Experiment No.: 1 - N Queens Problem

Aim:

To solve the N-Queens problem using the backtracking algorithm in Python, where the goal is to place N queens on an N×N chessboard such that no two queens threaten each other. The program will take N as input from the user and provide all possible solutions for placing the queens.

Algorithm:

1. Initialize the Board: Create an N×N board initialized with zeros (0). Each cell represents an empty slot where a queen might be placed.
2. Define the Backtracking function: Create a recursive function solveQueens(board, col) that attempts to place queens on the board column by column. If the column index equals N, it means all queens are successfully placed, and we have a found a solution.
3. Check Safety: Create a helper function isSafe(board, row, col) to check whether placing a queen at board[row][col] is safe. Ensure no other queens are on the same row, column or diagonal.
4. Place the Queen: For each row in the current column, check if it's safe to place a queen. If safe, place the queen and make a recursive call to place queens in the next column.
5. Backtrack: If placing a queen in any row of the current column doesn't lead to a solution, backtrack by removing the queen and trying the next row.
6. Print the Solution: Once all queens are placed successfully, add the board configuration to the list of solution.

7. Handle Multiple solution: Continue searching for solutions by backtracking to find all possible ways to place N queens.

8. User Input: Take input N from the user to determine the size of the board and the number of queens.

Program:

```
def isSafe(board, row, col, N):
    # Check this row on left side
    for i in range(col):
        if board[row][i] == 1:
            return False

    # Check upper diagonal on left side
    for i, j in zip(range(row, -1, -1), range(col, -1, -1)):
        if board[i][j] == 1:
            return False

    # Check lower diagonal on left side
    for i, j in zip(range(row, N), range(col, -1, -1)):
        if board[i][j] == 1:
            return False

    return True

def solveNQueens(board, col, N):
    # If all queens are placed, return True
    if col >= N:
        return True

    # Consider this column and try placing the
    queen in all rows one by one
```

```
    for i in range(N):
        if isSafe(board, i, col, N):
            # Place the queen
            board[i][col] = 1

            # Recur to place the rest of the queens
            if solveNQueens(board, col+1, N):
                return True

            # If placing queen in board[i][col] doesn't
            lead to a solution, backtrack
            board[i][col] = 0
    return False

def printSolution(board, N):
    for i in range(N):
        for j in range(N):
            if board[i][j] == 1:
                print("Q", end=" ")
            else:
                print(".", end=" ")
        print()
    print("\n")

def solveNQueensProblem(N):
    board = [[0 for _ in range(N)] for _ in range(N)]
    if not solveNQueens(board, 0, N):
        print("Solution does not exist")
        return False
    printSolution(board, N)
    return True
if __name__ == "__main__":
    N = int(input("Enter the values of N (4, 6, 8, ....): "))
    solveNQueensProblem(N)
```

Output:

Input: N=4

Output:

. Q . .
. . . Q
Q . . .
. . Q .

Result:

Thus, the program for N Queens Problem is successfully executed and the output is verified.