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

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

Big O notation is how computer scientists represent the efficiency of a computer algorithm in terms of time and space, based on the size of an algorithm’s input set. It gives the upper limit, which helps to evaluate how algorithms scale which can be used in comparing multiple algorithms.

* O(1) — Constant time: The performance remains constant no matter the input size.

Example: Accessing an element of an array using an index.

* O(log n) — Logarithmic time: As the input decreases, growth slows.

Example: Binary search.

* O(n) — Linear time: The running time grows with size of input.

Example: Traversing an unsorted list.

* O(n log n) — Linearithmic time: A mix of linear and logarithmic time.

Example: Mergesort and Heapsort.

* O(n²) — Quadratic time: Performance degrades very quickly with even minute changes in the input.

Example: Nested loops like in bubble sort

* O(2ⁿ) — Exponential time: Grows extremely fast and is thus inefficient for large inputs.

Example

* O(n!) — Factorial time: One of the slowest growth rates.

Example: Checking all possible permutations

Big O helps to compare algorithms and data structures based on the time and space complexity. It is used to select the most efficient approach for solving a problem.

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

Linear Search:

* The Best Case: O(1), when the array's first element is the target element.
* The Worst Case: O(n) When the array's last element is the target element or if the element does not exist in the array.
* Average-case: O(n). On average, the search will need to check half of the elements in the list.

Binary Search:

* The Best Case: O(1), when the array's middle element is the target element.
* The Worst Case: O(logn) when the array does not have the target element or is far away from the middle element.
* Average-case: O(log n). The search repeatedly divides the array in half.

When one sort is worth the speedup for repeated lookups, binary search is much more efficient, particularly as the list grows. Rather than examining each item individually, it reduces the search space severely with every comparison.  
  
But where we have a short list or occasional searches where sorting seems to decrease the efficiency, linear search gets the job done with less overhead.