Overview of Sorting Algorithms

Hossein Javidnia

1 Introduction

This document provides an overview of various sorting algorithms, including their characteristics in terms of stability, whether they are in-place, and their typical applications.

2 Sorting Algorithms

2.1 Merge-Sort

Description: Efficient for large datasets with a consistent time complexity of $O(nlogn)$. It is a divide-and-conquer algorithm.
In-Place: No ⚠
Stable: Yes ✅
Applications: Used in scenarios where stable sort is necessary, such as database algorithms and in sorting linked lists.

2.2 Quick-Sort

Description: Offers $O(nlogn)$ average time complexity and is generally faster than other $O(nlogn)$ algorithms. It works by selecting a 'pivot' element and partitioning the array around it.
In-Place: Yes ✅
Stable: No ⚠
Applications: Commonly used in systems where quick, efficient sorting is advantageous, like in various programming libraries and for large datasets.

2.3 Bucket Sort

Description: Efficient when input is uniformly distributed over a range. It distributes elements into buckets and then sorts these buckets individually.
In-Place: No ⚠
Stable: Yes ✅
Applications: Suitable for sorting data with floating point numbers and for scenarios where data distribution is known and uniform.

2.4 Insertion Sort

Description: Highly efficient for small datasets and simple to implement. It builds the final sorted array one item at a time.
In-Place: Yes ✅
Stable: Yes ✅
Applications: Ideal for small lists, as an introductory algorithm in teaching, and as the recursive base case for more complex sorts like Merge-Sort.

2.5 Selection Sort

Description: Simple algorithm that sorts an array by repeatedly finding the minimum element and putting it at the beginning.
In-Place: Yes ✅
Stable: No ⚠
Applications: Used in scenarios where memory writes are costly operations, such as in systems with limited write cycles.

2.6 Bubble Sort

Description: Simple sorting algorithm that repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order.
In-Place: Yes ✅
Stable: Yes ✅
Applications: Mainly used for educational purposes to teach sorting algorithms and for small, nearly sorted datasets.

2.7 Counting Sort

Description: Efficient for sorting small integers. It counts the number of occurrences of each value and uses this information to place each element in its correct position.
In-Place: No ⚠
Stable: Yes ✅
Applications: Best for scenarios where the range of data is not significantly greater than the number of objects to be sorted, such as sorting characters in a string.

2.8 Radix Sort

Description: Efficient for large datasets of integers or strings. It processes each digit of the numbers, starting from the least significant digit.
In-Place: No ⚠
Stable: Yes ✅
Applications: Useful for sorting large sets of data, such as phone numbers, or when the key size is small compared to the number of items.