**Big O Notation:** Big O notation is a mathematical notation used to describe the upper bound of the time or space complexity of an algorithm. It characterizes an algorithm's efficiency as a function of the input size nnn, focusing on the growth rate of the runtime or memory usage as nnn increases. Big O notation helps in comparing different algorithms and understanding how they scale.

* **Best Case:** The minimum time an algorithm takes to complete, given the most favorable input.
* **Average Case:** The expected time an algorithm takes over all possible inputs, providing a realistic measure of performance.
* **Worst Case:** The maximum time an algorithm takes, given the least favorable input, ensuring the system's robustness.

**Time Complexity Comparison:**

* **Linear Search:**
  + Best Case: O(1) (if the target is the first element)
  + Average and Worst Case: O(n) (where nnn is the number of products)
  + Suitable for small datasets or unsorted data.
* **Binary Search:**
  + Best Case: O(1) (if the middle element is the target)
  + Average and Worst Case: O(log n) (requires the array to be sorted)
  + Efficient for large, sorted datasets.
* **Suitability:** For an e-commerce platform, binary search is generally more suitable due to its logarithmic time complexity, which scales well with large datasets. However, this requires that the product list be sorted by the search key (e.g., productName). If sorting is feasible and maintaining the sorted order is not overly costly, binary search provides much faster search capabilities compared to linear search.
* For unsorted data or cases where maintaining a sorted array is impractical, linear search might still be used, albeit with lower efficiency. Additionally, for more advanced scenarios, data structures like balanced binary search trees (e.g., AVL trees) or hash tables may be considered to combine efficient search with flexibility in insertion and deletion operations.