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

Big O notation is a mathematical representation used to describe the efficiency of algorithms, specifically how their running time or space requirements grow relative to the size of the input (n).

How it helps in analyzing algorithms:

* It helps **analyze the efficiency** of algorithms.
* Allows you to **compare different approaches** regardless of hardware or language.
* Focuses on **scalability** — how performance changes as input grows.

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

Best Case**:** The scenario where the algorithm performs the minimum possible operations.

Average Case**:** The expected performance over a typical set of inputs.

Worst Case: The scenario where the algorithm performs the maximum possible operations.

1. Linear Search (on an unsorted array)

* Best Case:  
  The target element is the first item.  
  Time complexity: O(1)
* Average Case:  
  The target is in the middle or varies randomly.  
  Time complexity: O(n)
* Worst Case:  
   The target is the last item or not present at all.  
   Time complexity: O(n)

2. Binary Search (on a sorted array)

* Best Case:  
  The target is exactly at the middle of the array.  
  Time complexity: O(1)
* Average Case:  
  The search divides the array repeatedly in half.  
  Time complexity: O(log n)
* Worst Case:  
  The target is not present, or found after maximum divisions.  
  Time complexity: O(log n)

**Compare the time complexity of linear and binary search algorithms.**

Linear Search

* Best Case: O(1) — if the element is the first in the list.
* Average Case: O(n) — searches halfway through the list.
* Worst Case: O(n) — element is last or not found.
* Does not require a sorted array.
* Works on any unsorted list.
* Slower for large datasets.

Binary Search

* Best Case: O(1) — if the element is exactly in the middle.
* Average Case: O(log n) — list is divided in half at each step.
* Worst Case: O(log n) — element is not found after maximum splits.
* Requires a sorted array.
* Much faster and efficient for large datasets.
* Ideal when the data is pre-sorted by ID, name, or price.

**Discuss which algorithm is more suitable for your platform and why.**

For an e-commerce platform, binary search is more suitable than linear search. This is because product databases in such systems are typically large and are often sorted by product name, ID, or price. Binary search takes advantage of this sorted structure and provides much faster retrieval times, which is crucial for a responsive and efficient user experience. On the other hand, linear search would be too slow and inefficient, especially when customers are browsing through thousands of products. Therefore, although linear search is simple and works without sorting, binary search is the better choice for a platform that prioritizes speed, scalability, and performance.