Activity-2: Case Based Learning (CBL) assessment sheet

Case Study 1: Choosing the Right Sorting Algorithm for Large-Scale Data Processing in E-Commerce

Course Name: Data Structures & Algorithms (DSA)									**Date: 21/09/2025**		
Topic: Choosing the Right Sorting Algorithm for Large-Scale Data Processing in E-Commerce											
Roll No.: 22379	**Name:** Sanskar Manoj Tiwari										
ABC ID:	**3**	**9**	**8**	**7**	**4**	**6**	**9**	**0**	**3**	**8**	**0**	**0**

Case Title: Sorting Algorithms for Product Listings

Case Summary

An e-commerce platform needs to sort large volumes of product listings based on criteria like price, ratings, and popularity. The efficiency of sorting algorithms depends on dataset size and the input order (sorted, reverse, or random).

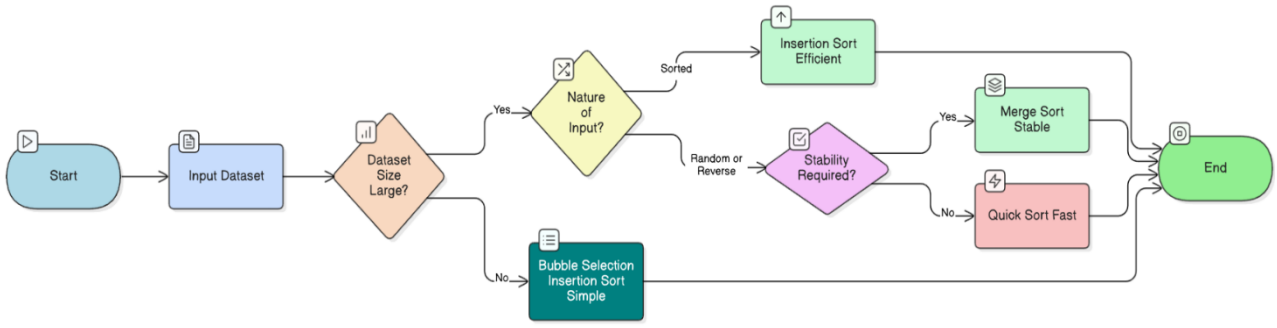
Bubble Sort – Simple, suitable for very small datasets; inefficient for large data ($O(n^2)$).

Selection Sort – Consistent $O(n^2)$ performance; minimizes swaps.

Insertion Sort – Efficient for small or nearly sorted datasets; adaptive to input order.

Merge Sort – Stable, guarantees O(n log n) performance; requires extra memory.

Quick Sort – Fast on average (O(n log n)), in-place; performance depends on pivot selection.



Flowchart: Decision Process for Choosing Sorting Algorithm

Student Reflection

Key challenges faced in algorithm selection:

- **Scalability & Performance:** The platform handles millions of product listings. Algorithms with $O(n^2)$ complexity, like Insertion Sort, are too slow. Efficient O(n log n) algorithms, such as Merge Sort, ensure fast loading and consistent performance, even for large datasets or pre-sorted/reverse-order inputs.

- **Stability Requirement:** Users often sort by multiple criteria (e.g., price, ratings, popularity). A **stable sort** preserves the relative order of items with equal keys, ensuring grouped products (like same brand or artist) remain consistent across consecutive sorts.

- **Memory Usage:** Large-scale sorting impacts memory. Out-of-place algorithms like Merge Sort require extra RAM (O(n)), which can increase server costs during peak loads, such as flash sales on major e-commerce platforms.

Algorithm	Time Complexity (Best)	Time Complexity (Average)	Time Complexity (Worst)	Space Complexity	Stability	Best Use Case
Bubble Sort	O(n)	$O(n^2)$	$O(n^2)$	O(1)	Stable	Teaching/demo, very small datasets
Insertion Sort	O(n)	$O(n^2)$	$O(n^2)$	O(1)	Stable	Small or nearly sorted datasets
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	O(1)	Unstable	Simple implementation, very small datasets
Merge Sort	O(n log n)	O(n log n)	O(n log n)	O(n)	Stable	Large-scale sorting, consistent performance
Quick Sort	O(n log n)	O(n log n)	$O(n^2)$	O(log n)	Unstable	Large random datasets, speed critical

Table: Comparison of Sorting Algorithms for E-Commerce Platform

Reason for choosing a specific sorting algorithm:

- For a commercial e-commerce platform, **Merge Sort** is the most robust and reliable choice. Its primary advantage is its **guaranteed O(nlogn) time complexity**. This provides predictable performance, ensuring the website remains responsive even when dealing with already sorted data, which is a common occurrence.

- Choosing Merge Sort is like an airline choosing a slightly slower but incredibly reliable aircraft model. While another model (like Quick Sort) might be faster under ideal conditions, Merge Sort's consistent performance prevents catastrophic delays (website timeouts) and ensures every "passenger" (user request) has a smooth journey.

- Furthermore, Merge Sort is a **stable sort**, which is critical for providing an intuitive multi-sorting experience, as described in the challenge above. The slight trade-off in memory usage is justified by the immense gain in reliability and user experience.

Real-World Example:

During Flipkart's "Big Billion Day" sale, millions of users sort products by "Lowest Price First." Merge Sort ensures consistent O(n log n) performance under heavy load, preventing timeouts and keeping the website responsive.

What could improve the solution's efficiency further?

- **Hybrid Sorting Algorithms:** Instead of using one single algorithm, a hybrid approach is often best. **Timsort**, the default algorithm used in Python and Java, is a prime example. It combines Merge Sort and Insertion Sort, making it extremely fast for real-world datasets, which are often partially sorted (e.g., a list of new arrivals is already mostly sorted by date).

- **Parallel Processing:** The work of sorting a massive dataset can be split among multiple processor cores. This is like having a team of librarians sorting books simultaneously in different aisles of a library instead of just one person doing everything. Merge Sort is naturally well-suited for this kind of parallelization, drastically cutting down processing time.

- **Data-Specific Algorithms:** For certain types of sorting, a specialized algorithm can be far superior. For example, if the platform wants to sort products into a few shipping categories (e.g., 'Same-Day Delivery', 'Standard', 'International'), a non-comparison algorithm like **Counting Sort** would be much faster than a general-purpose one like Merge Sort.

- **Leveraging the Database:** The most efficient solution is often to not sort in the application at all. Instead, the task is delegated to the database system (e.g., PostgreSQL). Databases are highly optimized for these operations and use powerful indexing systems (like B-Trees) to sort and retrieve data almost instantly. This is like asking a warehouse's inventory management system for a sorted list instead of manually sorting a paper manifest.

	Peer Review of Activity	Excellent	Good	Needs improvement	Remark
1.	**Case analysis depth**	☐	☐	☐	
2.	**Solution justification**	☐	☐	☐	
3.	**Peer discussion participation**	☐	☐	☐	

Rubrics wise marks

	Case-1		
Performance Indicator (PI)	**PI-1 [5]**	**PI-2 [5]**	**PI-3 [5]**
Marks			

Feedback/Comments:

Total Score: _____ / 15

_____		Teacher's Sign_____
Sign of student		Name of the teacher