Student’s Name: Jimmy Aubert

Professor’s Name: Judy Ligocki

Course Number: D777

Date: 06/01/2025

Designing a Scalable and Efficient Inventory Management System for Corollary Warehousing

While working at Pruhart Tech Solutions, I was entrusted with building an advanced *inventory* and order system for the logistics company Corollary *Warehousing*. The main goal for the client is to have *inventory* tracked in real-time, all orders processed smoothly, more space managed efficiently, alongside reliable analysis, and still maintain scalability and fault tolerance. To meet these business needs, I selected five key data structures that support fast operations, efficient memory usage, and maintainable architecture.

**A. Identification of Data Structures**

**Hash Table (Dictionary)**

Hash tables provide fast, real-time access to inventory by using unique product identifiers as keys. This allows constant-time (O(1)) operations for lookups, insertions, and deletions. They are ideal for checking stock levels, updating quantities during restocking, or processing orders quickly without scanning entire datasets (Tapia-Fernandez et al.).

**Priority Queue (Heap)**

Priority queues help optimize *warehouse* space by organizing products based on urgency, such as turnover rate or expiration date. Items with higher priority (for example, fast-moving or perishable goods) are stored for easy access, reducing waste and improving retrieval efficiency.

**Binary Search Tree (BST)**

A BST enables sorted inventory management by quantity, SKU, or price. When implemented as a “self-balancing *tree*” like an AVL *tree*, it maintains O(log n) performance for insertion, deletion, and search (Yuan and Belavina). This makes it suitable for generating sorted reports and performing range queries.

**Graph**

Graphs are used to map *warehouse* networks, where nodes stand for *warehouse* locations and lines for transportation between them. It helps direct orders, watch shipments, and find the most efficient route between *warehouses.*

**Queue (FIFO)**

FIFO queues manage “order processing” in the sequence received, ensuring fairness and real-time handing out. They help maintain “order consistency” and streamline the fulfillment channel across all *warehouses*.

**B. Evaluation and Comparison of Data Structures**

**Comparative Performance Analysis**

Each selected data structure offers unique advantages. Below is a summary of their time and space complexities:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Data Structure | Lookup | Insert | Delete | Space |
| Hash Table | O(1) avg | O(1) avg | O(1) avg | O(n) |
| Priority Queue | O(n) | O(log n) | O(log n) | O(n) |
| Binary Search Tree | O(log n) | O(log n) | O(log n) | O(n) |
| Graph (Adj. List) | O(1)–O(n) | O(1) | O(1) | O(V + E) |
| Queue (FIFO) | O(1) | O(1) | O(1) | O(n) |

**Tradeoffs and Efficiency**

Each data structure offers benefits and limitations.

* Hash tables enable fast inventory updates but do not maintain any order, making them unsuitable for sorted views (Tapia-Fernandez et al.).
* Priority queues allow intelligent prioritization of items, such as those nearing expiration, but are inefficient for random access or bulk updates.
* Storing data sorted in a Binary Search Tree (BST) can facilitate reporting, although constant balancing becomes necessary to avoid inefficiency (Yuan and Belavina).
* Graphs show how warehouses are organized effectively but can become cluttered as more parts are added.
* Queues make everything easy and consistent, but they do not offer features for handling complicated queries or data ties.

**System Reliability and Faults Tolerance**

Storing *data* across several servers or nodes using hash tables adds reliability and the ability to deal with faults (Tapia-Fernandez et al.). In addition, by incorporating an in-memory cache, Redis makes sure the *data* is safe and can be accessed quickly (Geng et al.). When queue-based tools such as deque or message queues are used for order processing, every service can work alone so that failure in one part does not lead to a shutdown of the service.

**Recommended Structures**

* **Inventory Tracking:** Hash Tables
* **Order Queueing:** FIFO Queue
* **Storage Prioritization:** Priority Queue
* **Report Generation:** BST
* **Inter-Warehouse Logistics:** Graphs

The data structures ensure a good balance in terms of performance, capability to grow, and dependability. They help distribute the system across many servers so it can handle more work as Corollary *Warehousing* expands.

**C. Identification and Evaluation of Python Libraries**

I suggest using the following Python libraries for applying the discussed data structures:

1. *collections*

Offers data structures that work efficiently and are high-performance, including *deque*, *defaultdict*, and *Counter*. It is best suited for queues and collections of similar items.

1. *heapq*

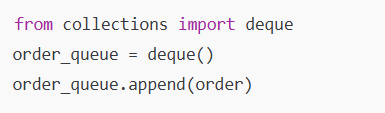
Allows developers to use priority queues. A binary heap algorithm is used, and this technique supports priority storage or fulfillment of items.

1. *networkx*

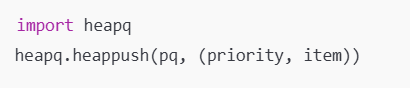
A good library for making and studying graphs. It enables the setting up of *warehouse* locations, the study of shipping routes, and the improvement of the paths.

**Useful Functions and Examples**

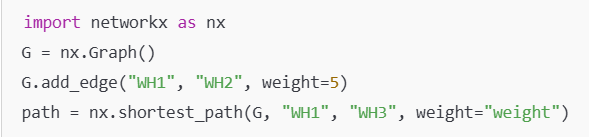
* From *collections*:
  + deque(): Fast appends/pops from both ends, used for FIFO order queues.



* From *heapq*:
  + heappush() and heappop(): Insert or delete elements in a priority queue.



* From *networkx*:
  + Shortest\_path(): Identifies the quickest way to move products within the warehouses.



**D. Application to Business Needs**

High-performing data structures and Python libraries respond to Corollary *Warehousing’s* most important requirements. Using a *hash table* allows real-time checking of inventory levels, *priority queues* organize storage and orders effectively, and *graphs* make delivery routing more efficient. The system is well designed because it uses *collections*, *heapq*, and *networkx* libraries to be stable, adapt to higher data volumes, and handle problems.

**Works Cited**

Geng, Yingjie, et al. “ER-KV: High Performance Hybrid Fault-Tolerant Key-Value Store.” *2021 IEEE 23rd Int Conf on High Performance Computing & Communications; 7th Int Conf on Data Science & Systems; 19th Int Conf on Smart City; 7th Int Conf on Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys)*, IEEE, Dec. 2021, pp. 179–88, <https://doi.org/10.1109/hpcc-dss-smartcity-dependsys53884.2021.00050>. Accessed 31 May 2025.

Tapia-Fernández, Santiago, et al. “Key Concepts, Weakness and Benchmark on Hash Table Data Structures.” *Algorithms*, vol. 15, no. 3, Mar. 2022, p. 100, <https://doi.org/10.3390/a15030100>.

Yuan, Lorraine, and Elena Belavina. *Privacy-Preserving Data-Driven Inventory Management*. Jan. 2025, <https://doi.org/10.2139/ssrn.5219878>.