DAA CASE STUDY

Scenario:
An e-commerce platform is implementing a feature where products need to be sorted by various attributes (e.g., price, rating, and name). The product list contains millions of items, and the sorting operation needs to be efficient and scalable.
1. What are the time and space complexities of the commonly used sorting algorithms (Quick Sort, Merge Sort)?
2. How do the characteristics of the data (e.g., range of prices, product name lengths) impact the choice of sorting algorithm?

1. Time and Space Complexities of Common Sorting Algorithms

Algorithm	Time Complexity (Best)	Time Complexity (Average)	Time Complexity (Worst)	Space Complexity
Quick Sort	O(nlogn)	O(nlogn)	O(n2) (unbalanced pivot)	O(logn)auxiliary (in-place)
Merge Sort	O(nlogn)	O(nlogn)	O(nlogn)	O(n)auxiliary

Key Points:

- **Quick Sort** is efficient in practice due to low constant factors and its in-place nature but suffers from poor performance if the pivot is not chosen wisely.
- **Merge Sort** guarantees O(nlog⁡n)O(n \log n)O(nlogn) performance regardless of data distribution but requires additional memory for merging.

---

2. Impact of Data Characteristics on Sorting Algorithm Choice

Range of Prices or Numerical Data:

- For a **small range** of values (e.g., product prices), counting sort or radix sort may be more efficient than comparison-based algorithms, achieving linear time complexity O(n+k)O(n + k)O(n+k), where kkk is the range of numbers.
- For a **large or arbitrary range**, comparison-based algorithms like Quick Sort or Merge Sort are preferred.

Product Name Lengths (String Sorting):

- Lexicographic sorting involves comparing strings character by character. Algorithms like Quick Sort and Merge Sort still perform well but may be slower due to increased comparison times.
- Radix sort can be used for fixed-length strings or cases where the length is bounded and known, providing near-linear time complexity.

Distribution and Size of Data:

- **Uniformly Distributed Data**: Quick Sort performs well with randomized pivot selection strategies.
- **Nearly Sorted Data**: Insertion Sort or TimSort (used by Python's sorted() and Java's Arrays.sort()) may outperform others due to better handling of such inputs.

Stability Requirements:Merge Sort is stable, meaning equal elements retain their relative order. Quick Sort is generally not stable unless modified.

Memory Constraints:If memory usage is a concern, Quick Sort (with in-place sorting) is often more suitable than Merge Sort.

DONE BY:
D.Ramya Sri
2211CS020127
AIML-Beta