Sorting Algorithms: Comparison and Analysis

Introduction

Sorting algorithms are fundamental in computer science for arranging data in a specific order. Different algorithms have varying performance characteristics depending on input size and nature.

Common Sorting Algorithms

Algorithm	Type	Best Case	Average Case	Worst Case	Space Complexity	Stable?
Bubble Sort	Comparison	O(n)	$O(n^2)$	$O(n^2)$	O(1)	Yes
Selection Sort	Comparison	$O(n^2)$	$O(n^2)$	$O(n^2)$	O(1)	No
Insertion Sort	Comparison	O(n)	$O(n^2)$	$O(n^2)$	O(1)	Yes
Merge Sort	Comparison / Divide & Conquer	O(n log n)	O(n log n)	O(n log n)	O(n)	Yes
Quick Sort	Comparison / Divide & Conquer	O(n log n)	O(n log n)	$O(n^2)$	O(log n)	No
Heap Sort	Comparison / Heap	O(n log n)	O(n log n)	O(n log n)	O(1)	No
Counting Sort	Non-comparison	O(n+k)	O(n+k)	O(n+k)	O(k)	Yes
Radix Sort	Non-comparison	O(nk)	O(nk)	O(nk)	O(n+k)	Yes

Note: n = number of elements, k = range of input values

Time Complexity Comparison

Sorting algorithms can be visualized in terms of time complexity, where array size (n) is on the x-axis and runtime is on the y-axis.

Observations

- **Bubble, Selection, Insertion:** Quadratic time; inefficient for large arrays.
- **Merge, Quick, Heap:** Linearithmic time; efficient for large arrays.
- **Counting, Radix:** Nearly linear time if k is small; suitable for integer keys.

Conclusion

Choosing a sorting algorithm depends on input size, data type, and stability requirements. For small arrays, insertion or bubble sort is acceptable. For large datasets, merge, quick, or heap sort is recommended. Non-comparison sorts excel when data has specific constraints.
