**Exercise 3: Sorting Customer Orders**

**Scenario:**

You are tasked with sorting customer orders by their total price on an e-commerce platform. This helps in prioritizing high-value orders.

**1. Understand Sorting Algorithms**

**Bubble Sort**

* Repeatedly swaps adjacent elements if they’re in the wrong order.
* **Time Complexity**:
  + Best: O(n) (already sorted with optimization)
  + Avg/Worst: O(n²)
* Simple but inefficient for large data.

**Insertion Sort**

* Builds the sorted list one item at a time.
* **Time Complexity**:
  + Best: O(n),
  + Avg/Worst: O(n²)
* Efficient for small or nearly sorted datasets.

**Merge Sort**

* Divide and conquer approach; divides the array and merges sorted halves.
* **Time Complexity**: O(n log n) for all cases.
* Requires extra space (not in-place).

**Quick Sort (Preferred)**

* Picks a pivot and partitions the array around it.
* **Time Complexity**:
  + Best/Average: O(n log n)
  + Worst: O(n²) (rare if pivot is well-chosen)
* **In-place** and fast in practice.

**5. Analysis**

**Time Complexity Comparison**

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithm** | **Best** | **Average** | **Worst** |
| Bubble Sort | O(n) | O(n²) | O(n²) |
| Quick Sort | O(n log n) | O(n log n) | O(n²)\* (rare) |

Quick Sort is **significantly faster** than Bubble Sort in most practical scenarios, especially with large datasets.

**Summary:**

* Bubble Sort is easy but inefficient (O(n²)).
* Quick Sort is much faster (O(n log n) on average), making it suitable for e-commerce systems handling many orders.
* Quick Sort is **generally preferred** due to its speed and in-place nature.