STIMULATED ANNEALING

CODE:-

```
print(f"SHREYASGOWDA C (1BM23CS319)")
import random
import math

def print_board(state):
    n = len(state)
    for row in range(n):
        line = ""
        for col in range(n):
            if state[col] == row:
                line += "Q "
            else:
                line += ". "
        print(line)
    print()

def calculate_cost(state):
    cost = 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]) == j - i:
                cost += 1
    return cost

def get_neighbor(state):

    n = len(state)
    neighbor = list(state)
    i, j = random.sample(range(n), 2)
    neighbor[i], neighbor[j] = neighbor[j], neighbor[i]
    return tuple(neighbor), (i, j)

def simulated_annealing(initial_state, initial_temp=1000, cooling_rate=0.95, min_temp=1e-3,
max_iter=1000):
    current_state = initial_state
    current_cost = calculate_cost(current_state)
    temperature = initial_temp
    path = [(current_state, current_cost, None)]

    print("Initial State:")
    print_board(current_state)
    print(f"Cost: {current_cost}\n")
```

```
    iteration = 0
    while temperature > min_temp and current_cost > 0 and iteration < max_iter:
        neighbor, swap = get_neighbor(current_state)
        neighbor_cost = calculate_cost(neighbor)

        cost_diff = neighbor_cost - current_cost


        if cost_diff < 0 or math.exp(-cost_diff / temperature) > random.random():
            current_state, current_cost = neighbor, neighbor_cost
            path.append((current_state, current_cost, swap))
            print(f"Iteration {iteration}: Swap columns {swap}")
            print_board(current_state)
            print(f"Cost: {current_cost}, Temperature: {temperature:.4f}\n")

        temperature *= cooling_rate
        iteration += 1

    print("Terminated.")
    return path

def get_initial_state():
    print("Enter the initial positions of the 4 queens (row for each column, 0-indexed):")
    positions = []
    for col in range(4):
        while True:
            try:
                pos = int(input(f"Column {col}: "))
                if 0 <= pos < 4:
                    positions.append(pos)
                    break
                else:
                    print("Invalid input. Enter a number between 0 and 3.")
            except ValueError:
                print("Invalid input. Please enter an integer.")
    return tuple(positions)


initial_state = get_initial_state()
solution_path = simulated_annealing(initial_state)

print("Final path:")
for i, (state, cost, swap) in enumerate(solution_path):
```

```
print(f"Step {i}:")
print_board(state)
print(f"Cost: {cost}")
if swap is not None:
print(f"Swap columns: {swap}")
print("-------------------")
```

OUTPUT:-

STIMULATED ANNEALING