**CASE STUDY**

Case Study 1: E-commerce Search Optimization

Scenario:  
An e-commerce platform like Amazon or Flipkart stores millions of products with details like Product ID, Name, Price, Category, and Ratings. Users frequently search for products by name, filter by price range, and sort by ratings. The system must support fast insertion, searching, and sorting of products.

Q1) Which search tree is suitable for implementing a product search system in an e-commerce database?

A B-tree or a Trie is most suitable for implementing a product search system. B-trees are commonly used in database indexing, while Tries are efficient for searching by product name, especially with prefix-based searches.

Q2) How does that tree help improve the efficiency of searching, filtering, and updating product listings?

* B-tree: Provides logarithmic search time (O(log n)) for insertion, deletion, and searching, making it ideal for database indexing and handling large datasets efficiently.
* Trie: Optimized for prefix searches, allowing quick product name suggestions.
* Both structures ensure balanced operations, reducing search time compared to unbalanced trees like BST.

Q3) What challenges arise when using BST or AVL trees for large-scale e-commerce applications?

* BST (Binary Search Tree): Can become unbalanced, leading to O(n) worst-case search time.
* AVL Tree: While always balanced, frequent rebalancing increases overhead, making it inefficient for high insertion rates in dynamic product databases.

Case Study 2: Banking System – Efficient Transaction Processing

Scenario:  
A bank needs a high-speed transaction processing system that efficiently searches, sorts, and updates millions of transactions based on timestamps.

Q1) Can we use the Red-Black tree in this case? Justify why it is preferable over AVL Tree.

Yes, a Red-Black tree is preferable because:

* It provides O(log n) search, insert, and delete operations.
* It maintains balance with fewer rotations than AVL trees, making insertions and deletions more efficient.
* Banking transactions require frequent inserts, and the Red-Black tree performs better than AVL in terms of insertion efficiency.

Q2) How does the self-balancing property of Red-Black Trees help ensure efficient transaction processing?

* Ensures that no path is more than twice as long as the shortest, keeping search times low.
* Reduces the number of rotations compared to AVL, making transaction updates (insert/delete) faster.
* Suitable for maintaining an ordered log of transactions, supporting efficient queries like finding recent transactions.

Case Study 3: AI and GPS System

Scenario:  
A ride-sharing app (Uber, Ola) must match customers with the nearest available driver in real-time. The app stores driver and passenger locations as (latitude, longitude) pairs and must find the nearest driver quickly.

Q1) Which tree is preferred for searching the nearest driver locations?

A k-d tree (k-dimensional tree) is preferred for nearest-neighbor searches in spatial data.

Q2) What are the limitations of this tree when dealing with real-time driver movement updates? How can these issues be mitigated?

* Limitation: K-d trees degrade in performance with frequent updates, as rebalancing is costly.
* Mitigation:
  + Use Quad-Trees or R-Trees, which are better suited for dynamic spatial data.
  + Implement approximate nearest neighbor (ANN) algorithms to improve real-time performance.
  + Use grid-based spatial partitioning for faster lookups in dense urban areas.