**Exercise 2: 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.

**Understanding Sorting Algorithms:**

 **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**: O(n^2) for the worst and average cases, O(n) for the best case (when the list is already sorted).

 **Insertion Sort:**

* **Description**: Builds the final sorted array one item at a time. It picks the next item and inserts it into the correct position within the already sorted part of the array.
* **Time Complexity**: O(n^2) for the worst and average cases, O(n) for the best case (when the list is already sorted).

 **Quick Sort:**

* **Description**: An efficient divide-and-conquer algorithm. It picks a "pivot" element and partitions the array into two halves, then recursively sorts the sub-arrays.
* **Time Complexity**: O(n log n) for the average case, O(n^2) for the worst case (which can be mitigated with good pivot selection).

 **Merge Sort:**

* **Description**: Another divide-and-conquer algorithm that divides the array into two halves, sorts each half, and then merges the sorted halves.
* **Time Complexity**: O(n log n) for all cases, as it always divides the array in half and merges sorted arrays.

**Setup:**

I have used Visual Studio to create a new project with the name Sorting Customer Orders.

**Implementation:**

I used both Bubble Sort and Quick Sort. Output obtained after using the both methods is of course same.

**Analysis:**

* **Bubble Sort**:
  + **Best Case**: O(n) - When the array is already sorted.
  + **Average Case**: O(n^2) - The average number of swaps and comparisons.
  + **Worst Case**: O(n^2) - When the array is sorted in reverse order.
* **Quick Sort**:
  + **Best Case**: O(n log n) - When the pivot divides the array into two equal halves.
  + **Average Case**: O(n log n) - Average scenario with good pivot selection.
  + **Worst Case**: O(n^2) - When the pivot selection is poor (e.g., always choosing the largest or smallest element), which can be mitigated with good pivot strategies like random pivoting.

**Why Quick Sort is generally preferred over Bubble Sort:**

Quick Sort is generally preferred over Bubble Sort because it is much more efficient for large datasets. While Bubble Sort has a time complexity of O(n^2) in the average and worst cases, Quick Sort has an average time complexity of O(n log n), making it significantly faster for large arrays. Additionally, Quick Sort's divide-and-conquer approach is more adaptable and can be optimized further with techniques like random pivoting to avoid worst-case scenarios.