RAJALAKSHMI INSTITUTE OF TECHNOLOGY
(An Autonomous Institution, Affiliated to Anna University, Chennai)

DEPARTMENT OF CSE (ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING)

ACADEMIC YEAR 2025 - 2026

SEMESTER III

ARTIFICIAL INTELLIGENCE LABORATORY

MINI PROJECT REPORT

REGISTER NUMBER	2117240030030
NAME	DIVYASHREE V
PROJECT TITLE	N- QUEEN PUZZLE
DATE OF SUBMISSION	28.10.2025
FACULTY IN-CHARGE	**Mrs. M. Divya**

Signature of Faculty In-charge

N-QUEENS PUZZLE — MINI PROJECT REPORT

➢ INTRODUCTION

The N-Queens puzzle is a classical problem in computer science and artificial intelligence that demonstrates the use of backtracking algorithms to explore possible configurations efficiently.

The challenge is to place N queens on an N×N chessboard so that no two queens threaten each other. This means no two queens can be in the same row, column, or diagonal.

The N-Queens problem is widely used to teach recursive algorithms, constraint satisfaction problems, and optimization techniques. It serves as a foundation for understanding complex search problems and demonstrates the importance of pruning and intelligent search strategies in AI-based problem-solving.

➢ PROBLEM STATEMENT

The problem requires placing N queens on an N×N chessboard in such a way that no two queens attack each other. The solution should represent all possible valid configurations.

For example, for N = 4, there are two valid ways to place the queens, while for N = 8, there are 92 solutions.

The challenge arises from the exponential growth in the number of possibilities as N increases. Hence, an efficient algorithmic approach is required to find solutions within a reasonable time.

➢ GOAL

The primary goal of this mini project is to design and implement a Python-based solution for the N-Queens problem using the backtracking algorithm. The objectives include:

- Understanding the concept of recursion and backtracking.

- Implementing an efficient search to find all valid queen placements.

- Displaying sample board configurations for small N values (e.g., 4 and 8).

- Analyzing the computational complexity of the algorithm.

➢ THEORETICAL BACKGROUND

The N-Queens problem is a variation of the 8-Queens puzzle first proposed in the 19th century. It belongs to the class of constraint satisfaction problems (CSPs),

where the goal is to assign values to variables under certain constraints.

In AI, backtracking is a general algorithmic technique used for solving such CSPs. It incrementally builds candidates for the solution and abandons a candidate as soon as it determines that it cannot lead to a valid complete solution — this process is called pruning.

The N-Queens problem illustrates the efficiency of backtracking over brute-force search by significantly reducing the search space.

➢ ALGORITHM EXPLANATION WITH EXAMPLE

The algorithm places queens one by one in different rows, checking for conflicts in columns and diagonals. When a safe position is found, the algorithm proceeds to the next row;

otherwise, it backtracks to the previous row and tries a new column position.

Example (N = 4):

Step 1: Place the first queen in row 0, column 1.

Step 2: Move to row 1 and place the next queen where it’s not attacked.

Step 3: Continue this until all queens are safely placed or backtrack if no position is valid.

Final valid boards for N=4 (2 solutions):

1)

```
. Q . .
```

. . . Q

Q . . .

. . Q .

2)

. . Q .

Q . . .

. . . Q

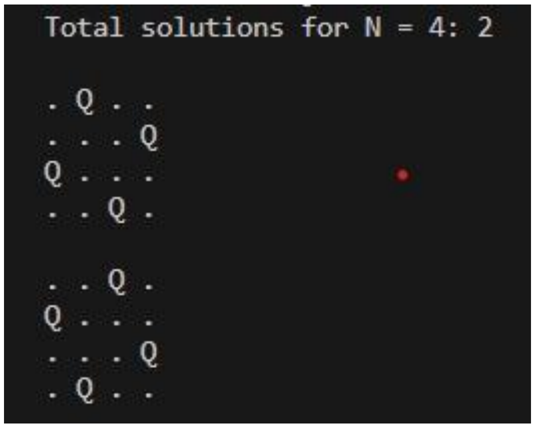
. Q .

➢ **CODE IMPLEMENTATION**

Below is the Python implementation of the N-Queens problem using backtracking:

```
# N-Queens Problem using Backtracking
def solve_n_queens(N):
    solutions = []
    board = [-1] * N   # board[i] = column position of queen in row i

    def is_safe(row, col):
        for r in range(row):
            c = board[r]
            # Check same column or diagonals
            if c == col or abs(c - col) == abs(r - row):
                return False
        return True
    def backtrack(row=0):
        if row == N:
            solutions.append(board.copy())
            return
        for col in range(N):
            if is_safe(row, col):
                board[row] = col
                backtrack(row + 1)
                board[row] = -1  # backtrack

    backtrack()
    return solutions



# Display solutions
def print_solutions(solutions, N):
    print(f"Total solutions for N = {N}: {len(solutions)}\n")
    for sol in solutions:
        for r in range(N):
            print(" ".join("Q" if sol[r] == c else "." for c in range(N)))
        print()

# Example
if __name__ == "__main__":
    N = 4
    sols = solve_n_queens(N)

    print_solutions(sols, N)
```

OUTPUT



```
Total solutions for N = 4: 2

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

. . Q .
Q . . .
. . . Q
. Q . .
```

- **RESULT**

This mini project successfully implements the N-Queens puzzle using a backtracking algorithm. It demonstrates how recursive search and pruning can be applied to solve complex combinatorial problems efficiently.

The approach is extendable to other constraint satisfaction problems such as Sudoku and graph coloring.

- **GITHUB LINK**

- **REFERENCE**

1. Russell, S. & Norvig, P. (2016). Artificial Intelligence: A Modern Approach.

2. Wikipedia - N-Queens Puzzle.

3. GeeksforGeeks - N Queen Problem using Backtracking.

4. TutorialsPoint - Backtracking Algorithm Overview.