**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).
2. **Setup:**
   * Create a class **Order** with attributes like **orderId**, **customerName**, and **totalPrice**.
3. **Implementation:**
   * Implement **Bubble Sort** to sort orders by **totalPrice**.
   * Implement **Quick Sort** to sort orders by **totalPrice**.
4. **Analysis:**
   * Compare the performance (time complexity) of Bubble Sort and Quick Sort.
   * Discuss why Quick Sort is generally preferred over Bubble Sort.

**CODE:**

using System;

class Order

{

public int OrderId { get; set; }

public string CustomerName { get; set; }

public decimal TotalPrice { get; set; }

public Order(int orderId, string customerName, decimal totalPrice)

{

OrderId = orderId;

CustomerName = customerName;

TotalPrice = totalPrice;

}

}

class SortOrders

{

//Bubble Sort

public static void BubbleSort(Order[] orders)

{

for (int i = 0; i < orders.Length - 1; i++)

{

for (int j = 0; j < orders.Length - i - 1; j++)

{

if (orders[j].TotalPrice > orders[j + 1].TotalPrice)

{

var temp = orders[j];

orders[j] = orders[j + 1];

orders[j + 1] = temp;

}

}

}

}

//Quick Sort

public static void QuickSort(Order[] orders, int low, int high)

{

if (low < high)

{

int pivot = Partition(orders, low, high);

QuickSort(orders, low, pivot - 1);

QuickSort(orders, pivot + 1, high);

}

}

private static int Partition(Order[] orders, int low, int high)

{

decimal pivot = orders[high].TotalPrice;

int i = low - 1;

for (int j = low; j < high; j++)

{

if (orders[j].TotalPrice < pivot)

{

i++;

var temp = orders[i];

orders[i] = orders[j];

orders[j] = temp;

}

}

var temp2 = orders[i + 1];

orders[i + 1] = orders[high];

orders[high] = temp2;

return i + 1;

}

public static void PrintOrders(Order[] orders)

{

foreach (var order in orders)

Console.WriteLine($"{order.OrderId}: {order.CustomerName} - ${order.TotalPrice}");

}

}

class Program3

{

static void Main()

{

Order[] orders = {

new Order(1, "Alice", 2500),

new Order(2, "Bob", 1800),

new Order(3, "Charlie", 3200),

new Order(4, "David", 1500)

};

Console.WriteLine("Bubble Sort:");

var bubbleSorted = (Order[])orders.Clone();

SortOrders.BubbleSort(bubbleSorted);

SortOrders.PrintOrders(bubbleSorted);

Console.WriteLine("\nQuick Sort:");

var quickSorted = (Order[])orders.Clone();

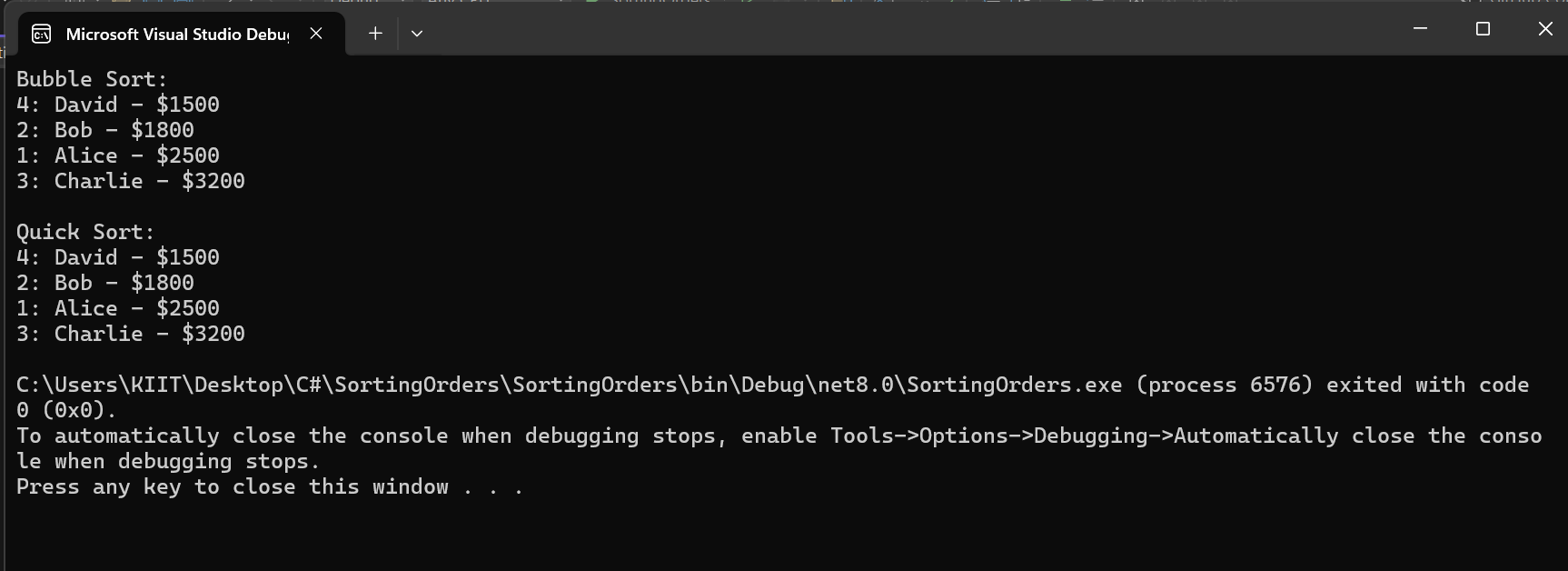
SortOrders.QuickSort(quickSorted, 0, quickSorted.Length - 1);

SortOrders.PrintOrders(quickSorted);

}

}

**OUTPUT:**



**ANALYSIS:**

****Time Complexity:****

* **Bubble Sort**: O(n²)
* **Quick Sort**: O(n log n) average, O(n²) worst

**Why prefer Quick Sort:**  
It’s faster on average and uses less memory than Merge Sort.