# **Empirical Analysis of Sorting Algorithms**

## **Introduction**

For this assignment, I developed an Order Management System that can sort, search, and prioritize customer orders using different algorithms. The main goal was to compare three sorting algorithms—QuickSort, MergeSort, and a built-in and analyze their efficiency and suitability for sorting order objects in C#.

## **Sorting Algorithms Implemented**

**-QuickSort (by Order ID):** I implemented QuickSort to sort the orders by their unique ID in ascending order. The algorithm uses the leftmost element as the pivot. QuickSort is known for being efficient on average but can have a worst-case scenario if the pivot choices are bad.

**-MergeSort (by Date Placed):** MergeSort was used to sort orders by the date they were placed, with the most recent first. MergeSort always splits the list in half, sorts each half, and then merges the results. It’s very reliable and gives consistent performance, but it uses extra memory for the merging step.

**-CustomSort (by Delivery Date):** For the third sort, I used C#’s built-in sorting functionality to sort by delivery date, again with the most recent first. Under the hood, this uses a hybrid algorithm that’s very fast and stable for objects.

## **Algorithm Analysis and Comparison**

### **QuickSort**

* **Best Case:** O(n log n)
* **Average Case:** O(n log n)
* **Worst Case:** O(n²)

**Strengths:**QuickSort is fast in most situations and doesn’t use much extra memory.

**Weaknesses:**Its speed can drop if the data is already nearly sorted, especially with bad pivot choices.

**MergeSort**

* **Best Case:** O(n log n)
* **Average Case:** O(n log n)
* **Worst Case:** O(n log n)

**Strengths:**MergeSort is very consistent and always performs well, no matter how the data looks. It’s also stable, meaning equal items keep their original order.

**Weaknesses:**It needs extra space for the merging process, which can be a drawback for very large datasets.

### **CustomSort (Built-in .NET Sort)**

* **Best Case:** O(n log n)
* **Average Case:** O(n log n)
* **Worst Case:** O(n log n)

This uses .NET’s built-in sort, which typically uses a hybrid algorithm called IntroSort. It’s optimized for objects and handles all cases efficiently.

**Strengths:**Very fast, stable, and already highly optimized by Microsoft.

**Weaknesses:**None significant for typical business data.

## **Which Algorithm Is Best for Sorting Orders?**

While all three algorithms will get the job done, I think using the built-in sort method is the best option for real applications. It’s extremely reliable, easy to use, and you don’t have to worry about worst-case scenarios or bugs in custom sorting code. MergeSort is also great if you need consistent performance and don’t mind using more memory. QuickSort is fine if implemented with good pivot logic, but you have to be careful to avoid bad cases.