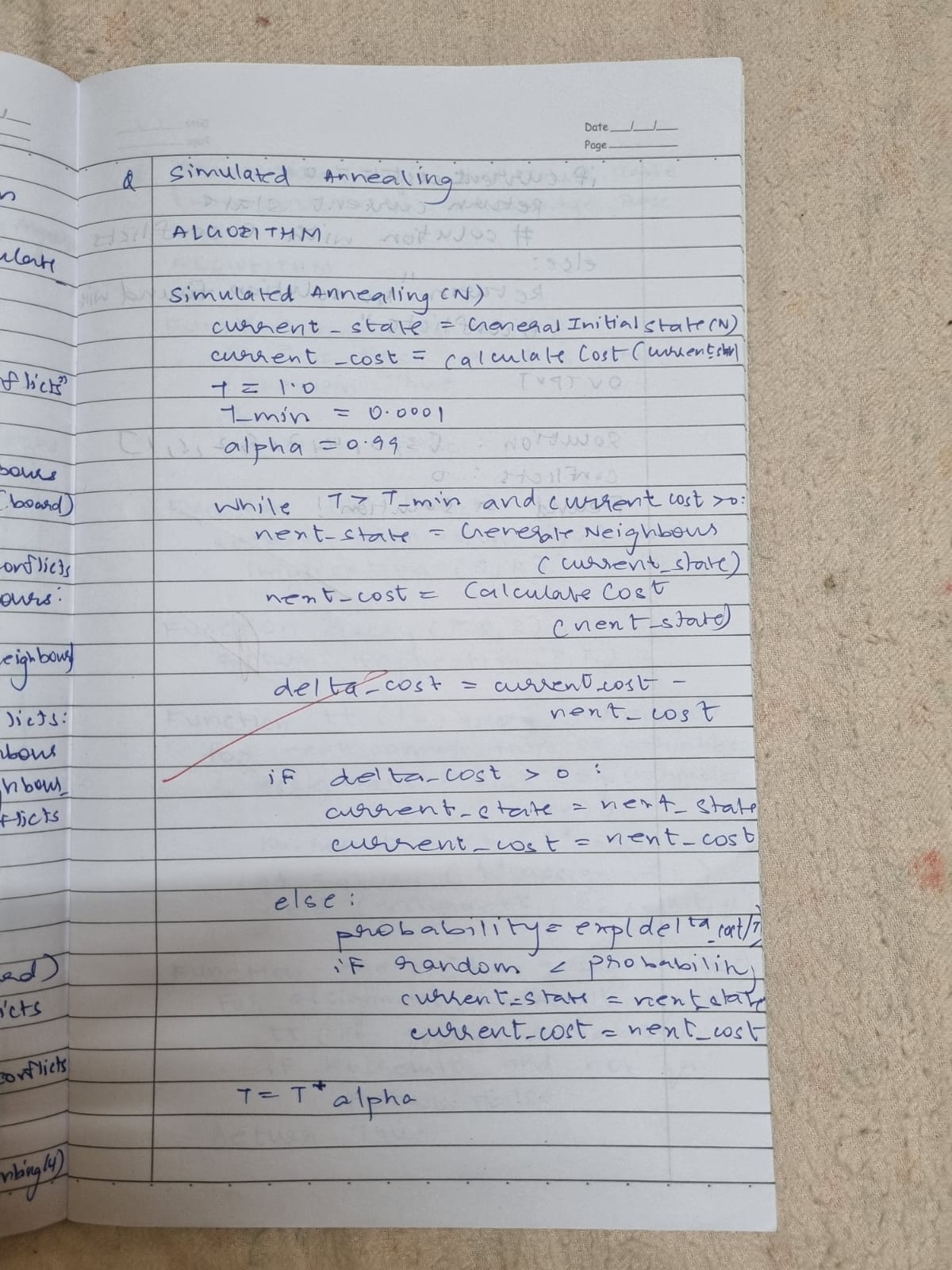
**N-Queens using Hill Climbing**

**Algorithm:**

****

**Code:**

import random  
import math  
  
# Configuration and parameters  
N = 8  # Number of queens and size of the board  
  
# Define a function to generate an initial random state  
def generate\_initial\_state():  
    # Place one queen in each column at a random row  
    return [random.randint(0, N-1) for \_ in range(N)]  
  
# Define a function to calculate the number of conflicts  
def calculate\_cost(state):  
    conflicts = 0  
    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):  
                conflicts += 1  
    return conflicts  
  
# Define a function to generate a neighboring state  
def generate\_neighbor(state):  
    neighbor = state[:]  
    col = random.randint(0, N - 1)  
    new\_row = random.randint(0, N - 1)  
    while new\_row == neighbor[col]:  
        new\_row = random.randint(0, N - 1)  
    neighbor[col] = new\_row  
    return neighbor  
  
# Simulated Annealing algorithm  
def simulated\_annealing():  
    current\_state = generate\_initial\_state()  
    current\_cost = calculate\_cost(current\_state)  
    T = 1.0  # Initial temperature  
    T\_min = 0.0001  # Minimum temperature threshold  
    alpha = 0.99  # Cooling rate  
  
    while T > T\_min and current\_cost > 0:  
        next\_state = generate\_neighbor(current\_state)  
        next\_cost = calculate\_cost(next\_state)  
        delta\_cost = current\_cost - next\_cost  
  
        if delta\_cost > 0:  
            # Move to the next state if it has fewer conflicts  
            current\_state, current\_cost = next\_state, next\_cost  
        else:  
            # Move to the next state with a probability that decreases with T  
            probability = math.exp(delta\_cost / T)  
            if random.random() < probability:  
                current\_state, current\_cost = next\_state, next\_cost  
  
        # Decrease temperature  
        T \*= alpha  
  
    return current\_state, current\_cost  
  
# Run Simulated Annealing and print result  
solution, conflicts = simulated\_annealing()  
print("Solution:", solution)  
print("Conflicts:", conflicts)  
  
if conflicts == 0:  
    print("Found a solution!")  
else:  
    print("Failed to find a solution with 0 conflicts.")

**Output:**

