Ahsanullah University of Science & Technology

Department of Computer Science & Engineering

Course No.	**:**	CSE 4108
Course Name	**:**	Artificial Intelligence Lab
Assignment No.	**:**	04

Submitted To :

Md. Siam Ansary	Tonmoy Hossain
Department of CSE, AUST	Department of CSE, AUST

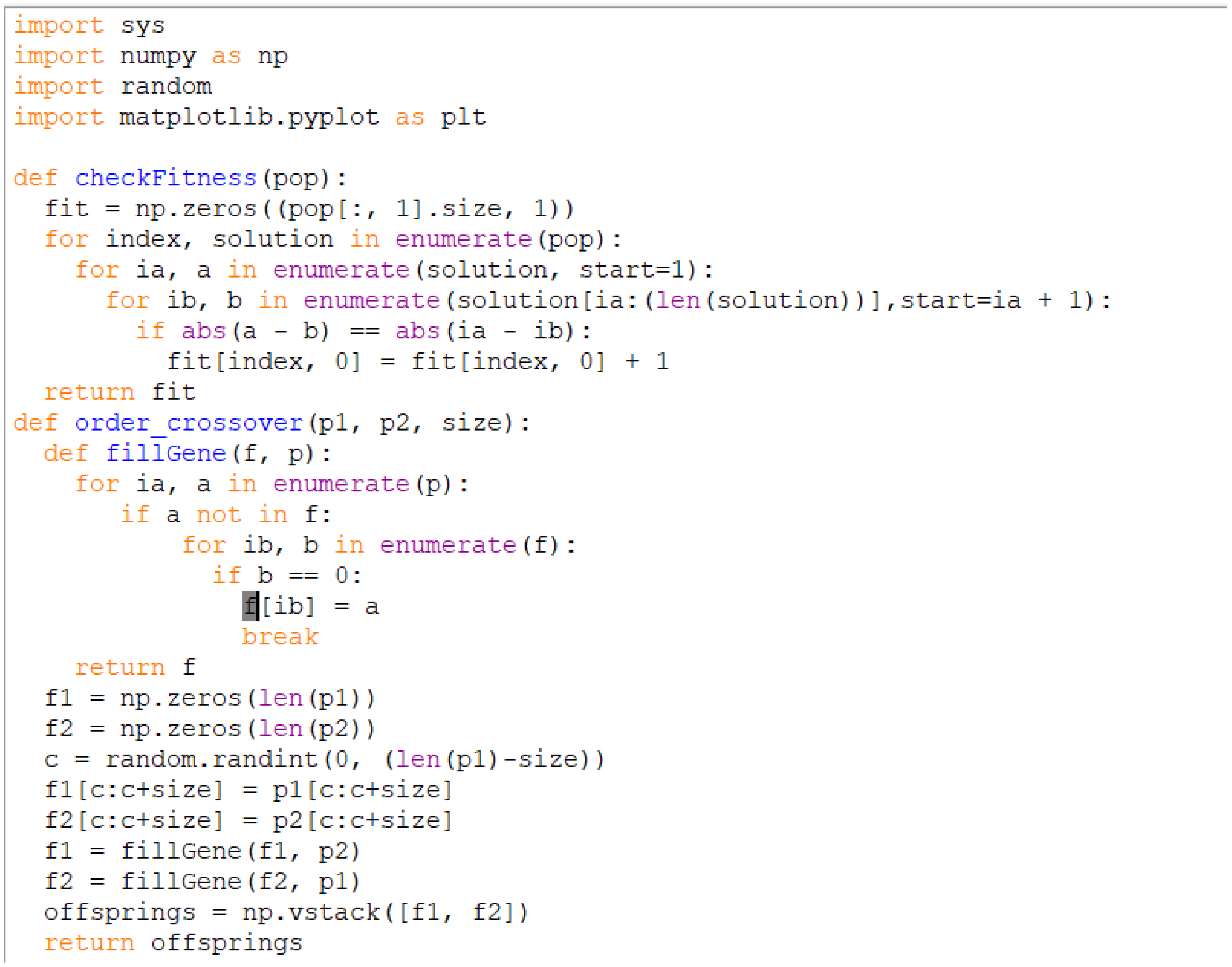
Submitted By:

Name	:	Sanjida Akter Ishita
ID No.	:	170204089
Session	:	Fall 2020
Section	:	B
Lab group	:	B2

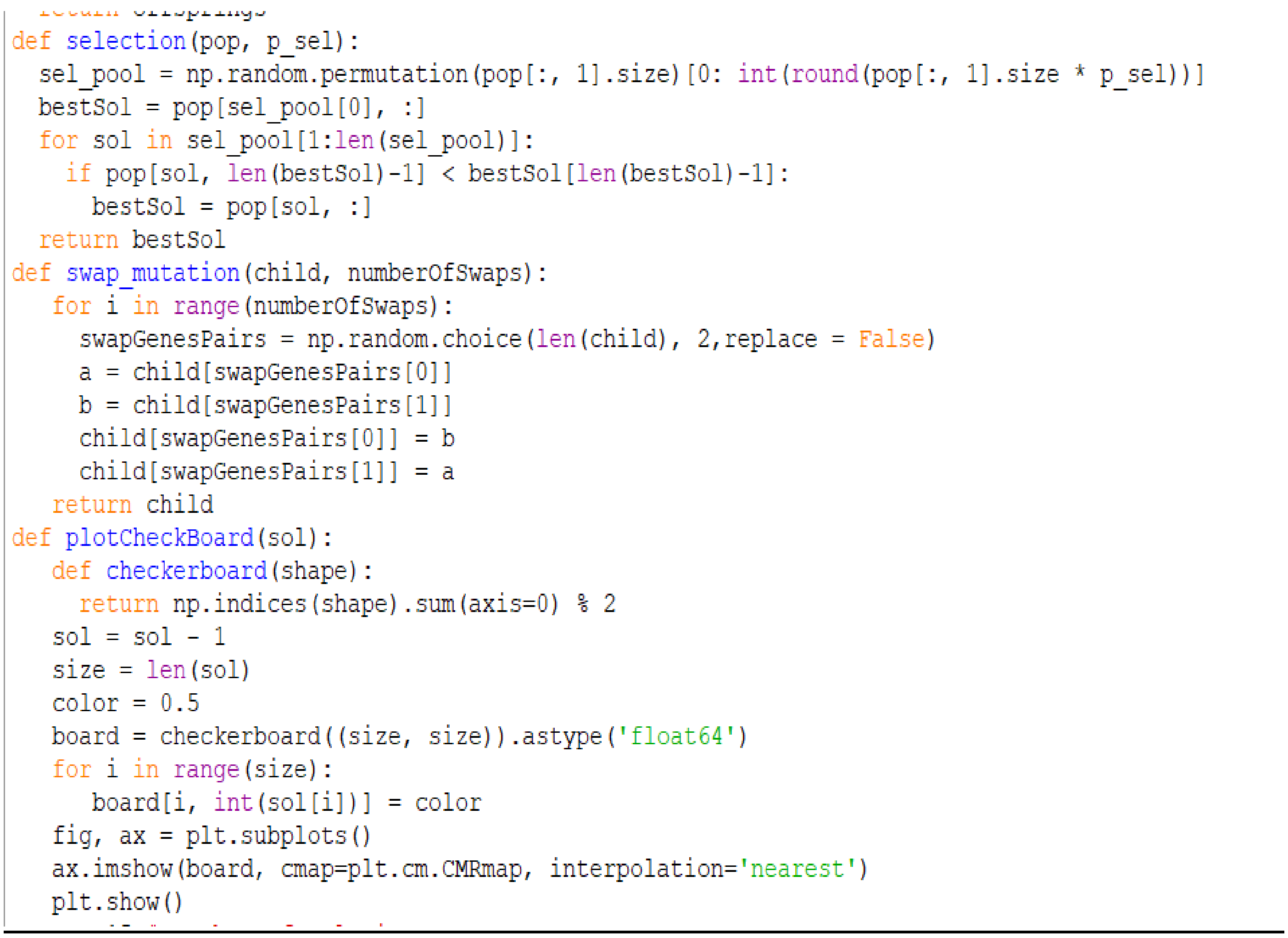
Date of Submission: September 8, 2021

QUESTION: Implement Genetic Algorithm for the 8 Queens Problem using Python. For Mutation step use Swap Mutation.

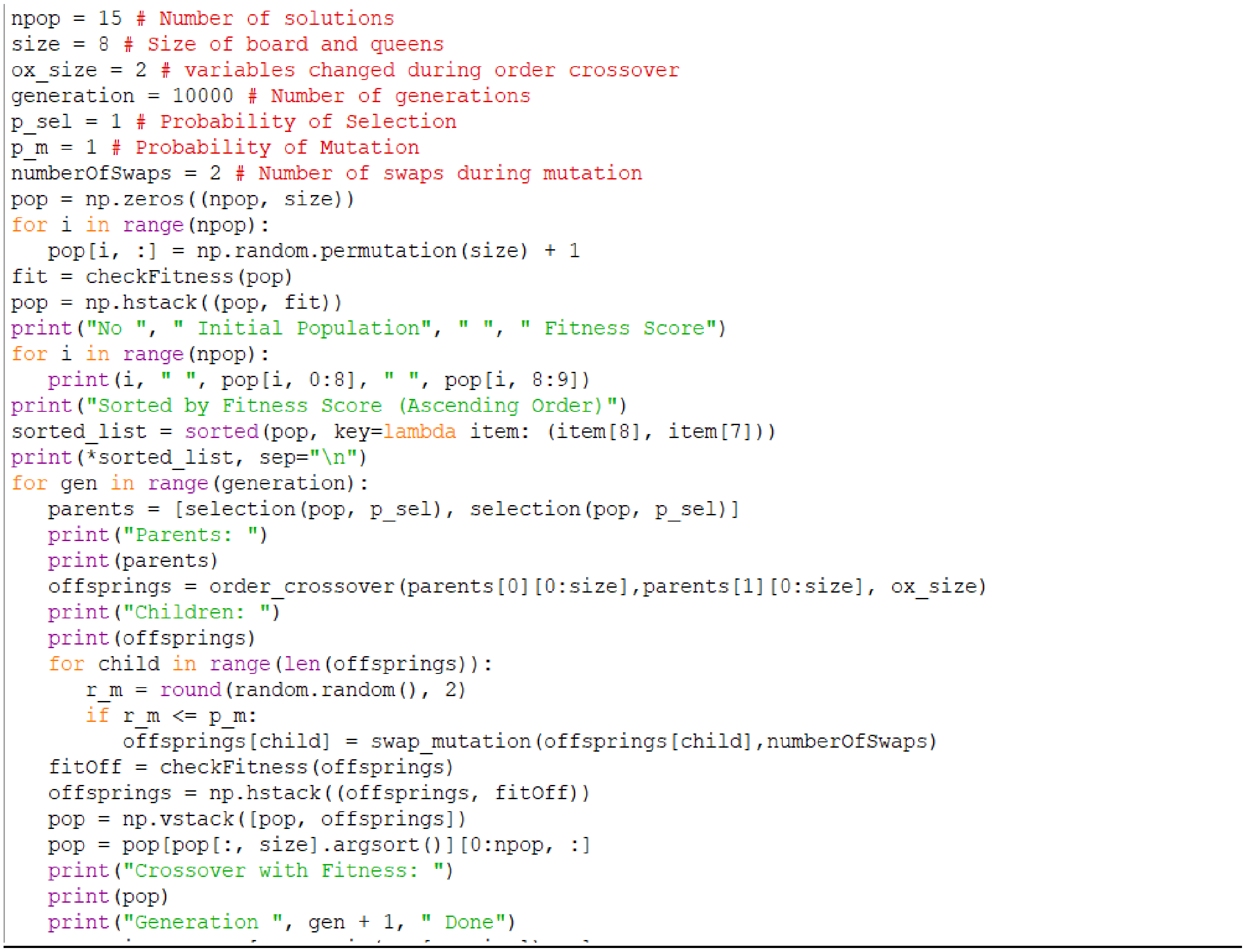
ANSWER: In this assignment I have to solve this problem using Genetic Algorithm with Swap Mutation. As output randomly taking some population will have shown with their fitness score and scoring them according to their fitness. I also will find the output in a chessboard.



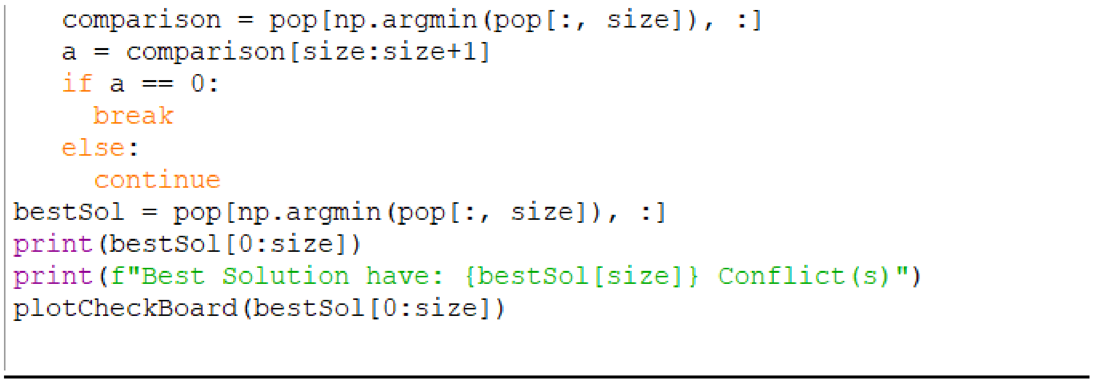
```
import sys
import numpy as np
import random
import matplotlib.pyplot as plt

def checkFitness(pop):
  fit = np.zeros((pop[:, 1].size, 1))
  for index, solution in enumerate(pop):
    for ia, a in enumerate(solution, start=1):
      for ib, b in enumerate(solution[ia:(len(solution))],start=ia + 1):
        if abs(a - b) == abs(ia - ib):
          fit[index, 0] = fit[index, 0] + 1
  return fit
def order_crossover(p1, p2, size):
  def fillGene(f, p):
    for ia, a in enumerate(p):
       if a not in f:
           for ib, b in enumerate(f):
             if b == 0:
               f[ib] = a
               break
    return f
  f1 = np.zeros(len(p1))
  f2 = np.zeros(len(p2))
  c = random.randint(0, (len(p1)-size))
  f1[c:c+size] = p1[c:c+size]
  f2[c:c+size] = p2[c:c+size]
  f1 = fillGene(f1, p2)
  f2 = fillGene(f2, p1)
  offsprings = np.vstack([f1, f2])
  return offsprings
```



```
  return offsprings
def selection(pop, p_sel):
  sel_pool = np.random.permutation(pop[:, 1].size)[0: int(round(pop[:, 1].size * p_sel))]
  bestSol = pop[sel_pool[0], :]
  for sol in sel_pool[1:len(sel_pool)]:
    if pop[sol, len(bestSol)-1] < bestSol[len(bestSol)-1]:
      bestSol = pop[sol, :]
  return bestSol
def swap_mutation(child, numberOfSwaps):
   for i in range(numberOfSwaps):
     swapGenesPairs = np.random.choice(len(child), 2,replace = False)
     a = child[swapGenesPairs[0]]
     b = child[swapGenesPairs[1]]
     child[swapGenesPairs[0]] = b
     child[swapGenesPairs[1]] = a
   return child
def plotCheckBoard(sol):
   def checkerboard(shape):
     return np.indices(shape).sum(axis=0) % 2
   sol = sol - 1
   size = len(sol)
   color = 0.5
   board = checkerboard((size, size)).astype('float64')
   for i in range(size):
      board[i, int(sol[i])] = color
   fig, ax = plt.subplots()
   ax.imshow(board, cmap=plt.cm.CMRmap, interpolation='nearest')
   plt.show()
```

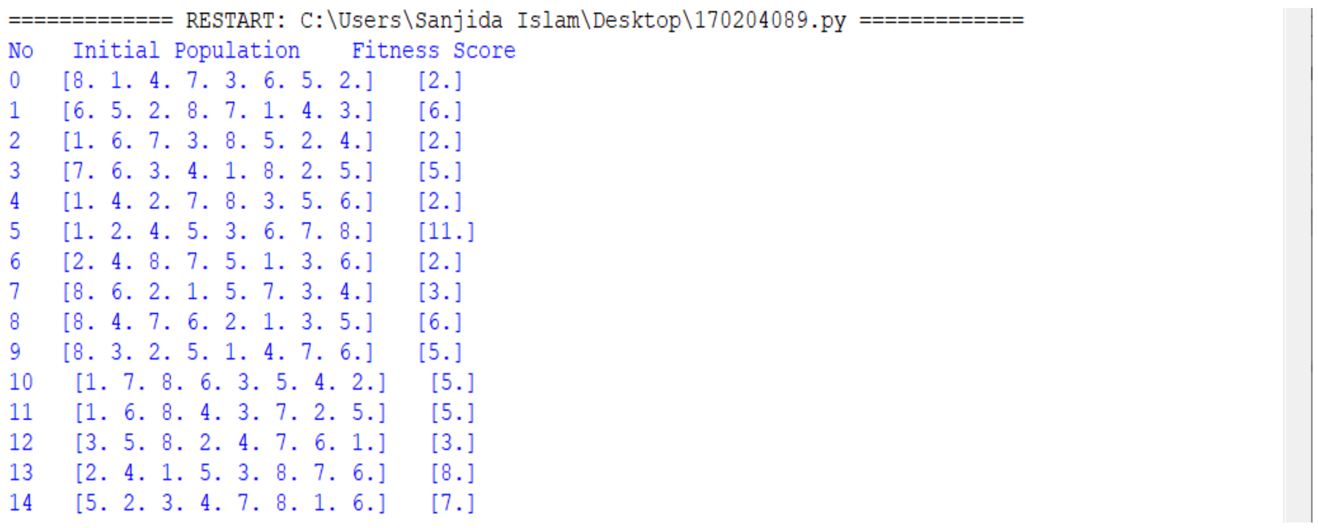


```
npop = 15 # Number of solutions
size = 8 # Size of board and queens
ox_size = 2 # variables changed during order crossover
generation = 10000 # Number of generations
p_sel = 1 # Probability of Selection
p_m = 1 # Probability of Mutation
numberOfSwaps = 2 # Number of swaps during mutation
pop = np.zeros((npop, size))
for i in range(npop):
   pop[i, :] = np.random.permutation(size) + 1
fit = checkFitness(pop)
pop = np.hstack((pop, fit))
print("No ", " Initial Population", " ", " Fitness Score")
for i in range(npop):
   print(i, " ", pop[i, 0:8], " ", pop[i, 8:9])
print("Sorted by Fitness Score (Ascending Order)")
sorted_list = sorted(pop, key=lambda item: (item[8], item[7]))
print(*sorted_list, sep="\n")
for gen in range(generation):
   parents = [selection(pop, p_sel), selection(pop, p_sel)]
   print("Parents: ")
   print(parents)
   offsprings = order_crossover(parents[0][0:size],parents[1][0:size], ox_size)
   print("Children: ")
   print(offsprings)
   for child in range(len(offsprings)):
      r_m = round(random.random(), 2)
      if r_m <= p_m:
         offsprings[child] = swap_mutation(offsprings[child],numberOfSwaps)
   fitOff = checkFitness(offsprings)
   offsprings = np.hstack((offsprings, fitOff))
   pop = np.vstack([pop, offsprings])
   pop = pop[pop[:, size].argsort()][0:npop, :]
   print("Crossover with Fitness: ")
   print(pop)
   print("Generation ", gen + 1, " Done")
```

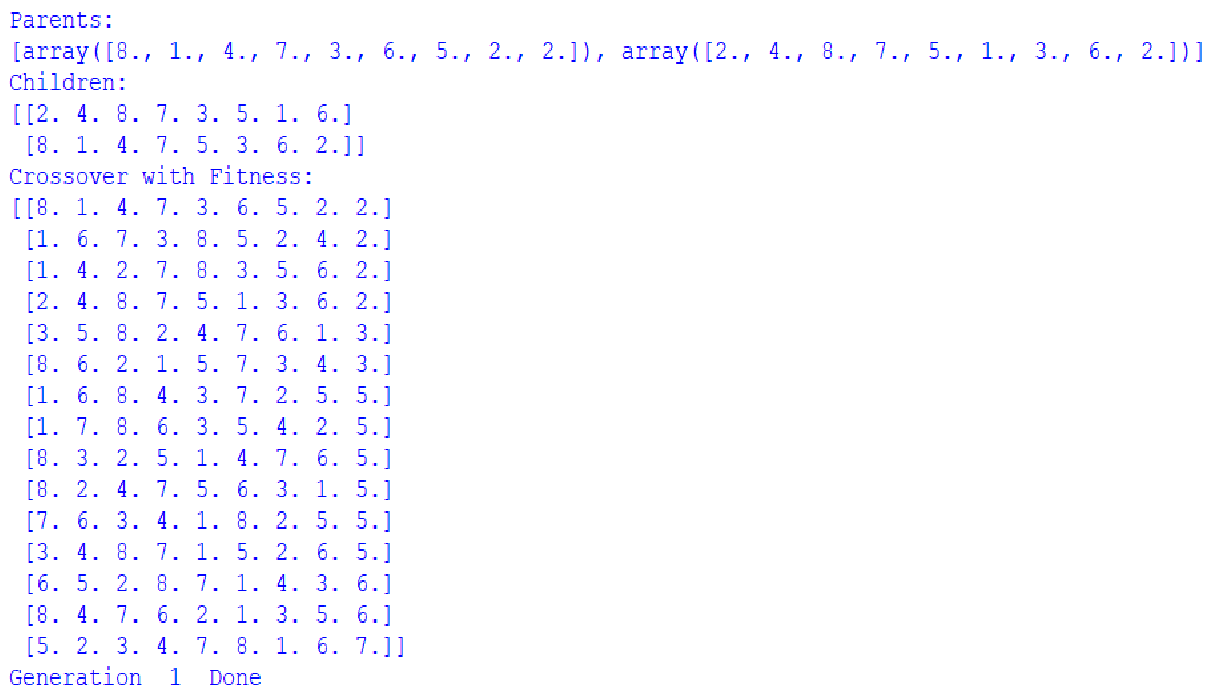


```
   comparison = pop[np.argmin(pop[:, size]), :]
   a = comparison[size:size+1]
   if a == 0:
     break
   else:
     continue
bestSol = pop[np.argmin(pop[:, size]), :]
print(bestSol[0:size])
print(f"Best Solution have: {bestSol[size]} Conflict(s)")
plotCheckBoard(bestSol[0:size])
```

Output:

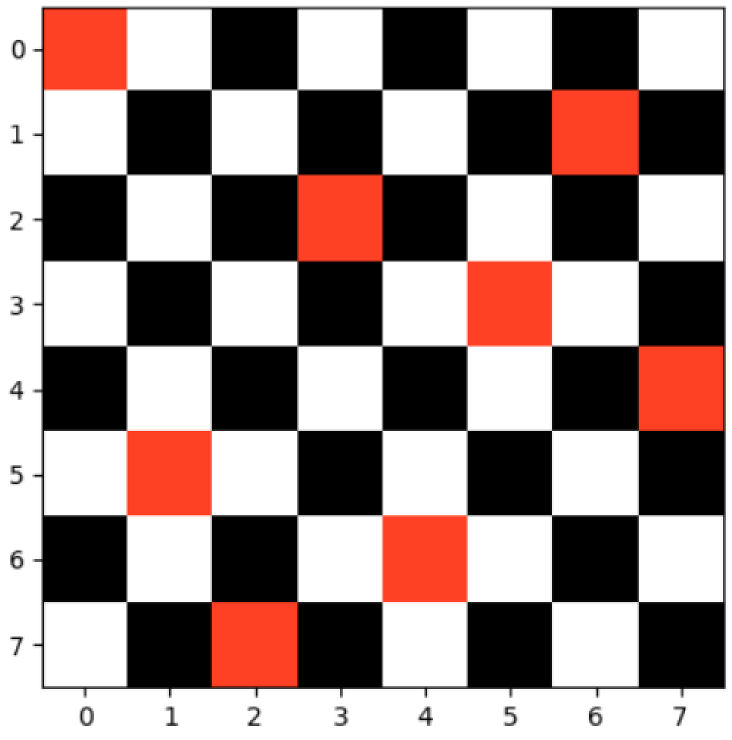


```
============= RESTART: C:\Users\Sanjida Islam\Desktop\170204089.py =============
No   Initial Population    Fitness Score
0   [8. 1. 4. 7. 3. 6. 5. 2.]   [2.]
1   [6. 5. 2. 8. 7. 1. 4. 3.]   [6.]
2   [1. 6. 7. 3. 8. 5. 2. 4.]   [2.]
3   [7. 6. 3. 4. 1. 8. 2. 5.]   [5.]
4   [1. 4. 2. 7. 8. 3. 5. 6.]   [2.]
5   [1. 2. 4. 5. 3. 6. 7. 8.]   [11.]
6   [2. 4. 8. 7. 5. 1. 3. 6.]   [2.]
7   [8. 6. 2. 1. 5. 7. 3. 4.]   [3.]
8   [8. 4. 7. 6. 2. 1. 3. 5.]   [6.]
9   [8. 3. 2. 5. 1. 4. 7. 6.]   [5.]
10   [1. 7. 8. 6. 3. 5. 4. 2.]   [5.]
11   [1. 6. 8. 4. 3. 7. 2. 5.]   [5.]
12   [3. 5. 8. 2. 4. 7. 6. 1.]   [3.]
13   [2. 4. 1. 5. 3. 8. 7. 6.]   [8.]
14   [5. 2. 3. 4. 7. 8. 1. 6.]   [7.]
```



```
Parents:
[array([8., 1., 4., 7., 3., 6., 5., 2., 2.]), array([2., 4., 8., 7., 5., 1., 3., 6., 2.])]
Children:
[[2. 4. 8. 7. 3. 5. 1. 6.]
 [8. 1. 4. 7. 5. 3. 6. 2.]]
Crossover with Fitness:
[[8. 1. 4. 7. 3. 6. 5. 2. 2.]
 [1. 6. 7. 3. 8. 5. 2. 4. 2.]
 [1. 4. 2. 7. 8. 3. 5. 6. 2.]
 [2. 4. 8. 7. 5. 1. 3. 6. 2.]
 [3. 5. 8. 2. 4. 7. 6. 1. 3.]
 [8. 6. 2. 1. 5. 7. 3. 4. 3.]
 [1. 6. 8. 4. 3. 7. 2. 5. 5.]
 [1. 7. 8. 6. 3. 5. 4. 2. 5.]
 [8. 3. 2. 5. 1. 4. 7. 6. 5.]
 [8. 2. 4. 7. 5. 6. 3. 1. 5.]
 [7. 6. 3. 4. 1. 8. 2. 5. 5.]
 [3. 4. 8. 7. 1. 5. 2. 6. 5.]
 [6. 5. 2. 8. 7. 1. 4. 3. 6.]
 [8. 4. 7. 6. 2. 1. 3. 5. 6.]
 [5. 2. 3. 4. 7. 8. 1. 6. 7.]]
Generation  1  Done
```

```
Sorted by Fitness Score (Ascending Order)
[8. 1. 4. 7. 3. 6. 5. 2. 2.]
[1. 6. 7. 3. 8. 5. 2. 4. 2.]
[1. 4. 2. 7. 8. 3. 5. 6. 2.]
[2. 4. 8. 7. 5. 1. 3. 6. 2.]
[3. 5. 8. 2. 4. 7. 6. 1. 3.]
[8. 6. 2. 1. 5. 7. 3. 4. 3.]
[1. 7. 8. 6. 3. 5. 4. 2. 5.]
[7. 6. 3. 4. 1. 8. 2. 5. 5.]
[1. 6. 8. 4. 3. 7. 2. 5. 5.]
[8. 3. 2. 5. 1. 4. 7. 6. 5.]
[6. 5. 2. 8. 7. 1. 4. 3. 6.]
[8. 4. 7. 6. 2. 1. 3. 5. 6.]
[5. 2. 3. 4. 7. 8. 1. 6. 7.]
[2. 4. 1. 5. 3. 8. 7. 6. 8.]
[ 1.  2.  4.  5.  3.  6.  7.  8. 11.]
```



Analyzing Code and Output: In output there are 15 randomly taking population and scoring or sorting them according to their fitness in ascending order. Then selecting 2 parents and did crossover and Swap Mutation to get children. After that they are sorted according to their fitness lastly. When fitness score is 0 and have 0 conflicts, we get the best solution.