4-2 Milestone Three Enhancement Two Algorithms and Data Structure

# Algorithms and Data Structure

URL: <https://github.com/oluwasegunsnhu/CS499_Artifact_2/>

The previous Linear Search Algorithm was done to work with simple codes for linear time searching. The aim was to correctly search for a product in less time. However, this code was not sufficient to work with software, databases, and mobile applications. In this artifact, linear search algorithm was updated to work with inventory management system database. However, due to several limitations, Linear Search Algorithm was replaced by B+ Tree algorithm.

The B+ tree is a balanced tree data structure that is widely used in database systems for its efficient search, insertion, and deletion operations. It organizes data in a hierarchical structure with multiple levels of nodes, which enables efficient access to the desired data. In a B+ tree, each internal node acts as an index, containing key-value pairs that guide the search process. The leaf nodes, at the bottom level of the tree, store the actual data records. This hierarchical arrangement allows for quick navigation from the root node to the appropriate leaf node, reducing the number of comparisons required to locate the desired data.

B+ trees are particularly well-suited for databases due to their ability to handle both point queries and range queries effectively. Point queries involve searching for a specific key in the database, and B+ trees excel in this scenario by minimizing the number of comparisons needed to find the desired record. Range queries, on the other hand, involve searching for a range of keys within the database. B+ trees are designed to maintain data in sorted order, enabling efficient range queries by easily traversing the tree to retrieve all records falling within a given key range. The performance of B+ trees is notable even for large databases. Most operations, including search, insertion, and deletion, have a logarithmic time complexity. This means that as the size of the database grows, the number of operations required to access or manipulate the data increases at a much slower rate compared to linear search or other data structures. Furthermore, B+ trees optimize disk access, making them suitable for databases stored on disk or in scenarios where memory is limited. By maximizing the number of keys in each node, B+ trees reduce the number of disk reads and writes, thereby minimizing I/O operations and improving overall performance.

# New Improvements:

The B+ tree algorithm provided for the inventory management database offers several advantages over a linear search algorithm. Here are the main points:

1. **Efficient Search:** The B+ tree algorithm provides efficient search operations with a time complexity of O(log n), where n is the number of items in the inventory. This logarithmic time complexity ensures fast retrieval of items even in large databases.
2. **Sorted Data Structure:** The B+ tree organizes the inventory data in a sorted manner based on the keys. This sorted structure enables efficient search operations using techniques like binary search within each node of the tree, reducing the number of comparisons needed to find a specific item.
3. **Range Queries:** The B+ tree is well-suited for range queries, allowing users to search for items within a specific range of keys. This capability is essential in inventory management systems where users may want to find all items with keys falling within a particular range, such as finding all items with prices between a minimum and maximum value.
4. **Reduced Disk I/O:** The B+ tree's hierarchical structure and balanced nature minimize disk I/O operations. As a result, accessing data from secondary storage (e.g., disk) is optimized, reducing the overall time required to search for items in the database.
5. **Efficient Insertion and Deletion:** The B+ tree algorithm provides efficient insertion and deletion operations while maintaining the balanced nature of the tree. When adding or removing items, the tree structure is adjusted through node splitting or merging, ensuring that the tree remains balanced and efficient for subsequent search operations.
6. **Scalability:** The B+ tree algorithm scales well for large inventory databases. As the number of items increases, the tree's height remains relatively small due to its balanced nature, resulting in efficient search operations that are not significantly affected by the database size.
7. **Better Performance:** Overall, the B+ tree algorithm offers superior performance compared to linear search, especially when dealing with large and frequently accessed inventory databases. The logarithmic time complexity, efficient range queries, reduced disk I/O, and balanced structure contribute to faster search operations and improved system responsiveness.

The B+ tree algorithm provides efficient search, range queries, and insertion/deletion operations, while minimizing disk I/O and offering scalability. These characteristics make it a suitable choice for an inventory management database, outperforming linear search in terms of performance and responsiveness.

## Briefly describe the artifact. What is it? When was it created?

Linear Search Algorithm was designed for a course project in “Data Structures and Algorithm” course. This algorithm was simply searching and accessing data elements. However, it had poor speed and performance when working for real-word applications. In this module, the artifact was a C++ code implementation of a B+ tree algorithm for an inventory management database.

## Justify the inclusion of the artifact in your ePortfolio. Why did you select this item? What specific components of the artifact showcase your skills and abilities in algorithms and data structure? How was the artifact improved?

I selected this artifact for inclusion in my ePortfolio because it demonstrates my proficiency in designing and implementing advanced data structures and algorithms. The B+ tree algorithm showcases my understanding of balanced tree structures, efficient search operations, and handling large datasets. The artifact was improved by incorporating additional features such as persistence using file I/O, error handling, memory management, duplicate handling, and a basic user interface for better usability.

## Did you meet the course objectives you planned to meet with this enhancement in Module One? Do you have any updates to your outcome-coverage plans?

Yes, the enhancement of the artifact aligns with the course objectives outlined in Module One. It allowed me to apply my knowledge of algorithms and data structures to a practical problem in the form of an inventory management system. The additional features implemented in the artifact demonstrate my ability to extend and enhance existing algorithms to meet specific requirements. There are no updates to my outcome-coverage plans currently.

## Reflect on the process of enhancing and/or modifying the artifact. What did you learn as you were creating it and improving it? What challenges did you face?

During the process of enhancing the artifact, I learned the importance of persistence in database systems and how to incorporate it using file I/O operations. I also gained experience in error handling and memory management techniques to ensure the robustness and efficiency of the code. One of the challenges I faced was handling deletion from internal nodes in the B+ tree algorithm, which required additional logic and considerations. However, through research and experimentation, I was able to overcome this challenge and successfully implement the desired functionality. Overall, the process of enhancing the artifact provided valuable insights into real-world application of algorithms and data structures, as well as practical considerations for database management systems.