Objective: **Implementation and analysis of 4-queen problem**

4-queen problem

It is based on two algorithm :

1) NAÏVE ALGORITHM

2) BACKTRACKING ALGORITHM

Naïve Algorithm:

```
while there are untried configurations
{
    generate the next configuration
     if queens don't attack in this configuration then
    {
        print this configuration;
    }
}
```

Backtracking Algorithm:

1) Start in the leftmost column

2) If all queens are placed

return true

3) Try all rows in the current column.

Do following for every tried row.

a) If the queen can be placed safely in this row then mark this [row, column] as part of the solution and recursively check if placing queen here leads to a solution.

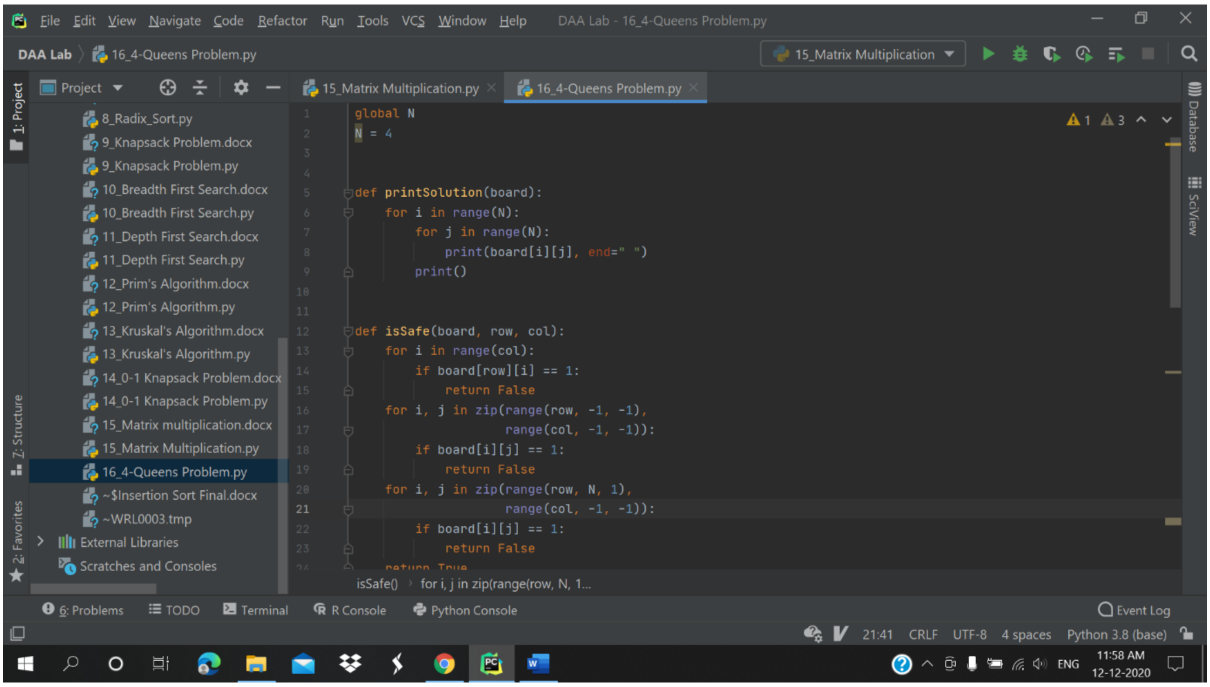
b) If placing the queen in [row, column] leads to a solution then return true.

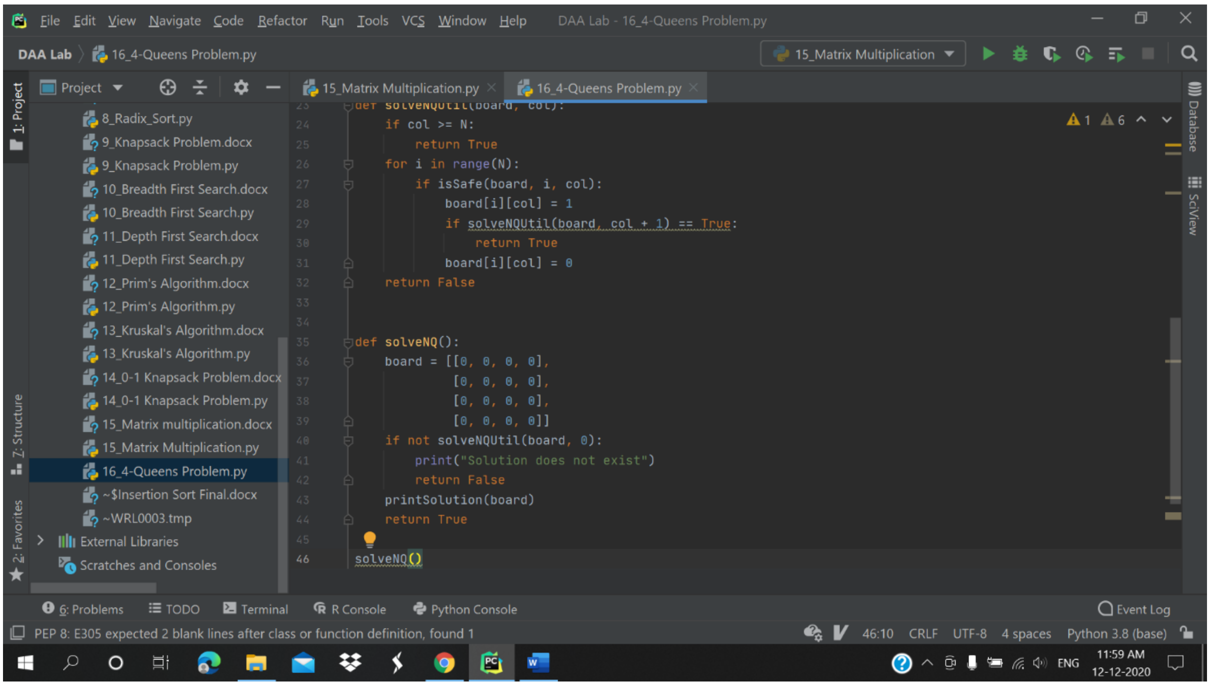
c) If placing queen doesn't lead to a solution then unmark this [row, column] (Backtrack) and go to step (a) to try other rows.

4) If all rows have been tried and nothing worked, return false to trigger backtracking.

Code:

```
global N
N = 4
def printSolution(board):
    for i in range(N):
        for j in range(N):
            print(board[i][j], end=" ")
        print()
def isSafe(board, row, col):
    for i in range(col):
        if board[row][i] == 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, N, 1),
                    range(col, -1, -1)):
        if board[i][j] == 1:
            return False
    return True
def solveNQUtil(board, col):
    if col >= N:
        return True
    for i in range(N):
        if isSafe(board, i, col):
            board[i][col] = 1
            if solveNQUtil(board, col + 1) == True:
                return True
            board[i][col] = 0
    return False
def solveNQ():
    board = [[0, 0, 0, 0],
             [0, 0, 0, 0],
             [0, 0, 0, 0],
             [0, 0, 0, 0]]
    if not solveNQUtil(board, 0):
        print("Solution does not exist")
        return False
    printSolution(board)
    return True
solveNQ()
```





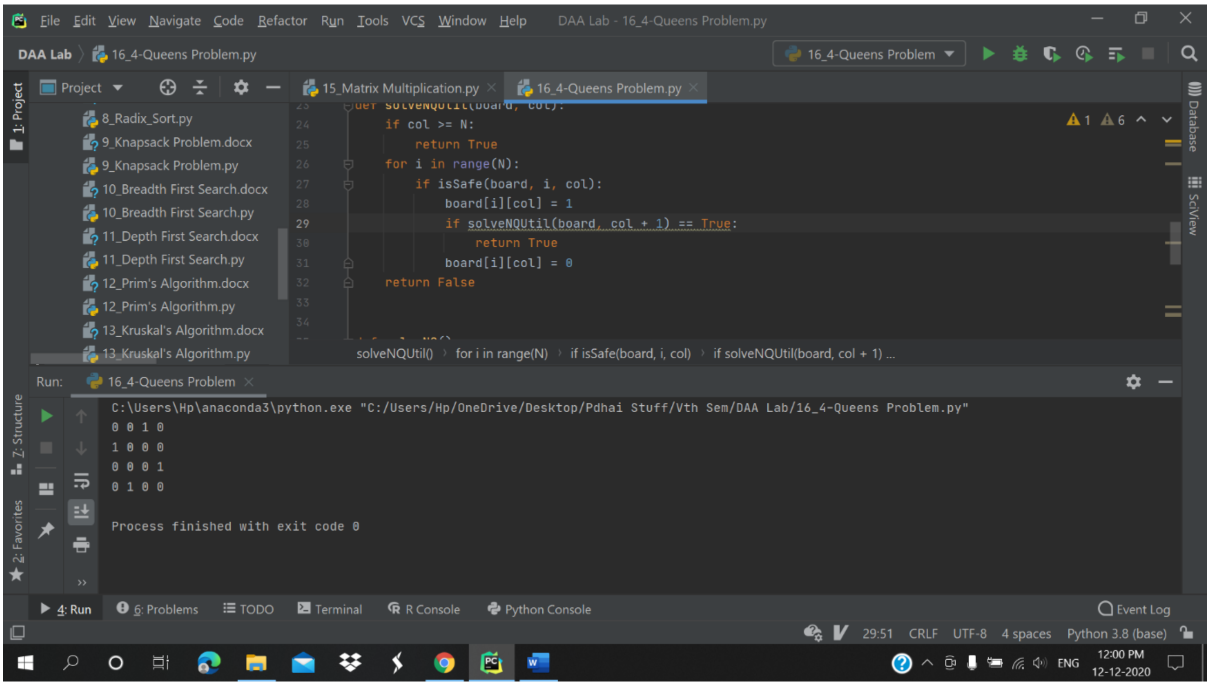
Output:

0 0 1 0

1 0 0 0

0 0 0 1

0 1 0 0



Time Complexity: O(2^n)