Assignment 3

The task is to apply Genetic Algorithm to search for the shortest route, while

visiting all the locations once and the starting and ending point at the location number 1.

1. Explain the important operations of the employed algorithm. (0.5)

My implementation of the Genetic Algorithm to solve the Traveling Salesman Problem contains the following components:

1. **Initialization**: In the `initPopulation` function, an initial population of individuals (routes) is created. Each individual's order is a randomly shuffled version of the default order, and its distance is calculated using the `fitnessScore` function. An order instance is a sequence of integers [0,N-1] representing index of a location from the parsed locations data.
2. **Evaluation**: The fitness of each individual is the total distance of the route, calculated in the `fitnessScore` function. The shorter the distance, the fitter the individual. For the calculation of the distance, Euclidian distance formula is applied for each location to location link, including the one from the last to the first one to complete the cycle.
3. **Selection**: The `tournament` function is used for selection. It randomly selects `k`

   (5 in my case) individuals from the population and returns the fittest one, in other words the one with shortest total distance. This function is called twice in the `runGeneticAlgorithm` function to select two parents for crossover.
4. **Crossover**: The `crossover` function performs ordered crossover on two parent individuals to create a child. It randomly selects a subsequence from the first parent, and fills the remaining positions with the genes from the second parent in the order they appear, without duplicating any genes. This sometimes produces members of the new generation that inherit best parts of 2 members of a previous generation, likely leading to better results.
5. **Mutation**: The `mutateInsert` function performs mutation on an individual with a certain mutation rate. If a randomly drawn number is below the mutation rate, then the current individual is passed through the mutation process. It randomly selects two positions in the individual's order and swaps their values.
6. **Replacement**: In the `runGeneticAlgorithm` function, the old population is replaced with the new population of children and 10% of the fittest individuals from the old population (elitism).

2. Explain the representation of the individual , a solution to the problem, in your algorithm

(0.5)

In my algorithm, Individual is given by the following object:

```
struct Individual {
    vector<int> order;
    double distance;
}
```

Here, order is a vector of integers in the range [0, locations.size -1]. Its elements represent indexes of locations from the locations vector which is where the parsed input data has been stored.

Distance is the fitness score value calculated by the `fitnessScore` function for this order instance.

3. Give the equation of the fitness function used by your algorithm. (0.25)

My fitness score functions looks like this:

```
for (int i = 0; i < order.size() - 1; ++i) {
        score += sqrt(pow(locations[order[i+1]].x - locations[order[i]].x, 2)
    }

    score += sqrt(pow(locations[order[0]].x - locations[order.back()].x, 2) +
```

The equation for score calculation is given by the Euclidian distance formula for two points:

$d = \sqrt{(x2 - x1)^2 + (y2 - y1)^2}$
It adds up this distance for all paths in between cities in the order of the individual, from order[0] to order[size-1], and then adds the last distance from the last locations to the first to account for the cycle.

4. Give the parameters used in your algorithm. Examples: Population size, crossover rate ... (0.25)

From my main function:

```
int totalCalcs = 250000;
int generationsTotal = 1000;
int populationSize = totalCalcs / generationsTotal;
```

Population size = 250

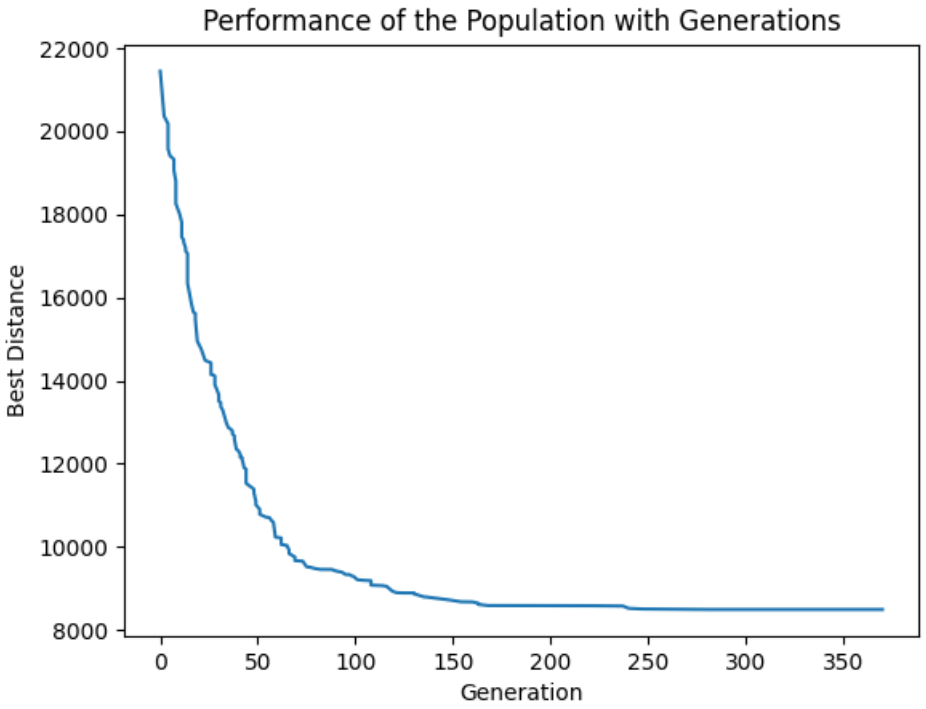
Mutation rate = 0.20

Elite size = 10% = 25

5. Illustrate how the performance of the population evolves with generations (with a figure.)

The figure below is generated based on best individual outputs from runGA function.

Here, it is notable that in the first 50 generations the population experiences rapid improvement as the distance goes down exponentially. Distance value of below 9000 is reached at approximately 120th generation, and the best distance overall was reached in generation 370. Afterwards, the population performance tends to converge to a certain minima and experience less variation from generation to generation.


Performance of the Population with Generations
Best Distance
Generation
22000
20000
18000
16000
14000
12000
10000
8000
0
50
100
150
200
250
300
350