CSP: N-QUEEN

```
def print_solution(board):
    for row in board:
        print(" ".join("Q" if cell else "." for cell in row))
    print("\n")


def solve_n_queens(n):
    board = [[0] * n for _ in range(n)]

    cols = [False] * n          # column occupied
    diag1 = [False] * (2 * n)   # '/' diagonal (row + col)
    diag2 = [False] * (2 * n)   # '\' diagonal (row - col + n)

    solutions = []

    def backtrack(row):
        if row == n:
            # Found a valid solution
            solution = [row[:] for row in board]
            solutions.append(solution)
            return

        for col in range(n):
            if not cols[col] and not diag1[row + col] and not diag2[row - col + n]:
                # Place queen
                board[row][col] = 1
                cols[col] = diag1[row + col] = diag2[row - col + n] = True
```

```
            backtrack(row + 1)

            # Remove queen (backtrack)
            board[row][col] = 0
            cols[col] = diag1[row + col] = diag2[row - col + n] = False

    backtrack(0)
    return solutions

# Example Usage
if __name__ == "__main__":
    n = 4  # You can change this to any n
    all_solutions = solve_n_queens(n)

    print(f"Total solutions for {n}-Queens: {len(all_solutions)}\n")
    for idx, sol in enumerate(all_solutions, 1):
        print(f"Solution {idx}:")
        print_solution(sol)
```

```
Total solutions for 4-Queens: 2

Solution 1:
. Q . .
. . . Q
Q . . .
. . Q .

Solution 2:
. . Q .
Q . . .
. . . Q
. Q . .
```

Possible Viva Questions on This Practical:

1. **What is the N-Queens Problem?**
    - The N-Queens Problem is a classical problem in computer science where you need to place n queens on an n x n chessboard in such a way that no two queens threaten each other. This means no two queens can share the same row, column, or diagonal.
2. **How does Backtracking work in solving the N-Queens Problem?**
    - Backtracking is used to explore all possible configurations for placing the queens. The algorithm places a queen in a column, moves to the next row, and repeats the process. If it finds that placing a queen violates any constraints (i.e., two queens threaten each other), it backtracks to the previous row and tries the next column.
3. **What is the role of the cols, diag1, and diag2 arrays?**
    - These arrays keep track of whether a column or diagonal is already occupied by a queen. This helps the algorithm efficiently check if a queen can be safely placed at a given position without checking the entire board.
4. **What is the base case in the solve function?**
    - The base case occurs when row == n, which means all queens have been placed successfully, and the solution can be printed.
5. **What happens when the algorithm can't place a queen in a row?**
    - When the algorithm cannot place a queen in any column in a given row, it backtracks by removing the queen from the previous row and trying a new position for that queen.
6. **What is the time complexity of this backtracking approach?**
    - The time complexity is generally **O(n!)** due to the fact that for each row, the algorithm attempts placing queens in n columns, leading to a recursive exploration of all potential placements.
7. **Can this method handle large values of n efficiently?**
    - For large n, the backtracking algorithm may not be efficient enough due to its factorial time complexity. Optimizations or alternative algorithms like constraint propagation may be needed for larger values of n.
8. **What would happen if the board was initialized differently, e.g., with some queens already placed?**

- If some queens are pre-placed, the algorithm would need to respect those placements while trying to place the remaining queens. This may lead to fewer solutions or no solutions, depending on the configuration.

9. **Explain how the algorithm uses Branch and Bound in this case.**
    - The algorithm uses a form of branch pruning by avoiding placing queens in positions that are already attacked (columns and diagonals marked as True). This effectively reduces the number of branches it explores.