LAB 6.

1.Write a program to implement Simulated Annealing Algorithm

Code:

```
import mlrose_hiive as mlrose
import numpy as np


def queens_max(position):
    no_attack_on_j = 0
    queen_not_attacking = 0
    for i in range(len(position) - 1):
        no_attack_on_j = 0
        for j in range(i + 1, len(position)):
            if (position[j] != position[i]) and (position[j] != position[i] + (j - i)) and
(position[j] != position[i] - (j - i)):
                no_attack_on_j += 1
        if (no_attack_on_j == len(position) - 1 - i):
            queen_not_attacking += 1
    if (queen_not_attacking == 7):
        queen_not_attacking += 1
    return queen_not_attacking


objective = mlrose.CustomFitness(queens_max)


problem = mlrose.DiscreteOpt(length=8, fitness_fn=objective, maximize=True,
max_val=8)
T = mlrose.ExpDecay()


initial_position = np.array([4, 6, 1, 5, 2, 0, 3, 7])
```

best_position, best_objective,fitness_curve=
mlrose.simulated_annealing(problem=problem, schedule=T,
max_attempts=500,init_state=initial_position)

print('The best position found is:', best_position)

print('The number of queens that are not attacking each other is:',
best_objective)

Output:

```
The best position found is: [4 1 7 0 3 6 2 5]
The number of queens that are not attacking each other is: 8.0
```

2.Tower Of Hanoi

Code:

```
import mlrose_hiive as mlrose
import numpy as np

def hanoi_fitness(state):
   correct_disks = 0
   destination_peg = 2
   for i in range(len(state)):
      if state[i] == destination_peg:
         correct_disks += 1
      else:
         break
   return correct_disks

fitness_fn = mlrose.CustomFitness(hanoi_fitness)
problem = mlrose.DiscreteOpt(length=3, fitness_fn=fitness_fn, maximize=True,
max_val=3)
schedule = mlrose.ExpDecay()

initial_state = np.array([0, 0, 0])
best_state, best_fitness, fitness_curve = mlrose.simulated_annealing(problem,
schedule=schedule, max_attempts=1000, init_state=initial_state)

print("Best state (final configuration):", best_state)
print("Number of correct disks on destination peg:", best_fitness)

def print_hanoi_solution(state):
```

```
    print("\nTower of Hanoi Configuration:")
    pegs = {0: [], 1: [], 2: []}
    for disk, peg in enumerate(state):
        pegs[peg].append(disk)
    for peg in pegs:
        print(f"Peg {peg}: {pegs[peg]}")

print_hanoi_solution(best_state)
```

Output:

```
Best state (final configuration): [2 2 2]
Number of correct disks on destination peg: 3.0

Tower of Hanoi Configuration:
Peg 0: []
Peg 1: []
Peg 2: [0, 1, 2]
```