WGUPS Routing Project Summation

Current WGUPS Algorithm Strengths

The current utilization of a heuristic sorting algorithm allows the WGUPS program to filter through large amounts of data with efficiency and scalability. With the use of the current Greedy algorithm in the package-to-route assigning and sorting process, the code is efficient and expandable by merit of how the code doesn't have to search for each optimal route for each package. Instead, this technique is able to compare smaller amounts of data to still find largely optimal routes for each item. Furthermore, with the time complexity of this algorithm being menial, this allows for faster and less intensive computing to be done, which can benefit the end user by mitigating resource usage.

Other Program Approaches

Similar to the Greedy methodology, other quicksorting algorithms such as the Quick Sort and Selection Sort methods also provide an efficient approach to organizing vast amounts of data. While both of these examples could meet the requirements of the WGUPS routing program, each uses procedures that can create a less than optimal result. With Quick Sort, the use of a pivot element is implemented to sort one section of data independent from the other, which would find and place shorter values based on the pivot but possibly not in an optimal order. With Selection sorting, data is organized based on whether or not it is more or less than a single element in a deduced set and then switches position accordingly. This can lead again to less than optimal sorting because the efficiency of the algorithm is O(n2), which may negatively affect the overall time complexity of the whole program.

While these techniques do help surface certain values, they both are constructed on the premise of sifting a section of data. The Greedy method differs from their approaches to this concept, as it targets the smallest value from a whole set of data that starts large and then narrows, allowing for more initial efficiency to be achieved.

Both the Quick Sort and Selection Sort could be added to this program in place of the Greedy method currently implemented for the package-to-route assigning and sorting processes. And, though both could provide a means of sorting and placing packages into the available trucks, the Greedy algorithm is perceived as more optimal in this situation because of its complexity efficiency and simple functionality of always choosing the smallest number at each point as it functions in the program provided in this scenario.

Project Improvements and Refinement

If this project were to be reworked, improvements and refinements of the data storage techniques and increased versatility of the program would be centerpoints. For data storage, if the project specifications would allow Python packages to be utilized, the code would have the opportunity to be more intuitive and readily extracted upon. And for increased versatility of this program, a more uniform data set would be ideal. Namely, if for the special notes section of the package data there were instead codes to be used in different

scenarios, this would allow for simpler processing of their specifications. Such inefficiencies could be solved by employing more intensive computational techniques such as an LLM to analyze the special notes automatically, but this would greatly increase the running resources needed for the project. More generally, this program could benefit from the refinement of specific package management to a more adaptive architecture that doesn't rely on user modification to achieve certain data outcomes.

Data Structure Analysis

The main data structures that comprise this WGUPS project were arrays and a package hashtable. These arrangements of data allowed for efficient and low-level functionality for the programs working data. Other data structures that would also meet the same requirements for this project include dictionaries (as in the Python dictionary architecture) and custom objects that could be built to harbor sets of data in a more congealed and organized manner.

With a Python dictionary architecture, the usage would be similar to the already used arrays and hashtable, but the overall process of creating and managing for both would be streamlined via the prebuilt architecture of the Python dictionary system.

With custom objects, the approach to data storage and manipulation would be more intuitive and allow for a lower complexity system overall due to it being a more simple approach (*such an object could be named "Package", and have different variables assigned within it to hold the location, truck number, delivery status and address, ect*.). Unlike the currently employed arrays that hold many different data sets in the current program, with custom objects different information could be stored in a more clear variable and can subsequently be operated on in a more clear manner as well.