtracks** and tries another position.
- This ensures all possibilities are explored efficiently.