In [1]:

```
import random
import copy
import math
import numpy as np
import time
```

In [2]:

```
cross_prob = 0
mutation_prob = 0
```

In [3]:

```
def print_board(individual, n):
    print()
    print('--------------N queens Board--------------')
    print()
    solution = []
    for x in range(n): # empty board!
        solution.append(['*'] * n)
    for z in range(n):
        solution[n-individual[z]][z] = 'Q'
    for z in solution:
        brackets_removed = str(z)[1:-1]
        brackets_removed = brackets_removed.replace("'", " ")
        brackets_removed = brackets_removed.replace(",", "")
        print(brackets_removed)
        print()
```

In [4]:

```
def fitness(individual):
    row_clashes = abs(len(individual) - len(np.unique(individual)))
    diagonal_collisions = 0
    n = len(individual)
    left_diagonal = [0] * 2*n
    right_diagonal = [0] * 2*n
    for i in range(n):
        left_diagonal[i + individual[i] - 1] += 1
        right_diagonal[len(individual) - i + individual[i] - 2] += 1
    diagonal_collisions = 0
    for i in range(2*n-1):
        counter = 0
        if left_diagonal[i] > 1:
            counter += left_diagonal[i]-1
        if right_diagonal[i] > 1:
            counter += right_diagonal[i]-1
        diagonal_collisions += counter / (n-abs(i-n+1))
    attack_pairs = row_clashes + diagonal_collisions
    return attack_pairs
```

In [5]:

```
def crossover(individual1, individual2, prob1):
    if prob1 < cross_prob:
        n = len(individual1)
        c = random.randint(0, n - 1)
        return individual1[0:c] + individual2[c:n]
    else:
        return individual1
```

In [6]:

```
def mutation(individual, prob2):
    if prob2 < mutation_prob:
        n = len(individual)
        c = random.randint(0, n - 1)
        m = random.randint(1, n)
        individual[c] = m
        return individual
    else:
        return individual
```

In [7]:

```
def generate_individual(n):
    result = list(range(1, n + 1))
    np.random.shuffle(result)
    return result
```

In [8]:

```
class Genetic(object):

    def __init__(self, n, pop_size):
        self.queens = []
        for i in range(pop_size):
            self.queens.append(generate_individual(n))

    def generate_population(self, random_selections=5):
        candid_parents = []
        candid_fitness = []
        for i in range(random_selections):
            candid_parents.append(self.queens[random.randint(0, len(self.queens) - 1)])
            candid_fitness.append(fitness(candid_parents[i]))
        test_list = []
        for i in range(0, len(candid_parents)):
            test_list.append([candid_fitness[i], candid_parents[i]])
        test_list.sort()
        x = test_list[0]
        y = test_list[1]
        temp1 = crossover(x[1], y[1], random.random())
        temp2 = crossover(y[1], x[1], random.random())
        p = mutation(temp1, random.random())
        q = mutation(temp2, random.random())
        if fitness(p) < x[0]:
            if not any(list == p for list in self.queens):
                self.queens.append(p)
                test_list.append([fitness(p), p])
        if fitness(q) < x[0]:
            if not any(list == p for list in self.queens):
                self.queens.append(q)
                test_list.append([fitness(q), q])

    def finished(self):
        for i in self.queens:
            if(fitness(i) == 0):
                return [1, i]
        return [0, self.queens[0]]

    def start(self, random_selections=5):
        count = 0
        while self.finished()[0] == 0:
            count = count+1
            self.generate_population(random_selections)
        final_state = self.finished()
        print()
        print('populations generated:', count)
        print()
        print(('Solution : ' + str(final_state[1])))
        print_board(final_state[1], n)
```

In [9]:

```
# ******************** N-Queen Problem With GA Algorithm ***********************
n = (int)(input('Enter the value of N : '))
max_pairs = (n*(n-1))/2
initial_population = (int)(input('Enter initial population size : '))
cross_prob = (float)(input('Enter crossover probablity:'))
mutation_prob = (float)(input('Enter mutation probablity:'))
begin_timer = time.time()
algorithm = Genetic(n=n, pop_size=initial_population)
algorithm.start()
print('Time Taken in seconds: ', time.time() - begin_timer)
```

```
Enter the value of N : 8
Enter initial population size : 10
Enter crossover probablity:.8
Enter mutation probablity:.2

populations generated: 4351

Solution : [6, 3, 1, 7, 5, 8, 2, 4]

--------------N queens Board--------------

 *   *   *   *   *   Q   *   * 

 *   *   *   Q   *   *   *   * 

 Q   *   *   *   *   *   *   * 

 *   *   *   *   Q   *   *   * 

 *   *   *   *   *   *   *   Q 

 *   Q   *   *   *   *   *   * 

 *   *   *   *   *   *   Q   * 

 *   *   Q   *   *   *   *   * 

Time Taken in seconds:  5.621634244918823
```

```
In [ ]:
```