Solution to Understanding Asymptotic Notation

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

Ans. Big O notation is a mathematical notation used in computer science to describe the upper bound or worst-case scenario of the runtime complexity of an algorithm in terms of the input size. It provides a standardized and concise way to express how the performance of an algorithm scales as the size of the input grows.

Understanding Big O Notation: In Big O notation, "O" represents the order of the function, and "f(n)" represents the function describing the algorithm's time complexity in terms of the input size "n." The notation "O(f(n))" signifies that the algorithm's time complexity grows no faster than a specific function of "n." Here, "f(n)" is a mathematical function describing how the algorithm's runtime increases as the input size grows.

For example: O(log n): Logarithmic time complexity, where the algorithm's runtime grows logarithmically with the input size.

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

Ans. Linear Search

Best Case: O(1) - The element is the first one.

Average Case: O(n) - The element is somewhere in the middle.

Worst Case: O(n) - The element is the last one or not present.

Binary Search (requires sorted array)

Best Case: O(1) - The element is the middle one.

Average Case: O(log n) - The element is somewhere in the array, requiring repeated halving.

Worst Case: O(log n) - The element is not present, requiring repeated halving until the array is exhausted.

Interpolation Search (Sorted and Uniformly Distributed List)

Best Case: O(1) - The target element is found in the first probe.

Average Case: O(log log n) - The target element is found after a few probes due to uniform distribution.

Worst Case: O(n) - The distribution is non-uniform, making the search degrade to linear search.

The most efficient way to search an element in a sorted array is binary search and in an unsorted array is linear search.