**LAB 4:**

**Program:** Implement Annealing Technique to Solve N-Queen’s Problem

**Code:**

import numpy as np

import random

import math

def create\_initial\_state(n):

return np.random.permutation(n)

def fitness(state):

attacks = 0

n = len(state)

for i in range(n):

for j in range(i + 1, n):

if state[i] == state[j] or abs(state[i] - state[j]) == abs(i - j):

attacks += 1

return (n \* (n - 1)) // 2 - attacks

def get\_neighbors(state):

neighbors = []

n = len(state)

for i in range(n):

for j in range(i + 1, n):

new\_state = state.copy()

new\_state[i], new\_state[j] = new\_state[j], new\_state[i]

neighbors.append(new\_state)

return neighbors

def probability(delta\_fitness, temperature):

if delta\_fitness > 0:

return 1.0

return math.exp(delta\_fitness / temperature)

def print\_grid(state):

n = len(state)

grid = np.full((n, n), '.')

for i in range(n):

grid[i, state[i]] = 'Q'

print("\n".join(" ".join(row) for row in grid))

print("\n")

def simulated\_annealing(n, initial\_temp, cooling\_rate, max\_iters):

current\_state = create\_initial\_state(n)

current\_fitness = fitness(current\_state)

best\_state = current\_state.copy()

best\_fitness = current\_fitness

temperature = initial\_temp

state\_space\_steps = [current\_state.copy()]

for iteration in range(max\_iters):

neighbors = get\_neighbors(current\_state)

new\_state = random.choice(neighbors)

new\_fitness = fitness(new\_state)

delta\_fitness = new\_fitness - current\_fitness

if delta\_fitness > 0 or random.uniform(0, 1) < probability(delta\_fitness, temperature):

current\_state = new\_state

current\_fitness = new\_fitness

if current\_fitness > best\_fitness:

best\_state = current\_state.copy()

best\_fitness = current\_fitness

state\_space\_steps.append(current\_state.copy())

temperature \*= cooling\_rate

if best\_fitness == (n \* (n - 1)) // 2:

break

return best\_state, best\_fitness, state\_space\_steps

N = 8

initial\_temperature = 10

cooling\_rate = 0.95

max\_iterations = 100

best\_position, best\_objective, steps = simulated\_annealing(N, initial\_temperature, cooling\_rate, max\_iterations)

print('The best position found is:', best\_position)

print('The number of non-attacking pairs of queens is:', best\_objective)

print("Grid representation of the best position:")

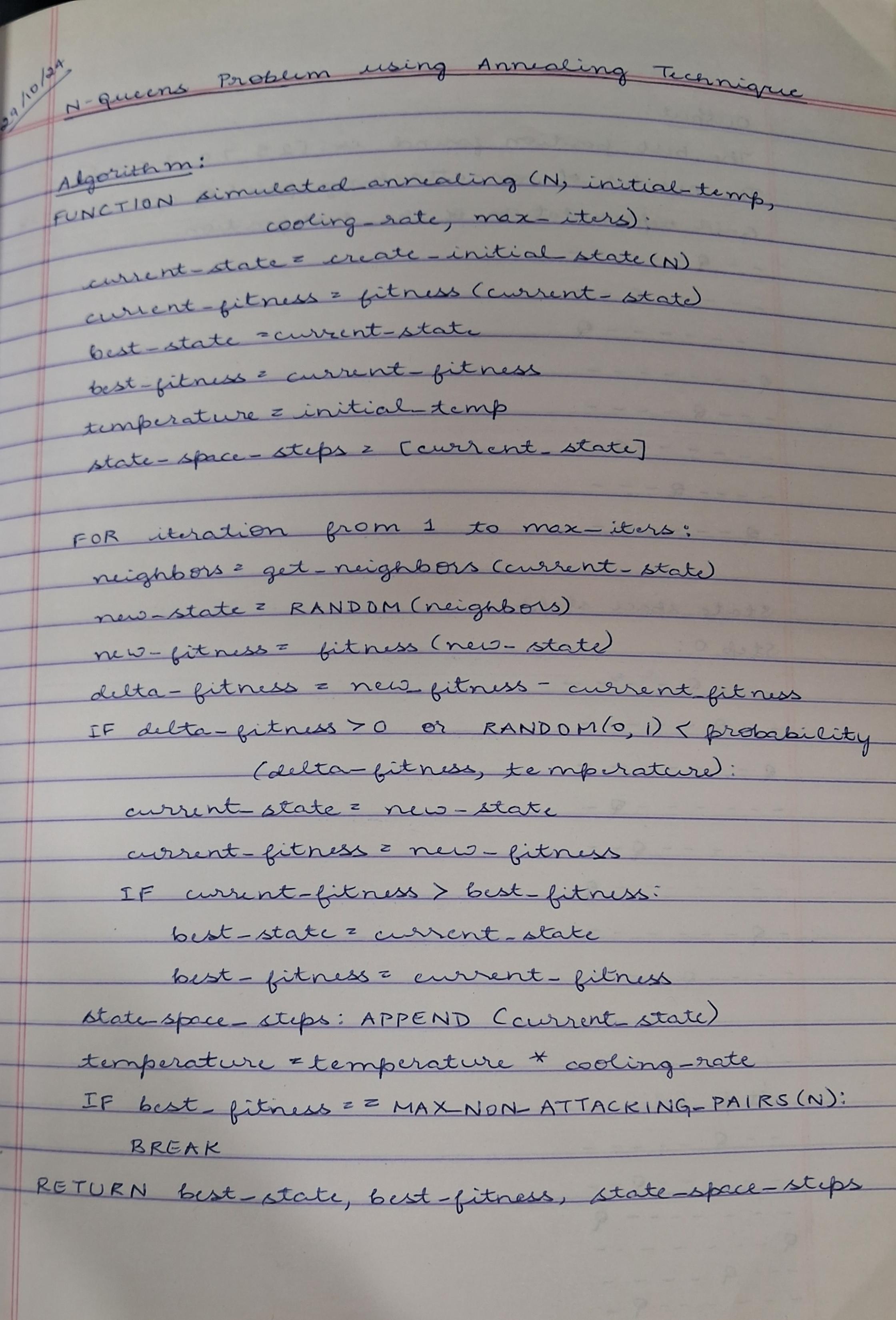
print\_grid(best\_position)

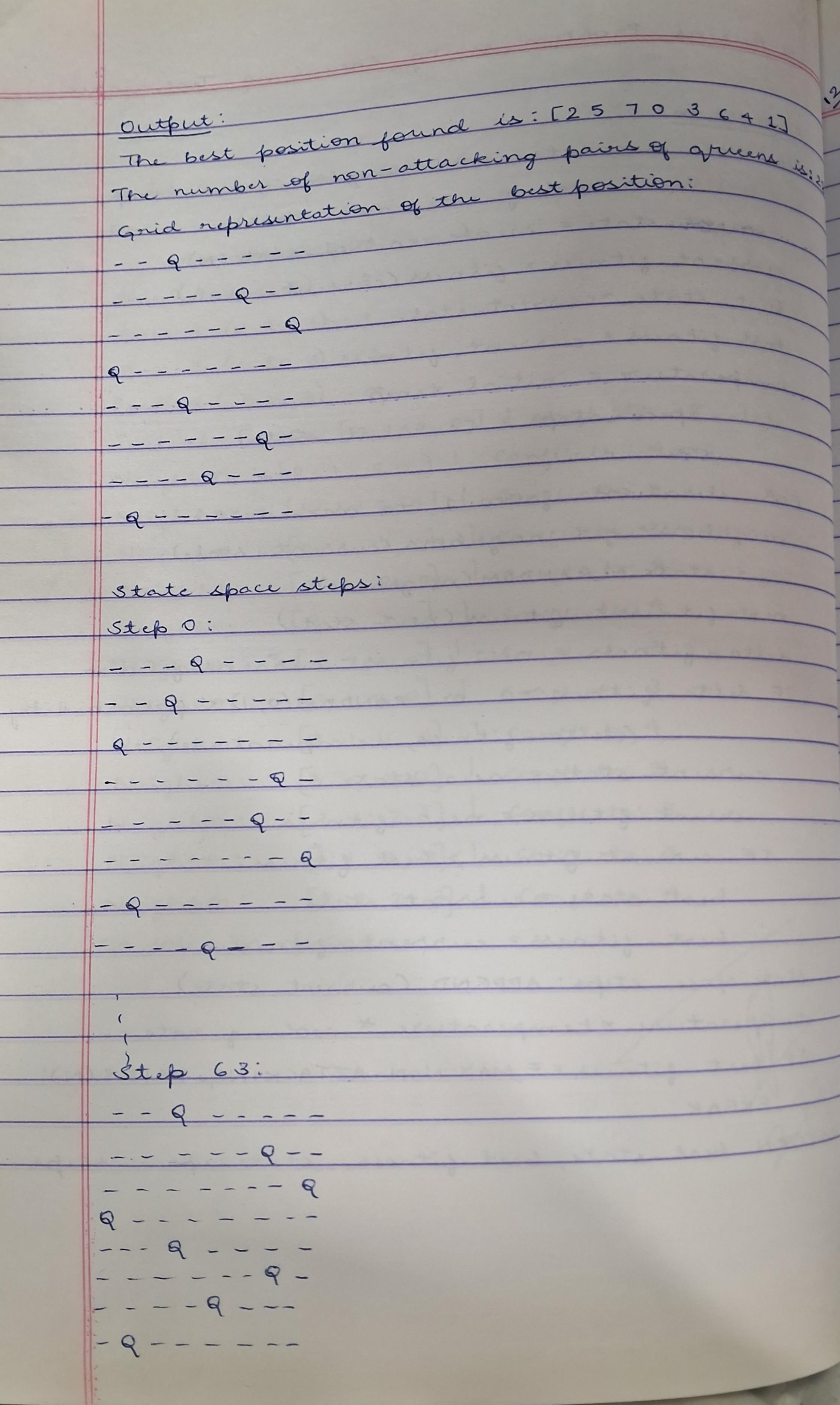
print("State space steps:")

for idx, step in enumerate(steps):

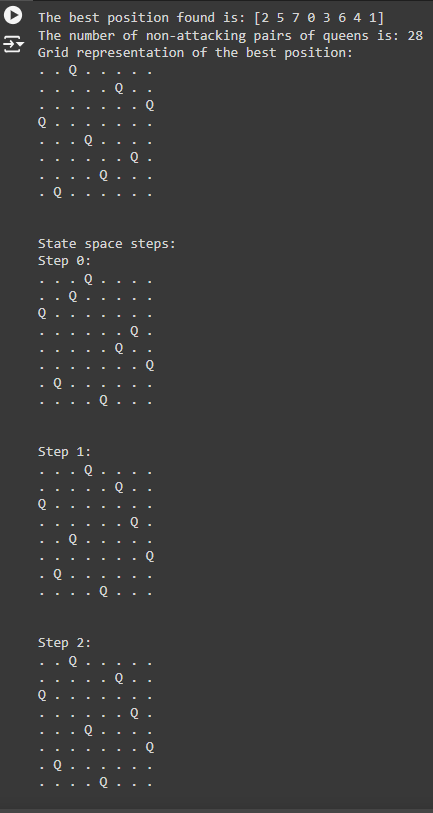
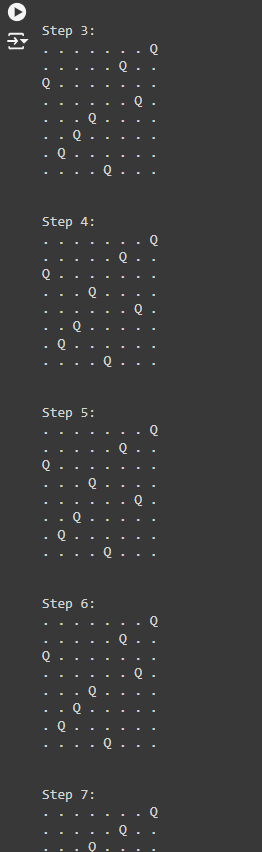
print(f"Step {idx}:")

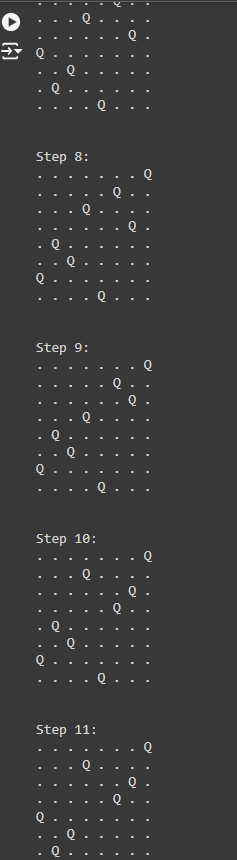
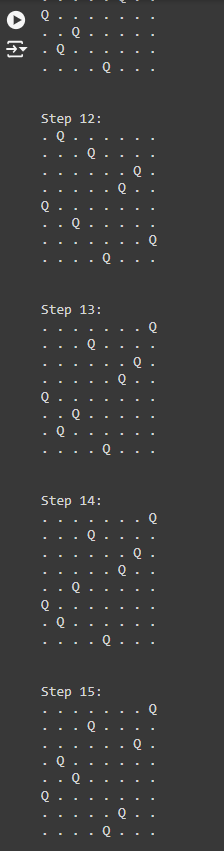
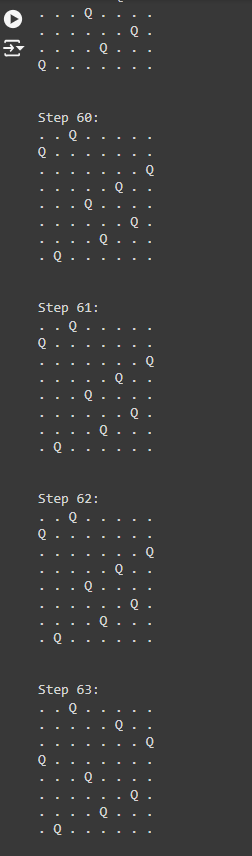
print\_grid(step)

**Algorithm:**

****

**Output:**

**** ****

**  **