Q1.a) Explain Big O notation and how it helps in analyzing algorithms.
Ans: Big O notation is a mathematical way to describe how the performance of an algorithm changes as the size of the input grows. It focuses on the time complexity (execution time) or space complexity (memory usage) in terms of input size (n). Big O helps us understand the efficiency of an algorithm regardless of hardware or language.
Example:

- If an algorithm has O(n) time complexity, the time it takes grows linearly with the size of the input.
- If it's O(log n), the time increases very slowly even as input size grows large (e.g., binary search).

Q1.b) Describe the best, average, and worst-case scenarios for search operations.
Ans:

- Best Case The target is found immediately (first element checked).
- Average Case The target is found somewhere in the middle or position is random.
- Worst Case The target is not found, or found at the end after checking everything.

Q4.a) Compare the time complexity of linear and binary search algorithms.

Search Type	Best Case	Average Case	Worst Case
Linear Search	O(1)	O(n)	O(n)
Binary Search	O(1)	O(log n)	O(log n)

Q4.b) Discuss which algorithm is more suitable for your platform and why.
Ans: Binary Search is more suitable for our platform, and here's why:
Reasons:

- Faster Performance: Binary Search has O(log n) time complexity, making it much faster for large product catalogs.
- Scalability: As your platform grows to 10,000+ products, Linear Search becomes inefficient.
- Sorted Data Advantage: Most product lists (by ID, name, price) can be sorted easily and stored in memory.
- Better User Experience: Faster searches improve site responsiveness, leading to better customer satisfaction.