Implementation of Bight Queens Problems.

Date: 5/3/25

Exp No: 1

Aim: To develop a python program that solves the 8-Queens problem using the Backtracking algorithm. The Program should ensure that no two queens should and attack eachother and display valid chessboard configurations.

Case Scenario.

A chessboard consists of 8×8 squares, and your task is to place 8 queens on the board such that no two queens attack eachother. Queens can attack in horizontal, Vertical, and diagonal directions.

Task Requirements

1. Problem Representation.
   Represent the 8 queen problem as a Constraint satisfaction problem.

2. Algorithm Implementation.
   Implement a solution using Backtracking

Code

```
N=8

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


def is_safe(board, row, col):
    for i in range(row):
        if board[i][col]==1:
            return false


for i,j in zip(range(row,-1,-1), range(col,-1,-1)):
    if board[i][j]==1
    return False.


for i,j in zip(range(row,-1,-1), range(col,N)):
    if board[i][j]==1:
        return False


return True


def solve_n_queens(board, row):
    if row>=N:
        print_solution(board)
        return True
```

3. Output Requirement.

Display a Valid 8x8 chessboard with queens (Q) placed in a correct manner.

4 Performance Analysis:

Compare execution time for different bord' sizes.

Procedure

1. Start
2). Initialise NxN chessboard with all empty positions ()
3): Define a position is_safe (board, row, col,N);
4). Define a recursive function solve_n_queen (board, row,
 - if row == N ? Print the Board (solution found)
 - Try placing queen in each column.
 - if is_safe () == true, place the queen and recurse for the next row.
 - if placing a queen leads to failure, Backtrack.

5) Call solve_n_queens() for the first row (row=0)

If a solution is found print the board

```
for col n range(N):
    if is_safe (board, row, col):
        Board[row][col]=1


if solve_n_queens(board, row+1):
        return True

Board[row][col]=0
 return False


def solve()

board = [[0]*N for _ in range (N)]
if not Solve_n_queens (board, 0):
    print("No Solution Exists")

  Solve()
```

Output

```
Q . . . . . . .

. . . . Q . . .

. . . . . . Q .

. . Q . . . . .

. . . . . Q . .

. Q . . . . . .

. . . Q . . . .

. . . . . . . Q
```

6) If a solution is found, print the board; else, print "No solution Exists".

7) End