**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.

**Steps:**

1. **Understand Sorting Algorithms:**
   * Explain different sorting algorithms (Bubble Sort, Insertion Sort, Quick Sort, Merge Sort).

Solution:

**Sorting Algorithms Overview**

1. **Bubble Sort**:
   * **Description**: Repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. This process is repeated until the list is sorted.
   * **Time Complexity**:
     + Best Case: O(n)O(n)O(n) (if the list is already sorted)
     + Average Case: O(n2)O(n^2)O(n2)
     + Worst Case: O(n2)O(n^2)O(n2)
   * **Usage**: Simple but inefficient for large datasets.
2. **Insertion Sort**:
   * **Description**: Builds the final sorted list one item at a time by repeatedly taking an unsorted item and inserting it into its correct position within the sorted portion.
   * **Time Complexity**:
     + Best Case: O(n)O(n)O(n) (if the list is already sorted)
     + Average Case: O(n2)O(n^2)O(n2)
     + Worst Case: O(n2)O(n^2)O(n2)
   * **Usage**: More efficient than Bubble Sort for small datasets or nearly sorted data.
3. **Quick Sort**:
   * **Description**: Divides the list into smaller sub-lists based on a pivot element and recursively sorts the sub-lists. The process is repeated until the entire list is sorted.
   * **Time Complexity**:
     + Best Case: O(nlog⁡n)O(n \log n)O(nlogn)
     + Average Case: O(nlog⁡n)O(n \log n)O(nlogn)
     + Worst Case: O(n2)O(n^2)O(n2) (can be mitigated with good pivot selection)
   * **Usage**: Efficient for large datasets, widely used in practice.
4. **Merge Sort**:
   * **Description**: Divides the list into two halves, recursively sorts each half, and then merges the sorted halves.
   * **Time Complexity**:
     + Best Case: O(nlog⁡n)O(n \log n)O(nlogn)
     + Average Case: O(nlog⁡n)O(n \log n)O(nlogn)
     + Worst Case: O(nlog⁡n)O(n \log n)O(nlogn)
5. **Setup:**
   * Create a class **Order** with attributes like **orderId**, **customerName**, and **totalPrice**.
6. **Implementation:**
   * Implement **Bubble Sort** to sort orders by **totalPrice**.
   * Implement **Quick Sort** to sort orders by **totalPrice**.

Solution:

Setup and Implementation part is in the attached java file.

1. **Analysis:**
   * Compare the performance (time complexity) of Bubble Sort and Quick Sort.

Solution:

Time Complexity Comparison

* Bubble Sort: O(n2)O(n^2)O(n2) in all cases, which is inefficient for large datasets.
* Quick Sort: Average time complexity of O(nlog⁡n)O(n \log n)O(nlogn), making it much more efficient for large datasets compared to Bubble Sort.
  + Discuss why Quick Sort is generally preferred over Bubble Sort.

Solution:

**Efficiency**: Quick Sort is significantly faster for large datasets due to its O(nlog⁡n)O(n \log n)O(nlogn) average time complexity.

**Partitioning**: Quick Sort's divide-and-conquer approach reduces the problem size with each recursive call, leading to faster sorting.

**Practical Use**: Quick Sort is often preferred in practical applications and library implementations due to its efficiency and adaptability