HILL CLIMBING:

lab4.py

```python
import random

def calculate_cost(board):
    n = len(board)
    cost = 0
    for i in range(n):
        for j in range(i + 1, n):
            # Check if queens are in the same row or diagonal
            if board[i] == board[j] or abs(board[i] - board[j]) == j - i:
                cost += 1
    return cost

def get_neighbors(board):
    neighbors = []
    n = len(board)
    for col in range(n):
        for row in range(n):
            if row != board[col]:
                neighbor = board[:]
                neighbor[col] = row
                neighbors.append(neighbor)
    return neighbors

def hill_climbing(n):
    current_state = [random.randint(0, n - 1) for _ in range(n)]
    current_cost = calculate_cost(current_state)

    while True:
        neighbors = get_neighbors(current_state)
        next_state = min(neighbors, key=calculate_cost)
        next_cost = calculate_cost(next_state)

        if next_cost >= current_cost:
            return current_state
        current_state = next_state
        current_cost = next_cost

```

```python
if __name__ == "__main__":
    n = int(input("Enter the number of queens: "))
    solution = hill_climbing(n)
    cost = calculate_cost(solution)

    if cost == 0:
        print("Solution found:", solution)
    else:
        print("Local maximum reached. Best solution:", solution, "with cost:", cost)

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Local maximum reached. Best solution: [0, 2, 3, 1] with cost: 1
PS C:\Users\BMSCECSE\Desktop\1BF24CS047>
```

SIMULATED ANNELING:

lab4.1.py > ...

```python
import random
import math

def calculate_cost(board):
    n = len(board)
    cost = 0
    for i in range(n):
        for j in range(i + 1, n):
            if board[i] == board[j]:
                cost += 1
            if abs(board[i] - board[j]) == j - i:
                cost += 1
    return cost

def get_random_neighbor(board):
    n = len(board)
    neighbor = board[:]
    row = random.randint(0, n - 1)
    new_row = random.randint(0, n - 1)
    while new_row == neighbor[row]:
        new_row = random.randint(0, n - 1)
    neighbor[row] = new_row
    return neighbor

def simulated_annealing(n, T_initial, cooling_rate, max_iterations):
    current_state = [random.randint(0, n - 1) for _ in range(n)]
    current_cost = calculate_cost(current_state)
    T = T_initial
    best_state = current_state
    best_cost = current_cost

    for iteration in range(max_iterations):
        next_state = get_random_neighbor(current_state)
        next_cost = calculate_cost(next_state)
        delta_e = current_cost - next_cost

        if delta_e > 0 or random.random() < math.exp(delta_e / T):
            current_state = next_state
            current_cost = next_cost
```

lab4.1.py > ...

```
25  def simulated_annealing(n, T_initial, cooling_rate, max_iterations):
38              current_state = next_state
39              current_cost = next_cost
40
41              if current_cost < best_cost:
42                  best_state = current_state
43                  best_cost = current_cost
44
45          T *= cooling_rate
46
47          if best_cost == 0:
48              break
49
50      return best_state, best_cost
51
52  n = int(input("Enter the number of queens: "))
53  T_initial = 1000
54  cooling_rate = 0.99
55  max_iterations = 10000
56
57  solution, cost = simulated_annealing(n, T_initial, cooling_rate, max_iterations)
58
59  if cost == 0:
60      print("Solution found:")
61      print(solution)
62  else:
63      print("No solution found. Best solution:")
64      print(solution, "with cost:", cost)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Python

```
Enter the number of queens: 4
Solution found:
Enter the number of queens: 4
Solution found:
[2, 0, 3, 1]
PS C:\Users\BMSCECSE\Desktop\1BF24CS047>
```