**Exercise 2: E-commerce Platform Search Function**

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

A. Defining an algorithm’s **worst-case** time complexity by using the Big-O notation, which determines the set of functions grows slower than or at the same rate as the expression. Furthermore, it explains the maximum amount of time an algorithm requires to consider all input values.

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

A. **1. Worst Case Analysis (Mostly used) :** In the worst-case analysis, we calculate the upper bound on the running time of an algorithm. We must know the case that causes a maximum number of operations to be executed. For Linear Search, the worst case happens when the element to be searched (x) is not present in the array. When x is not present, the search()function compares it with all the elements of arr[] one by one. Therefore, the worst-case time complexity of the linear search would be **O(n)**.

**2. Best Case Analysis (Very Rarely used) :** In the best-case analysis, we calculate the lower bound on the running time of an algorithm. We must know the case that causes a minimum number of operations to be executed. In the linear search problem, the best case occurs when x is present at the first location. The number of operations in the best case is constant (not dependent on n). So time complexity in the best case would be O(1).

**3. Average Case Analysis (Rarely used) :** In average case analysis, we take all possible inputs and calculate the computing time for all of the inputs. Sum all the calculated values and divide the sum by the total number of inputs. We must know (or predict) the distribution of cases. For the linear search problem, let us assume that all cases are uniformly distributed (including the case of x not being present in the array). So we sum all the cases and divide the sum by (n+1). Following is the value of average-case time complexity.

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

A. Linear Search-**Time Complexity:** O(n), where n is the size of the input array. The worst-case scenario is when the target element is not present in the array, and the function has to go through the entire array to figure that out.

Binary Search-**Time Complexity:** O(log n) – Binary search algorithm divides the input array in half at every step, reducing the search space by half, and hence has a time complexity of logarithmic order.

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

A. For an e-commerce platform with a large inventory and sorted product listings, **binary search** is more suitable due to its efficiency in handling large datasets.

* + Best for **large datasets** that are **sorted**.
  + Efficient for **frequent lookups** as it significantly reduces the number of comparisons needed by halving the search space with each step.
  + For an e-commerce platform with a large and **sorted inventory**, binary search is more efficient due to its **O(log n)** time complexity.
  + Reduces search time drastically compared to linear search’s **O(n)** time complexity, especially as the number of products grows.
  + **Pre-sorting** of data (which can be done periodically) allows binary search to leverage its efficient searching capabilities, making it ideal for handling large catalogs efficiently.