	Sort Algorithm	Time Complexity	Space C.	When To Use	Comparison	Final Advice
1	Bubble Sort	O(n^2)	O(1)	Small datasets or when simplicity is key	Simple but inefficient for large datasets. Slowest for most practical applications.	"Why use Bubble Sort when there are more efficient options?"
2	Selection Sort	O(n^2)	O(1)	When memory usage is a concern since it does not require additional space.	Slightly better than Bubble Sort in terms of swap optimization, but still inefficient for larger datasets.	"Selection Sort minimizes swaps, but does little for speed."
3	Insertion Sort	O(n^2)	O(1)	Small datasets or partially sorted data.	Faster than Bubble and Selection Sort in practice for small or nearly sorted datasets.	"Works well on nearly sorted data."
4	Quick Sort	O(n log n) or O(n^2)	O(log n)	For large datasets where in-place sorting is necessary.	One of the fastest sorting algorithms for large datasets on average, but vulnerable to poor performance in specific conditions (e.g., already sorted data).	"Quick Sort is fast and efficient, but can be slow in the worst case."
5	Merge Sort	O(n log n)	O(n)	When stability is required, or for large datasets where external sorting is necessary.	Stable and consistent in performance, but requires extra memory for merging.	"Merge Sort is reliable and stable but can be memory-intensive."
6	Counting Sort	O(n + k)	O(k)	When the range of the input is known and not too large, ideal for integers.	Extremely fast for specific situations, but not a general-purpose sort.	"Efficient for small ranges but not versatile."
7	Radix Sort	O(nk)	O(n + k)	When sorting integers or strings where a stable sort is required.	Excellent for fixed-size data like integers, but not suitable for comparison-based sorting.	"Radix Sort can be efficient when the range of data is manageable."
8	Bucket Sort	O(n + k)	O(n)	When data is uniformly distributed over a range and can be divided into buckets.	Similar to Counting Sort but more general, works best with uniformly distributed data.	"Bucket Sort can be very fast, but only with well-distributed data."
9	Comb Sort	<= O(n^2)	O(1)	As a refinement of Bubble Sort to optimize performance.	Slightly faster than Bubble Sort but still inefficient for large datasets.	"Comb Sort is faster than Bubble Sort but not the best for large data."
10	Shell Sort	O(n log n) or O(n^2)	O(1)	When optimizing Insertion Sort for larger datasets.	Works better than simple Insertion Sort but not as fast as Quick Sort.	"A clever tweak to Insertion Sort that’s better than Bubble, but not as good as Quick or Merge."
11	Heap Sort	O(n log n)	O(1)	When in-place sorting is necessary, and stability is not required.	Efficient in time complexity, but not as fast in practice as Quick or Merge Sort due to higher constant factors.	"Heap Sort is reliable, but not the fastest in practice."