# WGUPS Routing Program Overview

Stated Problem:

The purpose of this project is to create an algorithm using Python to develop a program that could be used to route package delivery. The program needs to successfully create a route that can deliver 40 packages using 3 trucks, each able to hold 16 packages. The total distance traveled must be kept under 140 miles for all trucks. The packages must be delivered by their scheduled times and may contain special instructions which will need to be followed.

## Algorithm Overview:

For this problem an implementation of Dijkstra’s Shortest Path Algorithm will be used to create the routes. A weighted adjacency matrix can be created from the provided distance table which can be used to implement the algorithm. The algorithm is created as follows:

1. Initialize all nodes with an infinite distance and add them to a queue of unvisited nodes
2. Set the start node distance to 0 and put it at the start of the queue
3. Create a loop that dequeues each unvisited node as the current node
4. Create a second loop to visit all the neighboring nodes to the current node and calculate their distance from the current node.
   1. if the new distance calculated is less than the node’s current distance set the distance equal to the new distance. A pointer to the prior node also needs to be set on the neighboring node to the current node
5. Repeat this process until all nodes have been visited

Operation time for Dijkstra’s Algorithm using a list for the unvisited queue is O(N^2). For this project I will use a minimum binary heap to reduce the complexity to O((E + N) log N). A binary heap is a strong choice because of the ordered structure of the data. The values with the lowest weight will be the quickest to access meaning that the program should be able to quickly identify the fastest path. Some drawbacks to using a binary heap is the inability for a CPU to use instruction-level parallelism in push/pop operations which could affect real world run times. Another disadvantage is that it can be difficult to search a binary heap efficiently for a specific node.

Each node in the binary heap will contain a reference to a package object for each package allowing access to information about each package such as address, weight, or status as object properties. A package ID will be used as the key to point to each individual package as each ID should be unique.

Dijkstra’s shortest path

**unvisitedQueue = []**

**unvisitedQueue.heapify  
For package in packages:**

**package.distance = float(‘inf’)**

**package.predecessor = None**

**unvisitedQueue.heappush(package)**

Building the queue using a minimum binary heap is O(log N)

**while (currentPackage = unvistedQueue.heappop):**

**for adjP in currentPackage.adjPackages:**

**edgeWeight = adjMatrix[currentPackage.adjIndex][adjP.adjIndex]**

**alternativePathDistance = currentPackage.distance + edgeWeight**

**if (alternativePathDistance < adjP.distance):**

**adjP.distance = alternativePathDistance**

**adjP. predecessor = currentPackage**

Removing an item from the minimum binary heap is O(log N) and then checking each of the adjacent packages (E) means the inner loop runs in E + N meaning the overall runtime for this portion of the algorithm is O((E+N) log N). The overall coding should run in O((E+N) log N). Additional handling will need to be encoded for grouped packages which have different delivery locations, they can be incorporated as a single package object with the edge weight values adjusted to account for the necessity of all packages being delivered on the same truck at the same time. The edge weight could also be calculated to include other factors such as scheduled delivery time.

The space complexity for the program can be measured by the two data structures needed for it to operate. First would be the initial package array O(P). Second would be the priority queue used to run the algorithm, denoted O(N). Overall the space complexity should be O(P+N)

This solution should scale comparatively well because the time complexity is logarithmic. This means that the amount of time required will not grow as rapidly for larger numbers of packages. Because the value used to determine which package to add to the truck next is generated based of a variety of factors the program should be easy to maintain and add additional functionality in the future. For example, if some sort of package priority needed to be added the edge weight calculation could be modified to include this value.

The programming environment I will be using for this task is VScode with Pylance and Python Debugger extensions. It will run on a desktop computer with an Intel i5-9600K and 32 gb of ram. The Python version I will use is 3.9.