**Final Project Report – Online Ordering System**

# 1. Project Overview:

This project is a console-based Online Ordering System developed in C++ using core data structures such as stacks, queues, arrays, and sorting algorithms. It simulates an e-commerce platform that allows users to browse products, add them to a cart, undo actions, place orders, and process them. The system emphasizes algorithmic thinking and practical data structure usage without reliance on external libraries.

# 2. Implemented Data Structures and Algorithms:

**Stack:** Used for undo functionality, allowing users to revert their last action of adding a product to the cart.  
**Queue:** Used to store and process orders in a First-In-First-Out (FIFO) manner.  
**Arrays:** Employed for managing the inventory, cart, and order lists efficiently.  
**Merge Sort:** Used to sort inventory products based on their IDs for fast lookup and display.

**3. Performance Analysis:**

The **Online Ordering System** is designed to work efficiently with small to moderately sized product inventories and order queues. For instance, as the system is built using arrays, we can directly access and modify data, which is very fast and helps in providing a smooth user experience.

* **Merge Sort (O(n log n) for Inventory Sorting):**  
  The system uses merge sort to keep the product inventory organized by their IDs. This is a well-known algorithm with a time complexity of O(n log n), meaning that it can handle large inventories efficiently without significant slowdowns, even as the number of products increases.
* **Array Access (O(1) for Data Retrieval):**  
  For managing the inventory, shopping cart, and order list, we use fixed-size arrays. One of the key benefits of using arrays is that they allow quick access to any product or order by simply indexing into the array. This makes the operations like adding or removing items very fast.
* **Stack and Queue Operations (O(1) for Push/Pop & Enqueue/Dequeue):**  
  The stack (used for undo functionality) and the queue (used for managing orders) are implemented to perform in constant time, O(1). This means no matter how many products or orders are handled, adding, removing, or retrieving items from the stack or queue will not slow down the system. Every operation happens instantly, ensuring that the user experience remains responsive even when processing multiple actions in quick succession.

# 4. Challenges Faced & Solutions:

**Challenge:** Implementing Undo without STL. **Solution:** Designed a custom stack with array-based storage.

**Challenge:** Managing inventory and avoiding duplicate IDs. **Solution**: Merge Sort and a product ID lookup function ensured product consistency.

**Challenge**: Efficiently removing items from the cart. **Solution:** Implemented in-place array shifting on undo.

# 5. Future Improvements:

* Add persistent storage (file I/O) for inventory and orders.
* Introduce user authentication and multiple user sessions
* Enhance UI using GUI frameworks or a web interface.
* Integrate real-time stock updates and live order tracking.

# 6. Course Concepts Utilization:

**1. Stacks and Queues:**  
We implemented a stack to manage the "Undo" feature in the shopping cart. Every time a user adds a product to the cart, it is pushed onto the stack. If the user decides to undo their last action, the system pops the last added product off the stack and removes it from the cart—just like the real-world "undo" button. Similarly, a queue was used to manage orders after checkout. This simulates a real-world order processing system, where the first order placed is the first to be processed (FIFO behavior).

**2. Sorting Algorithms (Merge Sort):**  
To keep the product inventory organized by product ID, we applied the mergesort algorithm. This ensures products are always displayed in a sorted order, which improves the user experience and also demonstrates efficient use of divide-and-conquer strategies taught in class.

**3. Arrays and Searching:**  
The entire system heavily relies on arrays for storing the inventory, shopping cart, and orders. Searching through these arrays (e.g., when looking up a product by ID) reflects our understanding of linear search techniques, as taught in data structures.

**4. Problem Decomposition and Modularity:**  
The project is designed in a modular way, where each part of the functionality (like adding products, displaying inventory, checking out, and processing orders) is handled in its own function or class. This makes the code easier to read**,** debug, and extend—a key principle emphasized in good software engineering and programming practices.

.