**Q1.a) Explain Big O notation and how it helps in analyzing algorithms.**

**Ans:** Big O notation is a mathematical representation that describes how an algorithm's resource usage (like time or memory) scales with the input size. It focuses on how an algorithm behaves as the input grows, helping us analyze its efficiency independent of hardware or programming language.  
**Example:**  
• An algorithm with O(n) time complexity means its execution time increases proportionally with input size.  
• O(log n) implies the algorithm becomes only slightly slower even with large inputs, such as in binary search.

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

**Ans:**  
• **Best Case:** The element is found in the first comparison itself.  
• **Average Case:** The element is located somewhere in the middle of the dataset or at a random position.  
• **Worst Case:** The element is either at the very end or not present at all, requiring all items to be checked.

**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 a better fit for our platform for several reasons:  
• **Speed:** It works in O(log n) time, making it significantly quicker for large datasets.  
• **Handles Scale:** It performs well even as the product database grows to thousands of items.  
• **Sorted Data Ready:** Product lists can be pre-sorted (by name, price, etc.), which is a key requirement for binary search.  
• **Improved User Experience:** Faster search times result in quicker page responses and better usability for customers.