**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).

**Bubble Sort**

Bubble Sort is a simple comparison-based sorting algorithm. It 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) - The list is already sorted.
  + Average Case: O(n^2) - The list is randomly ordered.
  + Worst Case: O(n^2) - The list is sorted in reverse order.

**Insertion Sort**

Insertion Sort is a simple comparison-based sorting algorithm. It builds the sorted list one item at a time by repeatedly taking the next item and inserting it into its correct position.

* **Time Complexity**:
  + Best Case: O(n) - The list is already sorted.
  + Average Case: O(n^2) - The list is randomly ordered.
  + Worst Case: O(n^2) - The list is sorted in reverse order.

**Quick Sort**

Quick Sort is an efficient comparison-based sorting algorithm. It works by selecting a 'pivot' element and partitioning the array into two sub-arrays, according to whether the elements are less than or greater than the pivot. The sub-arrays are then sorted recursively.

* **Time Complexity**:
  + Best Case: O(n log n) - The list is evenly divided.
  + Average Case: O(n log n) - The list is randomly ordered.
  + Worst Case: O(n^2) - The list is already sorted or nearly sorted, leading to unbalanced partitions (can be mitigated with good pivot selection).

**Merge Sort**

**ggg**Merge Sort is a stable, comparison-based sorting algorithm. It divides the list into two halves, recursively sorts each half, and then merges the sorted halves back together.

* **Time Complexity**:
  + Best Case: O(n log n) - The list is evenly divided.
  + Average Case: O(n log n) - The list is randomly ordered.
  + Worst Case: O(n log n) - The list is sorted in reverse order.

1. **Setup:**

A project is named as OrderSorting.

1. **Implementation:**

The implementation code is provided in the wordpad.

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

**Time Complexity Comparison**

* **Bubble Sort**:
  + Best Case: O(n) - The list is already sorted.
  + Average Case: O(n^2) - The list is randomly ordered.
  + Worst Case: O(n^2) - The list is sorted in reverse order.
* **Quick Sort**:
  + Best Case: O(n log n) - The list is evenly divided.
  + Average Case: O(n log n) - The list is randomly ordered.
  + Worst Case: O(n^2) - The list is already sorted or nearly sorted, leading to unbalanced partitions (can be mitigated with good pivot selection).
    1. **Discuss why Quick Sort is generally preferred over Bubble Sort.**

**Performance**: Quick Sort has a significantly better average-case time complexity (O(n log n)) compared to Bubble Sort (O(n^2)).

**Efficiency**: Quick Sort is more efficient for larger datasets due to its divide-and-conquer approach.

**Practicality**: Despite its worst-case scenario being O(n^2), Quick Sort can be optimized with good pivot selection (e.g., using median-of-three or random pivot), making it very efficient in practice.

While Bubble Sort is easier to implement and understand, its quadratic time complexity makes it impractical for large datasets, making Quick Sort a more suitable choice for sorting customer orders by total price on an e-commerce platform.