Branch & bound:  
# Function to check whether it's safe to place a queen at arr[x][y]

def is\_safe(arr, x, y, n):

# Check vertical column

for row in range(x):

if arr[row][y] == 1:

return False

# Check upper-left diagonal

row, col = x, y

while row >= 0 and col >= 0:

if arr[row][col] == 1:

return False

row -= 1

col -= 1

# Check upper-right diagonal

row, col = x, y

while row >= 0 and col < n:

if arr[row][col] == 1:

return False

row -= 1

col += 1

return True

# Recursive function for placing queens using Branch and Bound

def branch\_and\_bound\_nqueen(arr, x, n):

if x >= n: # All queens placed

return True

for col in range(n):

if is\_safe(arr, x, col, n):

arr[x][col] = 1 # Place queen

if branch\_and\_bound\_nqueen(arr, x + 1, n):

return True # Solution found

arr[x][col] = 0 # Backtrack

return False # No position is safe in this row

# Main function

def main():

print("Name: Prachi Karande")

print("Roll no: TACO22134")

n = int(input("Enter the number of Queens: "))

arr = [[0] \* n for \_ in range(n)]

if branch\_and\_bound\_nqueen(arr, 0, n):

for row in arr:

print(' '.join(str(cell) for cell in row))

else:

print("No solution found.")

# Run main function

if \_\_name\_\_ == '\_\_main\_\_':

main()

Backtracking:  
print("Name: Prachi Karande", flush=True)

print("Roll no: TACO22134", flush=True)

print() # Blank line for spacing

# Class for N-Queens using Backtracking

class BacktrackingNQueens:

def \_\_init\_\_(self, n):

self.n = n

self.board = [[0 for \_ in range(n)] for \_ in range(n)]

self.solutions = []

# Check if it's safe to place a queen at board[row][col]

def is\_safe(self, row, col):

for i in range(row):

# Check column

if self.board[i][col] == 1:

return False

# Check upper-left diagonal

if col - (row - i) >= 0 and self.board[i][col - (row - i)] == 1:

return False

# Check upper-right diagonal

if col + (row - i) < self.n and self.board[i][col + (row - i)] == 1:

return False

return True

# Recursive solver to find all solutions

def solve(self, row=0):

if row == self.n:

solution = []

for i in range(self.n):

row\_solution = ''

for j in range(self.n):

row\_solution += 'Q' if self.board[i][j] == 1 else '.'

solution.append(row\_solution)

self.solutions.append(solution)

return

for col in range(self.n):

if self.is\_safe(row, col):

self.board[row][col] = 1 # Place queen

self.solve(row + 1) # Recursive call

self.board[row][col] = 0 # Backtrack

# Print all solutions

def print\_solutions(self):

print(f"\nTotal solutions for N = {self.n}: {len(self.solutions)}\n")

for idx, solution in enumerate(self.solutions, 1):

print(f"Solution {idx}:")

for row in solution:

print(row)

print()

# Run the code

if \_\_name\_\_ == "\_\_main\_\_":

try:

n = int(input("Enter the size of the chessboard: "))

queens\_bt = BacktrackingNQueens(n)

queens\_bt.solve()

queens\_bt.print\_solutions()

except ValueError:

print("Please enter a valid integer.")