**Annotations**

**Requirement I: Algorithm name: *2-opt algorithm***

1. **Strengths of the algorithm**
   * The travelling salesman problem is a combinatorial optimization problem which means can be categorised as NP-hard. With this in mind, then 2-opt was the choice to solve it because it is easy to implement and produces a local optima solution(an approximate solution to the global optima) based on 2 edges selected
   * The algorithm yields decent results at a very small implementation cost
   * To determine if a move will produce an improved tour is done through cost check e.g, if the cost(i,j) < cost(i+1, j+1) then improvement is met.
   * Considering the tour as a complete graph, the moves in this algorithm are feasible
2. **Verification**

+---------------------------------------------------------------------------------------------+

| 195 W Oakland Ave | 8:09:20 | Delivered |

+---------------------------------------------------------------------------------------------+

| 2010 W 500 S | 8:15:00 | Delivered |

+---------------------------------------------------------------------------------------------+

| 4300 S 1300 E | 8:29:00 | Delivered |

+---------------------------------------------------------------------------------------------+

| 4580 S 2300 E | 8:29:00 | Delivered |

+---------------------------------------------------------------------------------------------+

| 4580 S 2300 E | 8:29:00 | Delivered |

+---------------------------------------------------------------------------------------------+

| 177 W Price Ave | 8:35:40 | Delivered |

+---------------------------------------------------------------------------------------------+

| 3595 Main St | 8:45:40 | Delivered |

+---------------------------------------------------------------------------------------------+

| 1330 2100 S | 8:51:00 | Delivered |

+---------------------------------------------------------------------------------------------+

| 300 State St | 9:10:20 | Delivered |

+---------------------------------------------------------------------------------------------+

| 3365 S 900 W | 9:28:00 | Delivered |

+---------------------------------------------------------------------------------------------+

| 4580 S 2300 E | 9:39:00 | Delivered |

+---------------------------------------------------------------------------------------------+

| 410 S State St | 9:53:40 | Delivered |

+---------------------------------------------------------------------------------------------+

| 380 W 2880 S | 9:53:40 | Delivered |

+---------------------------------------------------------------------------------------------+

+ | COST or DISTANCE = 35.2 MILES | +

To verify that the algorithm generalizes well based on the criteria and requirements, we consider a sample output above from the first delivery made after running the algorithm. We can see from the output that the algorithm gets the shortest distance of 35.2 miles.

1. **Other heuristic algorithm options to consider using**
2. 3-opt algorithm
3. Greedy algorithm
4. Genetic algorithm

* The 3-opt algorithm is similar to our algorithm choice except that it considers transposing two nodes at a time. e.g, i->j->k->i which are the two possible ways of transposing.
* The greedy algorithm is the simplest improvement algorithm.
  + It starts with the departure node 1
  + calculates all distances to other n - 1 nodes
  + Moves to the closest node
  + Take current node as the departing node and select the next nearest node from the remaining n-2 nodes
  + The process continues until all the nodes are visited once and only once then back to Node 1
* The genetic algorithm uses the idea of natural selection. Obtain the maximum number of individuals in the population P and the maximum number of generations G from the user, generate P solutions for the first generation’s population randomly, and represent each solution as a string.
  + Determine fitness of each solution in the current generation’s population routes
  + Generate solutions for the next generation’s population

**Requirement J: Next trial**

Given this project again, I would consider using Lin Kernighan Heuristic algorithm because it is effective in the sense that it reaches a near-optimal solution for the symmetric travelling salesman problem.

**Requirement K: Data structure name: *Hash table***

**K1: Verification**

K1a: Efficiency

The data structure I opted for in a hashtable of arrays of tuples, the tuple holds the key/values. Using an interpolation search the values can be retrieved in *log(log(n)).* With this data structure, one can have 100% memory efficiency when you choose a perfect hash function for the keys that you want to store in advance. To make it faster, you make the table or array larger to obtain a good load factor because it has to resolve the least collisions, leading to the least amount of cache misses.

K1b: Overhead

Depending on the hash function chosen for finding the index, poor hash leads to collisions. Collision happens when the index of the key you want to insert in the table has a value already occupying the position. There are however, techniques to deal with this problem, for instance using double hashing.

K1c: Implication when changing the table

For a given hash function that is not degenerate, the largest table is also the fastest because it has to resolve the least collisions, leading to the least amount of cache misses.

**K2: Other options**

1. Array
2. Linked list

* In array data structure, a collection of elements is stored and each element is identified by an array index. Accessing an element requires an iteration over the array as opposed to a hash table in which you accept a value using its key with no iteration. This renders the hash table as the preferable choice.
* Linked list on the other hand stores collection of data elements whose order is not given by their physical placement in memory. Elements are basically nodes which represent a sequence and each node contains data and a reference or link to the next node in the sequence.

**References**

O. Amble, D. E. Knuth. *Ordered hash table*. <https://academic.oup.com/comjnl/article/17/2/135/525363>

Wikipedia: [2-opt algorithm](https://en.wikipedia.org/wiki/2-opt)

<https://arthur.maheo.net/python-local-tsp-heuristics/>

[Travelling salesman problem](https://en.wikipedia.org/wiki/Travelling_salesman_problem)