นายจิรภัทร แก้วส่งแสง รหัสนักศึกษา 63010139

รายงาน N – Queen

Source Code

Iterative

```
import time

class QueenChessBoard:
    def __init__(self, size):
        # board has dimensions size x size
        self.size = size
        # columns[r] is a number c if a queen is placed at row r and
column c.
        # columns[r] is out of range if no queen is place in row r.
        # Thus after all queens are placed, they will be at positions
        # (columns[0], 0), (columns[1], 1), ... (columns[size - 1],
size - 1)
        self.columns = []

    def place_in_next_row(self, column):
        self.columns.append(column)

    def remove_in_current_row(self):
        return self.columns.pop()

    def is_this_column_safe_in_next_row(self, column):
        # index of next row
        row = len(self.columns)

        # check column
        for queen_column in self.columns:
            if column == queen_column:
                return False

        # check diagonal
        for queen_row, queen_column in enumerate(self.columns):
            if queen_column - queen_row == column - row:
                return False

        # check other diagonal
```

```
        for queen_row, queen_column in enumerate(self.columns):
            if ((self.size - queen_column) - queen_row
                == (self.size - column) - row):
                return False

        return True

    def display(self):
        for row in range(self.size):
            for column in range(self.size):
                if column == self.columns[row]:
                    print('Q', end=' ')
                else:
                    print('-', end=' ')
            print()


def solve_queen(size):
    """Display a chessboard for each possible configuration of placing
n queens
    on an n x n chessboard and print the number of such
configurations."""
    board = QueenChessBoard(size)
    number_of_solutions = 0

    row = 0
    column = 0
    # iterate over rows of board
    while True:
        # place queen in next row
        while column < size:
            if board.is_this_column_safe_in_next_row(column):
                board.place_in_next_row(column)
                row += 1
                column = 0
                break
            else:
                column += 1

        # if could not find column to place in or if board is full
        if (column == size or row == size):
            # if board is full, we have a solution
            if row == size:
                board.display()
                print()
                number_of_solutions += 1
```

```
                # small optimization:
                # In a board that already has queens placed in all rows
except
                # the last, we know there can only be at most one
position in
                # the last row where a queen can be placed. In this
case, there
                # is a valid position in the last row. Thus we can
backtrack two
                # times to reach the second last row.
                board.remove_in_current_row()
                row -= 1

            # now backtrack
            try:
                prev_column = board.remove_in_current_row()
            except IndexError:
                # all queens removed
                # thus no more possible configurations
                break
            # try previous row again
            row -= 1
            # start checking at column = (1 + value of column in
previous row)
            column = 1 + prev_column

    print('Number of solutions :', number_of_solutions)

if __name__ == "__main__":
    start_time = time.time()

    n = int(input('Enter N Queen : '))
    solve_queen(n)

    print("Runtime :",time.time() - start_time,"seconds")
```

Recursive

```
import time

def isSafe(board, row, colum):

    #check straight line
    #print_Board(board,N)
    for i in range(N):
        if (board[row][i] == 'Q') or (board[i][colum] == 'Q'):
            return False
    #check diagonal
    for i in range(N):
        for j in range(N):
            if (i+j == row + colum ) or (i-j==row-colum):
                if (board[i][j] == 'Q'):
                    return False

    return True


def printSolution(board,N):
    for r in range(N):
        for c in range(N):
            print(board[r][c],end=" ")
        print()
    print()


def nQueen(board, N, r,solution):

    # if `N` queens are placed successfully, print the solution
    if r == N:
        printSolution(board,N)
        solution += 1
        return solution

    # place queen at every square in the current row `r`
    # and recur for each valid movement
    for i in range(N):

        # if no two queens threaten each other
        if isSafe(board, r, i):
            # place queen on the current square
            board[r][i] = 'Q'
```

```
            # recur for the next row
            solution = nQueen(board, N, r + 1,solution)

            # backtrack and remove the queen from the current square
            board[r][i] = '-'

    return solution

if __name__ == '__main__':

    start_time = time.time()
    # `N × N` chessboard
    N = int(input("Enter N Queen : "))

    # `board[][]` keeps track of the position of queens in
    # the current configuration
    board = [['-']*N for i in range(N)]

    solution = 0

    solution = nQueen(board,N, 0,solution)

    print("Number of  Solution :",solution)
    print("Runtime :",time.time() - start_time,"seconds")
```

รายละเอียดเครื่องคอมพิวเตอร์ CPU Memory

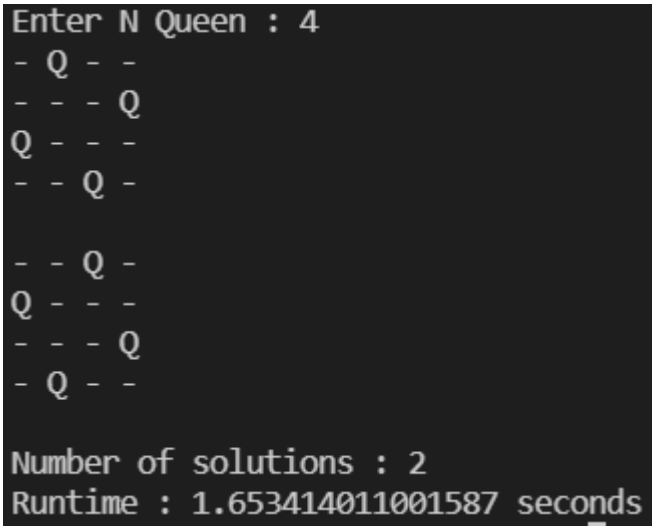
CPU : Intel(R) Core(TM) i7 – 6700HQ

RAM : 8 GB

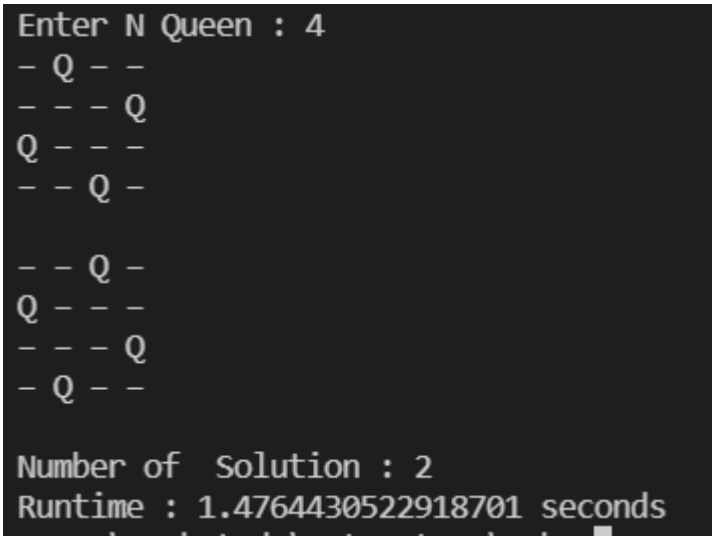
Capture การ Run และจับเวลาของแต่ละ Input

***Input เริ่มจาก 4 ถึง 20

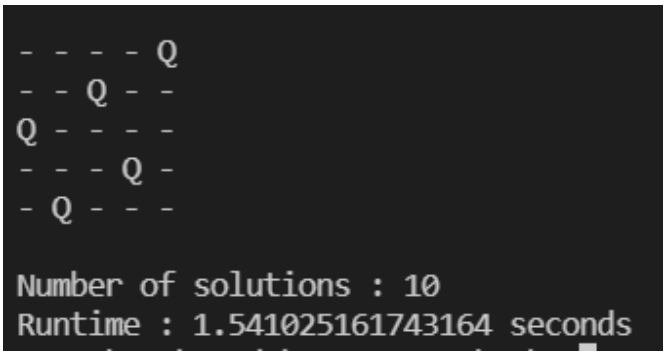
Input : 4 (Iterative)



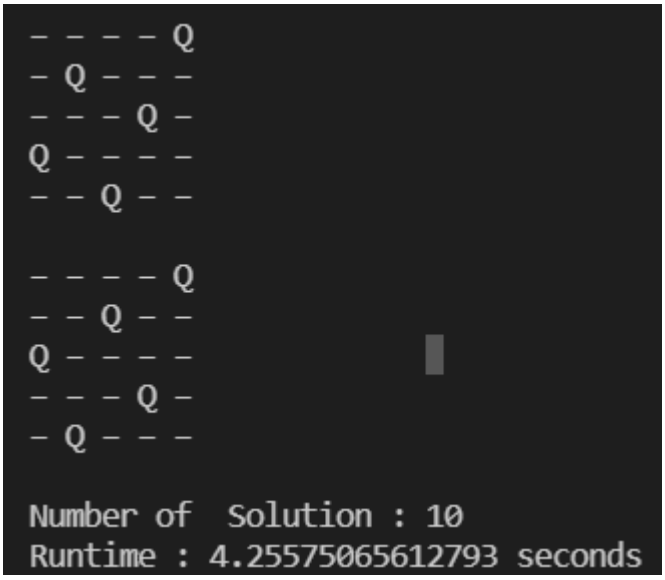
Input : 4 (Recursive)



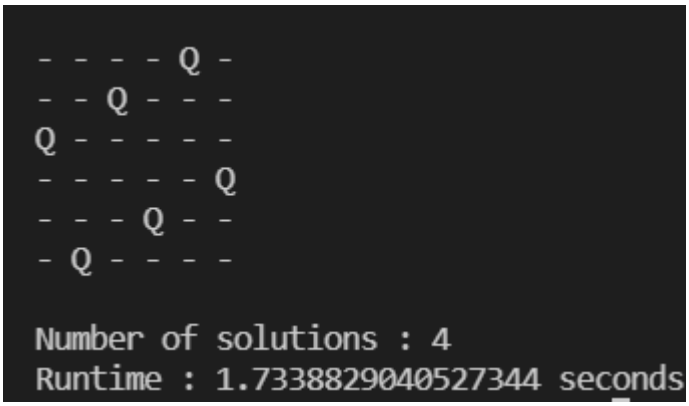
Input : 5 (Iterative)



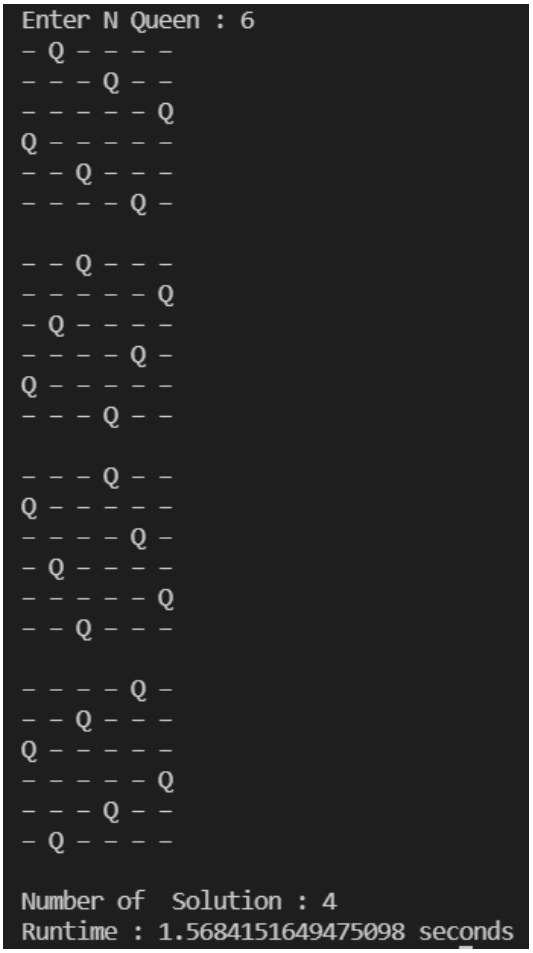
Input : 5 (Recursive)



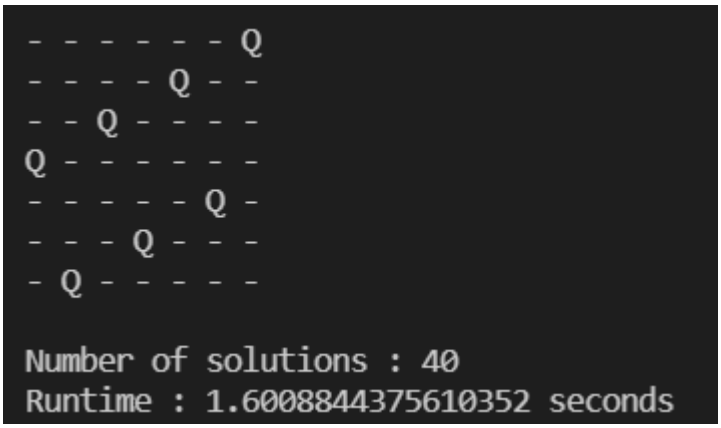
Input : 6 (Iterative)



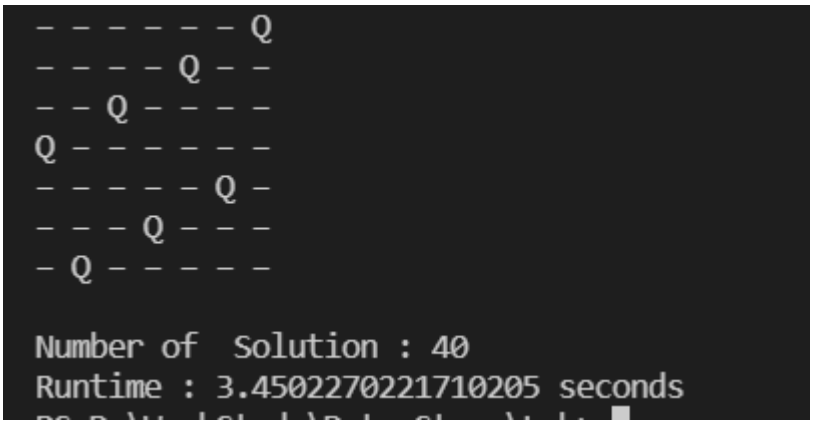
Input : 6 (Recursive)



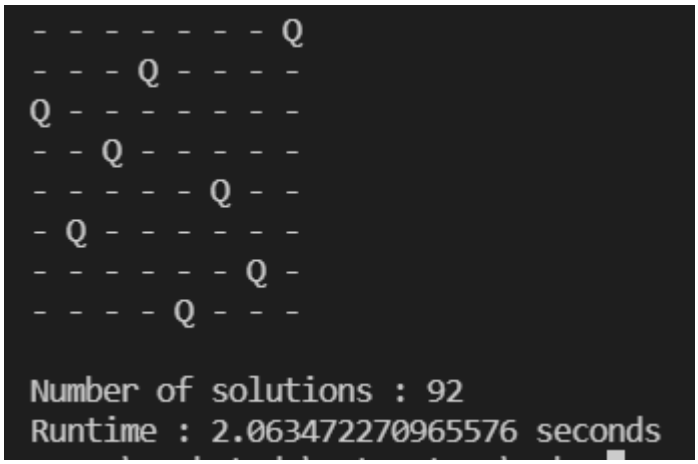
Input : 7 (Iterative)



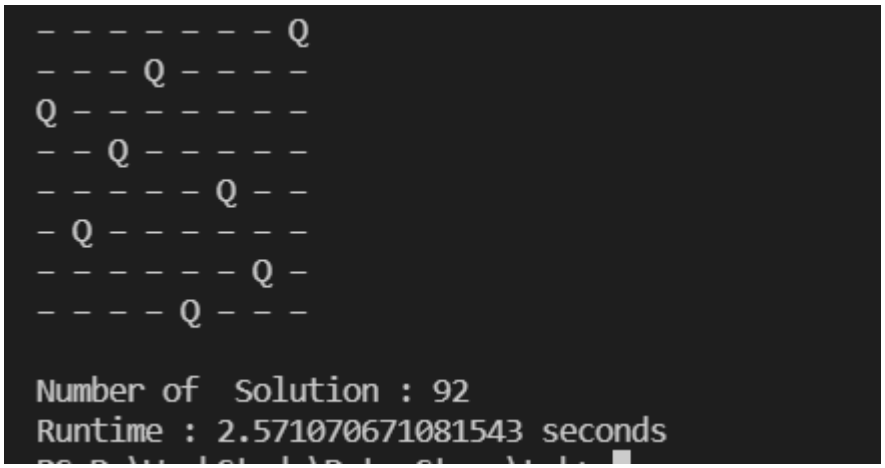
Input : 7 (Recursive)



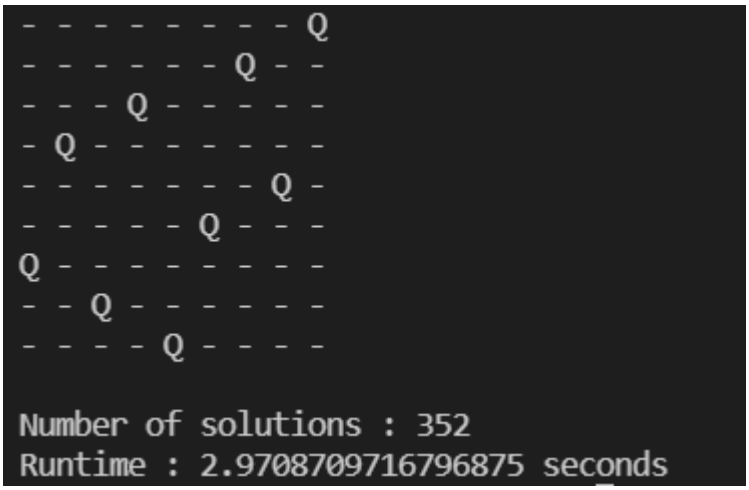
Input : 8 (Iterative)



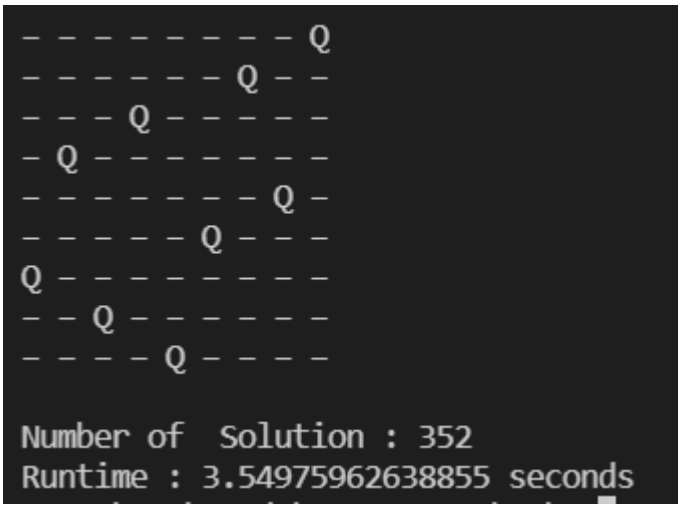
Input : 8 (Recursive)



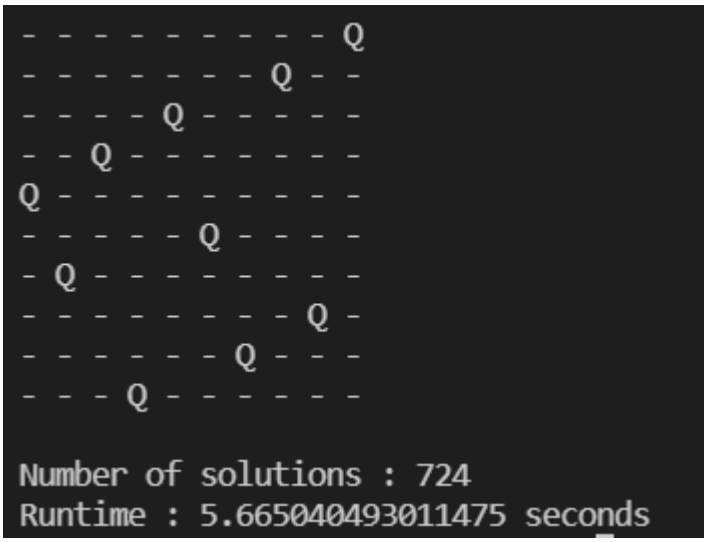
Input : 9 (Iterative)



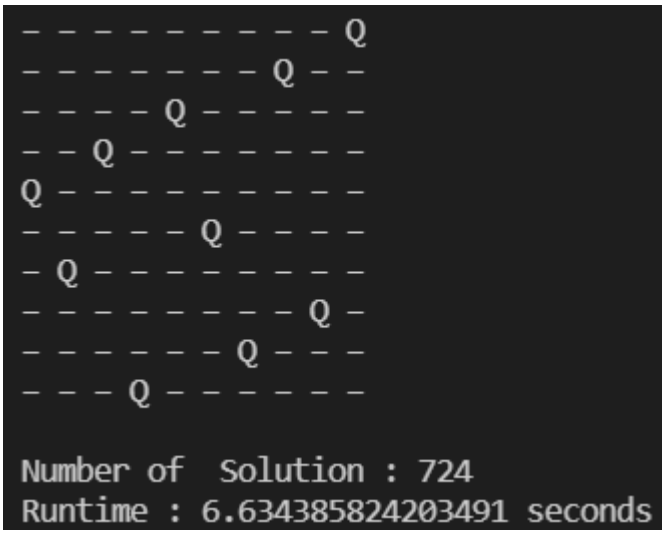
Input : 9 (Recursive)



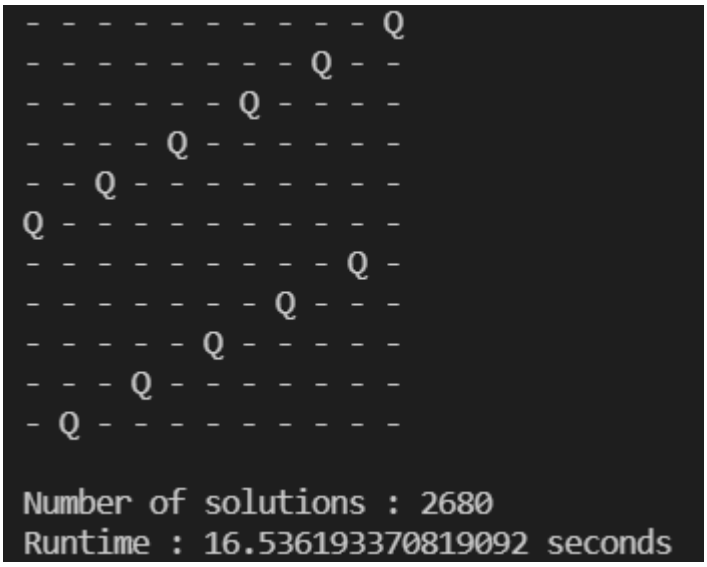
Input : 10 (Iterative)



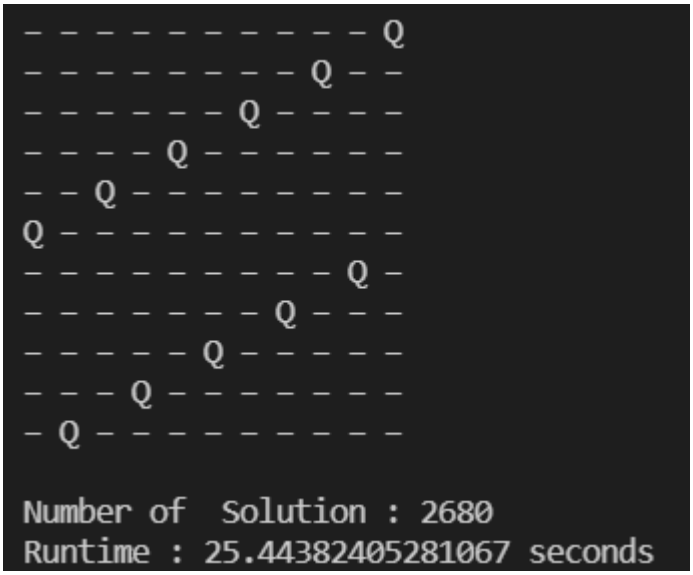
Input : 10 (Recursive)



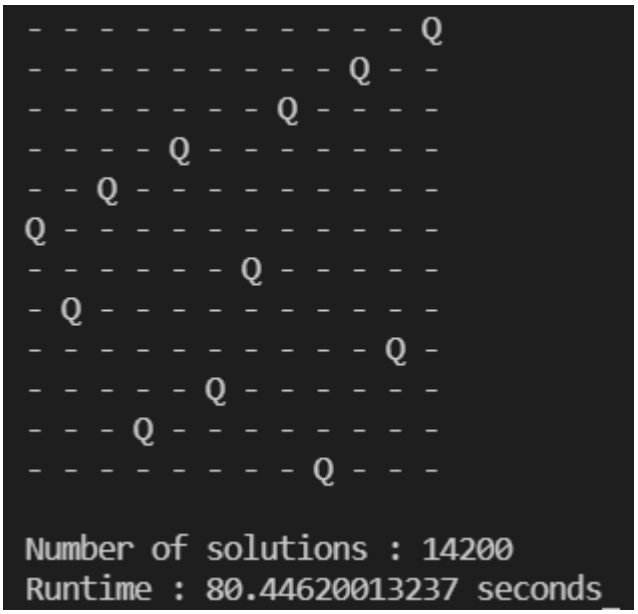
Input : 11 (Iterative)



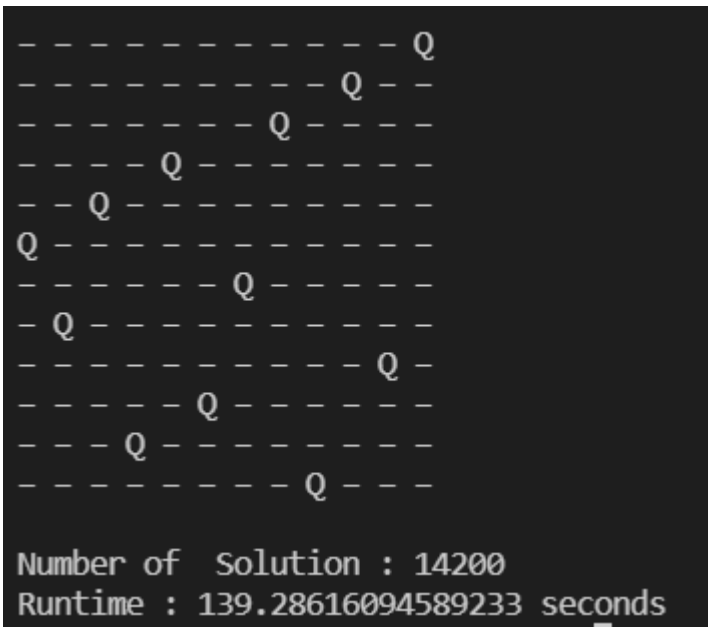
Input : 11 (Recursive)



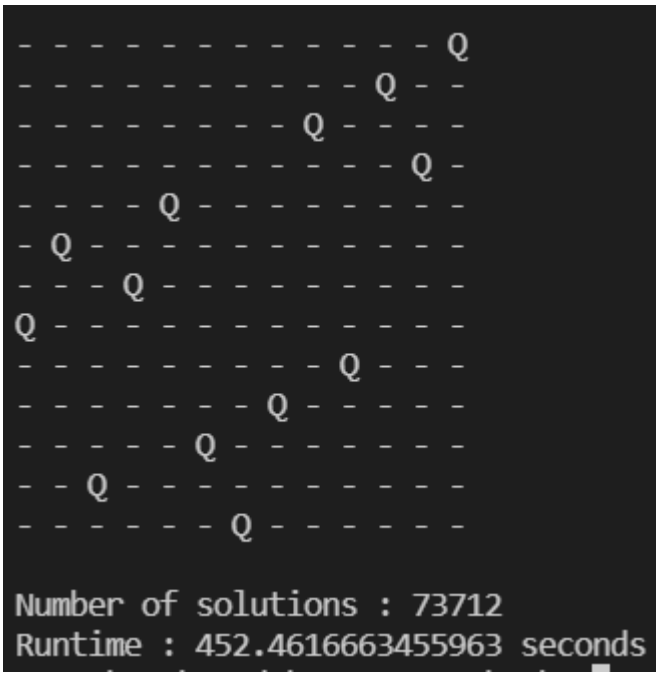
Input : 12 (Iterative)



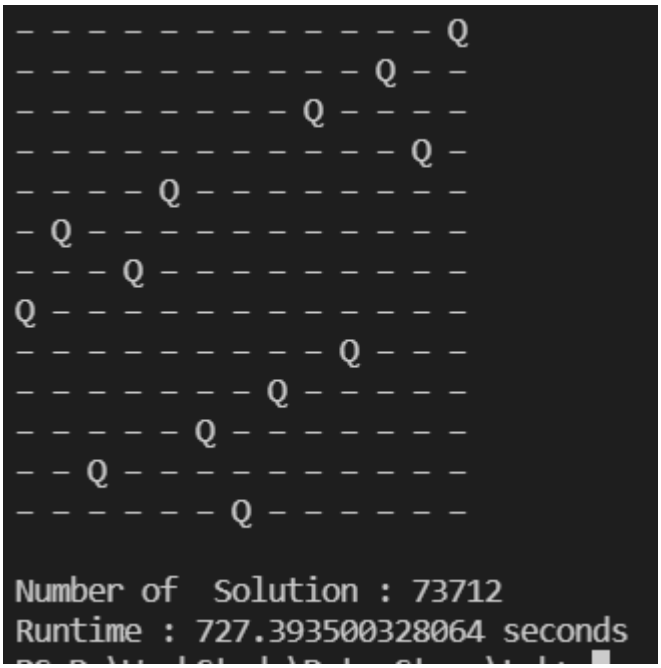
Input : 12 (Recursive)



Input : 13 (Iterative)



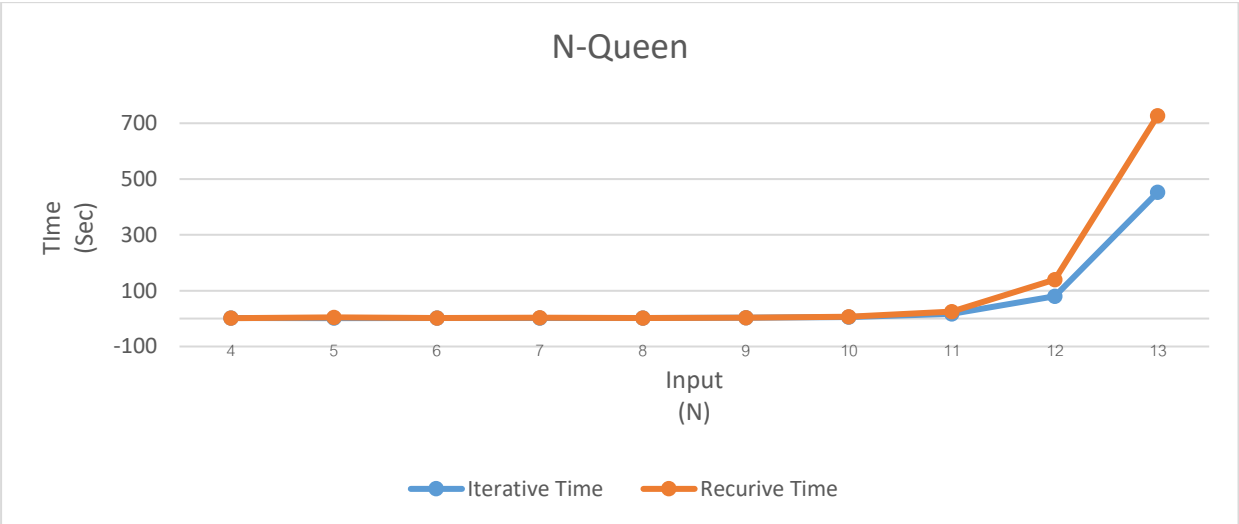
Input : 13 (Recursive)



ตารางบันทึกผล

Input (N)	Solution	Iterative Time (Sec)	Recursive Time (Sec)
4	2	1.653414	1.476443
5	10	1.541025	4.255751
6	4	1.733882	1.568415
7	40	1.600884	3.450227
8	92	2.063472	2.571071
9	352	2.970871	3.549759
10	724	5.66504	6.634385
11	2680	16.536193	25.443824
12	14200	80.4462	139.28616
13	73712	452.461666	727.3935

กราฟเปรียบเทียบเวลาในการ Run ของ Iterative และ Recursive



การวิเคราะห์ผลลัพธ์ที่ได้

เมื่อเทียบดูเวลาจากตารางจะเห็นได้ว่าในกรณีแรกๆ ที่มี solution น้อย นั้นจะมีกรณีที่ เวลาของ recursive algorithm น้อยกว่า iterative algorithm แต่เมื่อช่วงหลังกรณีที่ input มีค่ามากขึ้นทำให้ solution มากขึ้นนั้น เวลา runtime ของ recursive algorithm นั้นจะเริ่มมากกว่า iterative algorithm เรื่อยและ runtime จะยิ่งห่างมากขึ้นเรื่อยตาม input ออย่างเห็นได้ชัด สังเกตได้จากกราฟที่ช่วงแรก เวลาจะมีความใกล้เคียงกัน และตั้งแต่ช่วง input เท่ากับ 11 ขึ้นไปจะเห็นได้ว่ากราฟของ recursive algorithm มีความสูงมากกว่า กราฟของ iterative ขึ้นเรื่อย

สรุปได้ว่าในกรณีที่ input มีค่าน้อยช่วง 4 -10 สามารถใช้ iterative algorithm หรือ recursive algorithm ก็ได้ เพราะว่า runtime มีความแตกต่างกันไม่มากและบางค่า recursive algorithm ก็มี runtime ที่เร็วกว่า ส่วนในกรณีที่มี input ค่าสูงขึ้นมากเท่าไหร่ ในกรณีที่ต้องการ runtime ที่มีค่าน้อยควรใช้ iterative algorithm ซึ่งเร็วว่า recursive algorithm อย่างเห็นได้ชัด

บรรณานุกรม

GeeksforGeeks .// (03 Aug, 2021) .// Python Program for N Queen Problem | Backtracking-3 .// สืบค้นเมื่อ 9 ตุลาคม 2564

จาก https://www.geeksforgeeks.org/python-program-for-n-queen-problem-backtracking-3/

SANFOUNDRY.// Python Program to Solve n-Queen Problem without Recursion .// สืบค้นเมื่อ 9 ตุลาคม 2564

จาก https://www.sanfoundry.com/python-program-solve-n-queen-problem-without-recursion/

Grepper by Handsome Hedgehog .// (May 09 2020) .// how to count running time in python .// สืบค้นเมื่อ 9 ตุลาคม 2564

จาก https://www.codegrepper.com/code-examples/python/how+to+count+running+time+in+python

Google Colab by kiatnarong tongprasert .// (5 Oct,2021) .// N – Queen Problem .// สืบค้นเมื่อ 9 ตุลาคม 2564

จาก https://colab.research.google.com/drive/1nhVvTij1LuF-nB1okf9MHtyTdpmARzdG#scrollTo=j2pKQ01B7ICG