1. **Understand Asymptotic Notation:**

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

Soln:

**Big O Notation:**

Big O notation is a mathematical notation used to describe the upper bound of an algorithm's running time or space complexity. It provides an asymptotic analysis of the performance, indicating how the running time or space requirements grow with the size of the input.

**How Big O Notation Helps**:

**Scalability**: It helps in understanding how an algorithm performs as the input size increases.

**Performance Comparison**: It provides a basis for comparing the efficiency of different algorithms.

**Optimization**: It aids in identifying bottlenecks and optimizing the critical parts of an algorithm.

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

Soln:

**Time Complexity:**

* **Linear Search**:
  + **Best Case**: O(1) (if the target is at the first position).
  + **Average Case**: O(n) (searching through half the array on average).
  + **Worst Case**: O(n) (if the target is at the end or not present).
* **Binary Search**:
  + **Best Case**: O(1) (if the target is at the middle).
  + **Average Case**: O(log n) (dividing the search space in half each time).
  + **Worst Case**: O(log n) (if the target is at the ends or not present).
* Compare the time complexity of linear and binary search algorithms.

**Linear Search**

**Time Complexity**:

* + **Best Case**: O(1) - The target element is at the first position.
  + **Average Case**: O(n)- The target element can be anywhere in the array.
  + **Worst Case**: O(n) - The target element is at the last position or not present in the array.

**Space Complexity**: O(1) - No additional space is required.

**Binary Search**

**Time Complexity**:

* + **Best Case**: O(1) - The target element is at the middle position.
  + **Average Case**: O(log n)- The array is divided into halves repeatedly.
  + **Worst Case**: O(log n) - The target element is at the last position of the divided segments.

**Space Complexity**:

* + **Iterative Implementation**: O(1) - No additional space is required.
* Discuss which algorithm is more suitable for your platform and why.

Soln:

**Binary Search** is more suitable for an e-commerce platform for the following reasons:

1. **Large Datasets**: E-commerce platforms typically have a large number of products. The logarithmic time complexity of binary search ensures that search operations remain fast, even as the number of products grows.
2. **Frequent Searches**: Users frequently search for products. Fast search times improve the user experience, making the platform more responsive.
3. **Static or Sorted Data**: Product catalogs on e-commerce platforms are often sorted by product IDs or other attributes. This sorting can be maintained easily, making binary search an ideal choice.

* Compare the performance (time complexity) of Bubble Sort and Quick Sort.

**Bubble Sort**:

**Best Case**: O(n) (when already sorted).

**Average Case**: O(n²).

**Worst Case**: O(n²).

**Quick Sort**:

**Best Case**: O(n log n).

**Average Case**: O(n log n).

**Worst Case**: O(n²) (with poor pivot choices).

* Discuss why Quick Sort is generally preferred over Bubble Sort.

**Quick Sort** is generally preferred for larger datasets due to its better average and worst-case performance compared to **Bubble Sort**.