Derek Zhang
026225028
Assignment 2
CECS 451

Hill Climbing Algorithm

```
PS C:\Users\026225028\Downloads\CECS451>  c:; cd '
c:\Users\026225028\Downloads\CECS451'; & 'C:\Users
\026225028\AppData\Local\Programs\Python\Python311
\python.exe' 'c:\Users\026225028\.vscode\extension
s\ms-python.python-2023.16.0\pythonFiles\lib\pytho
n\debugpy\adapter/../..\debugpy\launcher' '57659'
'--' 'C:\Users\026225028\Downloads\CECS451\Assignm
ent 2\hill-climbing.py'
Time taken:  0.0020020008087158203
Original:
[[0 0 0 0 0 1 0 0]
 [0 0 0 0 0 0 1 0]
 [0 0 0 0 0 0 0 1]
 [1 0 0 0 0 0 0 0]
 [0 0 0 0 1 0 0 0]
 [1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 1 0]
 [0 0 0 0 1 0 0 0]]
original fitness:  9
Solution found
[[0 0 1 0 0 0 0 0]
 [0 0 0 0 0 1 0 0]
 [0 0 0 0 0 0 0 1]
 [0 1 0 0 0 0 0 0]
 [0 0 0 1 0 0 0 0]
 [1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 1 0]
 [0 0 0 0 1 0 0 0]]
new fitness:  0
====================================
Number of restarts:  38
====================================
```

The hill climbing algorithm is a bit difficult to calculate for time complexity, as it functionally restarts whenever it reaches a local maxima. Given this problem, in the worst case scenario, it always gets trapped at a local maxima, and is forced to restart, giving it an infinite time complexity.

Genetic Algorithm

```
PS C:\Users\026225028\Downloads\CECS451>  c:; cd 'c:\Users\026225028\D
1'; & 'C:\Users\026225028\AppData\Local\Programs\Python\Python311\pyth
rs\026225028\.vscode\extensions\ms-python.python-2023.16.0\pythonFiles
ugpy\adapter/../..\debugpy\launcher' '62199' '--' 'C:\Users\026225028\
51\Assignment 2\genetic.py'
Starting algorithm...
Parameters:
 Max Generations: 1000
 Population Size: 10
 Mutation Chance: 25%
 Crossover Point: Random between 2-5
Solution found on generation 275
Time taken:  1.191084384918213
Solution found:
[[0 0 1 0 0 0 0 0]
 [0 0 0 0 1 0 0 0]
 [0 1 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 1]
 [0 0 0 0 0 1 0 0]
 [0 0 0 1 0 0 0 0]
 [0 0 0 0 0 0 1 0]
 [1 0 0 0 0 0 0 0]]
PS C:\Users\026225028\Downloads\CECS451>
```

The genetic algorithm's time complexity is more complex, as there are a lot of moving parts within it. Assuming n is the number of parents, and g is the number of generations:
In the worst case scenario, it would find the solution on the very last generation. Each generation would take at least n^2 time to calculate, since calculations need to be run for each parent, as well as generating the children of each parent. Additional calculation time is needed for mutations, which are randomized and as a result unreliable to predict. This can be added with the variable m for mutation time per generation. This results in the calculation being:

$$O(n) = g*(n^2+m)$$

Additional Notes:
I slightly modified the genetic algorithm's calculation from the ones we went over in class. Specifically, the probability of two parents breeding was changed from

$$P(n) = 28 - fitness / total$$

to a function that would favor lower attacking queen pairs. This function is as follows:

$$P(n) = 1.4^{28 - fitness / total}$$

Fitness in this case refers to the number of attacking queen pairs, and as a result 28-fitness results in higher numbers being more favorable.