LAB 5:

Program: Implement Annealing Technique to Solve N-Queens 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
```

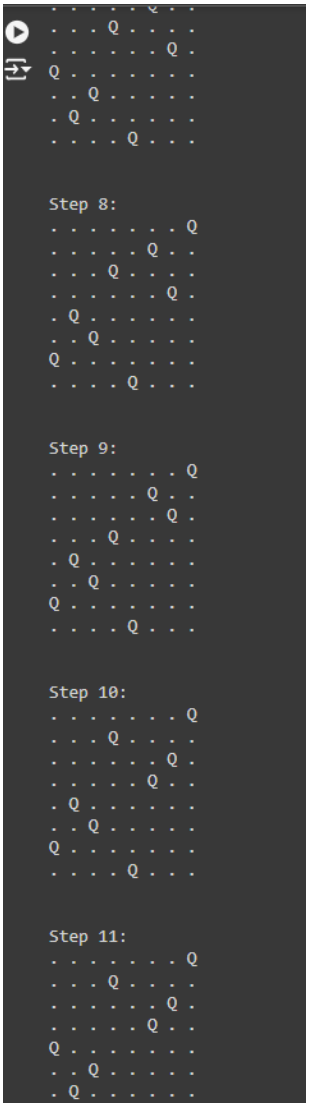
```
max_iterations = 100

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)
```

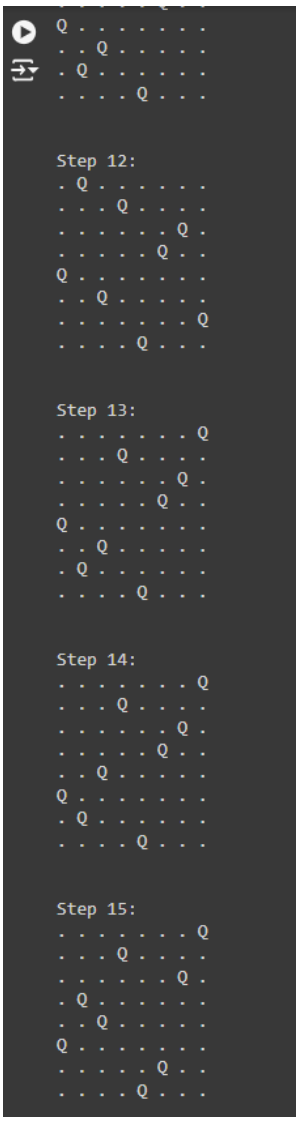
Algorithm:

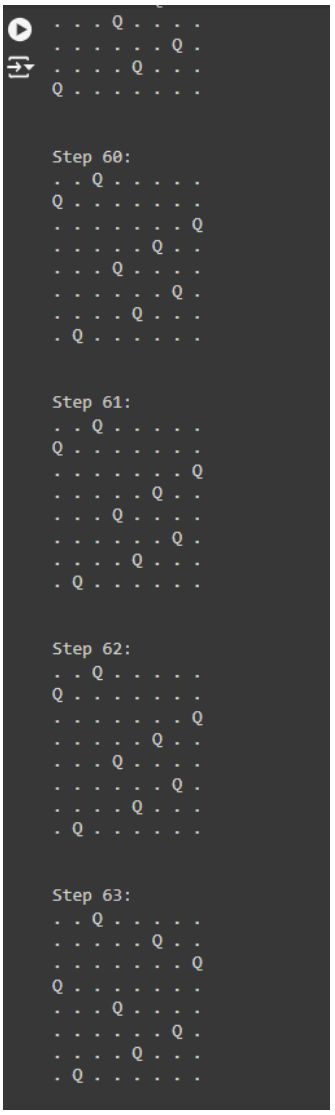
Output:

```
The best position found is: [2 5 7 0 3 6 4 1]
The number of non-attacking pairs of queens is: 28
Grid representation of the best position:
. . Q . . . . .
. . . . . Q . .
. . . . . . . Q
Q . . . . . . .
. . . Q . . . .
. . . . . . Q .
. . . . Q . . .
. Q . . . . . .

State space steps:
Step 0:
. . . Q . . . .
. . Q . . . . .
Q . . . . . . .
. . . . . . Q .
. . . . . Q . .
. . . . . . . Q
. Q . . . . . .
. . . . Q . . .

Step 1:
. . . Q . . . .
. . . . . Q . .
Q . . . . . . .
. . . . . . Q .
. . Q . . . . .
. . . . . . . Q
. Q . . . . . .
. . . . Q . . .

Step 2:
. . Q . . . . .
. . . . . Q . .
Q . . . . . . .
. . . . . . Q .
. . . Q . . . .
. . . . . . . Q
. Q . . . . . .
. . . . Q . . .
```

```
Step 3:
. . . . . . . Q
. . . . . Q . .
Q . . . . . . .
. . . . . . Q .
. . . Q . . . .
. . Q . . . . .
. Q . . . . . .
. . . . Q . . .

Step 4:
. . . . . . . Q
. . . . . Q . .
Q . . . . . . .
. . . Q . . . .
. . . . . . Q .
. . Q . . . . .
. Q . . . . . .
. . . . Q . . .

Step 5:
. . . . . . . Q
. . . . . Q . .
Q . . . . . . .
. . . Q . . . .
. . . . . . Q .
. . Q . . . . .
. Q . . . . . .
. . . . Q . . .

Step 6:
. . . . . . . Q
. . . . . Q . .
Q . . . . . . .
. . . . . Q . .
. . . Q . . . .
. . Q . . . . .
. Q . . . . . .
. . . . Q . . .

Step 7:
. . . . . . . Q
. . . . . Q . .
. . . Q . . . .
```


. . . . . Q . .
. . . Q . . . .
. . . . . Q .
Q . . . . . .
. . Q . . . .
. Q . . . . .
. . . . Q . . .
Step 8:
. . . . Q
. . . . . Q . .
. . . Q . . . .
. . . . . Q .
. Q . . . . . .
. . Q . . . . .
Q . . . . . .
. . . . Q . . .
Step 9:
. . . . Q
. . . . . Q . .
. . . . . Q .
. . . Q . . . .
. Q . . . . . .
. . Q . . . . .
Q . . . . . .
. . . . Q . . .
Step 10:
. . . . Q
. . . Q . . . .
. . . . . Q .
. . . . . Q . .
. Q . . . . . .
. . Q . . . . .
Q . . . . . .
. . . . Q . . .
Step 11:
. . . . Q
. . . Q . . . .
. . . . . Q .
. . . . . Q . .
Q . . . . . .
. . Q . . . . .
. Q . . . . . .


Q . . . . . .
. . Q . . . . .
. Q . . . . . .
. . . . Q . . .
Step 12:
. Q . . . . . .
. . . Q . . . .
. . . . . Q .
. . . . . Q . .
Q . . . . . .
. . Q . . . . .
. . . . Q
. . . . Q . . .
Step 13:
. . . . Q
. . . Q . . . .
. . . . . Q .
. . . . . Q . .
Q . . . . . .
. . Q . . . . .
. Q . . . . . .
. . . . Q . . .
Step 14:
. . . . Q
. . . Q . . . .
. . . . . Q .
. . . . . Q . .
. . Q . . . . .
Q . . . . . .
. Q . . . . . .
. . . . Q . . .
Step 15:
. . . . Q
. . . Q . . . .
. . . . . Q .
. Q . . . . . .
. . Q . . . . .
Q . . . . . .
. . . . . Q . .
. . . . Q . . .


. . . Q . . . .
. . . . . Q .
. . . . Q . . .
Q . . . . . .
Step 60:
. . Q . . . . .
Q . . . . . .
. . . . Q
. . . . . Q . .
. . . Q . . . .
. . . . . Q .
. . . . Q . . .
. Q . . . . . .
Step 61:
. . Q . . . . .
Q . . . . . .
. . . . Q
. . . . . Q . .
. . . Q . . . .
. . . . . Q .
. . . . Q . . .
. Q . . . . . .
Step 62:
. . Q . . . . .
Q . . . . . .
. . . . Q
. . . . . Q . .
. . . Q . . . .
. . . . . Q .
. . . . Q . . .
. Q . . . . . .
Step 63:
. . Q . . . . .
. . . . . Q . .
. . . . Q
Q . . . . . .
. . . Q . . . .
. . . . . Q .
. . . . Q . . .
. Q . . . . . .

29/10/24 Tuesday

Bafna Gold
Date: Page: 23

Lab -5

Analyse and Implement N Queens problem using Simulated Annealing technique

Algorithm:

```
FUNCTION Simulated_annealing (N, initial_temp, cooling_rate, max_iters):
Current_State    = create_initial_State (N)
Current_fitness  = fitness (current_State)
best_State       = current_State
best_fitness     = current_fitness.
temperature      = initial_temp
State_Space_Steps = [current_State]

FOR iteration FROM I TO MAX_iters:
neighbours = get_neighbors (current_State)
new_State  = RANDOM (neighbors)
new_fitness = fitness (new_State)
delta_fitness = new_fitness - current_fitness
IF delta_fitness >0 or RANDOM(0,1) < probability
    (delta_fitness, temperature):
  Current_State = new_State
  Current_fitness = new_fitness:
    best_state = current_State
    best_fitness = current_fitness
State_Space_Steps: APPEND (current_State)
temperature = temperature * cooling_rate
IF best_fitness == MAX_NON_ATTACKING_
                   PAIRS (N):
  BREAK
RETURN best_State, best_fitness, State_space_S
```

output:
The best position found is: [2 5 7 0 3 6 4 1]
The number of non-attacking pairs of queens is: 28
Grid representation of the best position:
_ _ Q _ _ _ _ _
_ _ _ _ _ Q _ _
_ _ _ _ _ _ _ Q
Q _ _ _ _ _ _ _
_ _ _ Q _ _ _ _
_ _ _ _ _ _ Q _
_ _ _ _ Q _ _ _
_ Q _ _ _ _ _ _

State Space Steps:
Step 0:
_ _ _ Q _ _ _ _
_ _ Q _ _ _ _ _
Q _ _ _ _ _ _ _
_ _ _ _ _ _ Q _
_ _ _ _ _ Q _ _
_ _ _ _ _ _ _ Q
_ Q _ _ _ _ _ _
_ _ _ Q _ _ _ _
;
Step 63:
_ _ Q _ _ _ _ _
_ _ _ _ _ Q _ _
_ _ _ _ _ _ _ Q
Q _ _ _ _ _ _ _
_ _ _ Q _ _ _ _
_ _ _ _ _ _ Q _
_ _ _ _ Q _ _ _
_ Q _ _ _ _ _ _