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

**1. Asymptotic Notation**

Asymptotic notation is a mathematical concept used to describe the running time or space requirements of an algorithm relative to the size of its input. It helps in analyzing algorithms by abstracting away constants and lower-order terms.

Big O notation is a commonly used asymptotic notation, representing the worst-case complexity of an algorithm. It is used to find an upper bound on time taken by an algorithm or data structure.

Best, average, and worst-case scenarios describe different performance outcomes of search operations:

• Best Case: Best case will be when the desired element is found at the beginning of the search.

• Average Case: When the element is found somewhere in the middle of the dataset that case will be called as average case.

• Worst Case: The worst case will be when the element is either not present or is present at the last place, requiring traversal of the entire dataset.

Example: f(n) = 3n² +2n + 1000Logn + 5000  
- After ignoring lower order terms, we get the highest order term as 3n²  
- After ignoring the constant 3, we get n²  
- Therefore the Big O value of this expression is O(n²)

Here’s a brief overview of Big O notation:

* O(1): Constant time complexity. The runtime or space requirement is fixed and does not change with the input size. For example, accessing an element in an array by index.
* O(log n): Logarithmic time complexity. The runtime or space requirement grows logarithmically with the input size. For example, binary search in a sorted array.
* O(n): Linear time complexity. The runtime or space requirement grows linearly with the input size. For example, linear search in an unsorted array.
* O(nlogn): Linearithmic time complexity. The runtime or space requirement grows proportionally to \( n \log n \). For example, efficient sorting algorithms like mergesort and heapsort.
* O(n²): Quadratic time complexity. The runtime or space requirement grows quadratically with the input size. For example, bubble sort or insertion sort.

**2. How Big O notation helps in analyzing Algorithms**

Big O notation helps analyze algorithms by providing a standardized way to describe their efficiency in terms of time and space as input size grows. It allows developers to predict how an algorithm's performance will scale, compare different algorithms for the same task, and identify potential bottlenecks. By focusing on the dominant factors affecting runtime or memory usage, Big O helps optimize algorithms for better scalability and ensures they perform efficiently across various input sizes.

**3. Search Operations**

1. Linear Search:

* Best Case:

The target element is found at the first position.

Time Complexity: O(1).

* Average Case:

The target element is found somewhere in the middle of the array.

Time Complexity: O(n/2), which simplifies to O(n).

* Worst Case:

The target element is the last element or not present at all, requiring traversal of the entire array.

Time Complexity: O(n).

**2.** Binary Search:

* Best Case:

The target element is the middle element of the array in the first step.

Time Complexity: O(1).

* Average Case:

The target element is found after repeatedly dividing the array and narrowing down the search space.

Time Complexity: O(logn), as the array is divided in half with each step.

* Worst Case:

The target element is not present, requiring the algorithm to narrow down the search space completely.

Time Complexity: O(logn).

**Algorithm Suitability**

For E-commerce platform function:

* **Binary Search** can be used for this platform as it is more suitable because it efficiently handles large datasets and frequent search queries.
* **Linear Search** can be considered only for small datasets or unsorted data when the overhead of sorting is undesirable.