## F. Justification of Package Delivery Algorithm:

### 1. STRENGTHS OF THE HASH TABLE ALGORITHM:

- **Greedy Algorithm Strengths:**
  - **Incremental Route Decision-making:** The greedy approach is designed for making decisions that seem best at the current moment. This results in decisions that are locally optimal, meaning it chooses the next best package or route without considering the entire set of unprocessed packages. While this might not always lead to the global optimum, it does provide a reasonable and quick solution.
  - **Priority Handling:** Your system prioritizes certain packages based on delivery deadlines. The greedy nature of the algorithm helps in making these on-the-fly decisions to prioritize immediate deliveries.
  - **Scalability:** The greedy algorithm is generally fast, which means it can handle a larger number of packages without a significant increase in processing time. However, its ability to consistently find the most efficient route isn't guaranteed as the dataset grows.
- **Hash Table Strengths:**
  - **Adaptive Data Handling:** Hash tables are dynamic data structures, meaning they can handle changes like additions and deletions efficiently. This makes them suitable for applications where package details might be updated frequently.
  - **Immediate Access:** Hash tables, when well-implemented, offer average O(1) time complexity for look-ups. This speed ensures quick data retrieval, which is crucial for real-time applications.
  - **Real-time Status Updates:** The efficiency of hash tables enables the system to quickly update and retrieve the status of a package, making it possible to track real-time changes.

### 2. MEETING SCENARIO REQUIREMENTS:

- Efficient Package Management: With hash tables, packages are inserted and retrieved rapidly based on package ID, ensuring timely loading onto trucks and tracking.
- Accurate Delivery Status: Hash table design tracks the delivery status changes in real-time, from 'at the hub', 'en route', to 'delivered' states, including the exact time.

### 3. ALTERNATE ALGORITHMS:

- Nearest Neighbour Algorithm:
- Prioritizes immediate proximity and creates routes based on the nearest unvisited package. This might be more efficient in scenarios where packages are clustered.
- Christofides Algorithm:
- An approximation algorithm that guarantees delivery routes within 1.5 times the optimal solution. Combines minimum spanning trees with the shortest path problem to ensure better efficiency than the nearest neighbour.

#### A. DIFFERING ASPECTS:

- Nearest Neighbour: Prioritizes immediate closeness but can lead to longer overall paths if not recalculated frequently.
- Christofides: More computationally intensive but can result in more efficient routes, especially for larger sets of packages.

## G. Project Retrospective:

- Enhanced Routing: Further optimizations can be made by integrating real-time traffic data or considering package priorities more dynamically.
- Modular Design: Future revisions might incorporate a more modular code structure, allowing easier testing, debugging, and scalability.

## H. Data Structure Suitability:

- Effectiveness: Your hash table efficiently addresses the requirements of package insertion, retrieval, and status updates, ensuring timely deliveries.

### A. ALTERNATE DATA STRUCTURES:

- Balanced Binary Search Trees (e.g., AVL Trees):
- Allows for keeping data in a sorted manner and ensures that the trees remain balanced for efficient O(log n) look-ups.
- Array-based Lists:
- Fixed-size datasets can use array-based lists for efficient access, especially if combined with binary search on sorted lists.

### B. DIFFERENCES:

- Balanced Trees: They keep data sorted and automatically balance themselves post insertions/deletions. This might provide slower insertions as compared to hash tables due to rebalancing but ensures efficient look-ups.
- **Array-based Lists**: Allow for constant-time access to elements based on index but need O(n) time for search operations if not sorted. Sorting allows for O(log n) searches but requires extra memory.