**E-COMMERCE PLATFORM SEARCH FUNCTION**

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

**Big-O**, commonly referred to as “**Order of**”, is a way to express the **upper bound**of an algorithm’s time complexity, since it analyses the**worst-case** situation of algorithm. It provides an**upper limit** on the time taken by an algorithm in terms of the size of the input. It’s denoted as**O(f(n))**, where**f(n)** is a function that represents the number of operations (steps) that an algorithm performs to solve a problem of size **n**.

For example:

* O(1): Constant time complexity, where the algorithm's runtime remains constant regardless of the input size.
* O(log n): Logarithmic time complexity, where the algorithm's runtime grows logarithmically with the input size.
* O(n): Linear time complexity, where the algorithm's runtime grows linearly with the input size.
* O(n log n): Linearithmic time complexity, commonly seen in efficient sorting algorithms like mergesort and heapsort.
* O(n^2): Quadratic time complexity, where the algorithm's runtime grows quadratically with the input size.

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

**Best-case Scenario:** The scenario where the algorithm performs the minimum number of steps. For search operations, the best case is usually when the target element is the first element in the dataset.

**Average-case Scenario:** The scenario representing the expected number of steps an algorithm will take for a typical input. For search operations, it assumes the target element could be anywhere in the dataset.

**Worst-case Scenario:** The scenario where the algorithm performs the maximum number of steps. For search operations, it occurs when the target element is the last element or not present in the dataset.

**Time Complexity Analysis:**

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

**Linear Search:**

Best-case: O(1)

Average-case: O(n)

Worst-case: O(n)

**Binary Search:**

Best-case: O(1)

Average-case: O(log n)

Worst-case: O(log n)

Comparison:

**Linear Search** is simple and doesn't require the array to be sorted. It's suitable for small datasets or unsorted arrays.

**Binary Search** requires a sorted array but significantly reduces the number of comparisons to O(log n)which makes it efficient for large datasets.

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

**Binary search** is faster than regular search for finding products in a big online store. Because it can check half the products each time, it's really quick.

Plus, online stores often sort products by price or popularity, which makes binary search even better.