sort comparison

best, average, worst.

	time complexity	space complexity	technique
selection Sort	O(n^2)	O(1)	Brute Force
bubble Sort	O(n)/O(n^1)/O(n^1)	O(1)	Brute Force
quick Sort	O(n log n)/O(n log n)/O(n^2)	O(log n)/O(log n)/O(n)	Divide and Conquer
merge Sort	O(n log n)	O(n)	Divide and Conquer
insertion Sort	O(n)/O(n^2)/O(n^2)	O(1)	Brute Force
count Sort	O(n + k)	O(k)	Special Case

---

	pros	cons
selection Sort	- Stable sorting algorithm (preserves the order of equal elements). - Low space complexity (O(1)).	- Poor performance for large datasets (O(n^2) time complexity). - Makes many comparisons and swaps, leading to inefficiency for larger arrays.
bubble Sort	- Stable sorting algorithm. - Low space complexity (O(1)).	- Extremely inefficient for large datasets (O(n^2) time complexity in most cases). - Makes many unnecessary comparisons and swaps, even when the array is partially sorted.
quick Sort	- Excellent average and best-case performance (O(n log n) time complexity). - Efficient for large datasets due to its divide-and-conquer approach.	- Worst-case performance can be O(n^2), which occurs with a poorly chosen pivot element. - Additional space complexity (O(log n) on average) due to the recursion stack.
merge Sort	- Guaranteed O(n log n) time complexity for all cases. - Efficient for large datasets due to its divide-and-conquer approach. - Stable sorting algorithm.	- Requires additional space (O(n)) for the temporary array used during merging. - Slightly more complex to implement compared to selection or bubble sort.

	pros	**cons**
insertion Sort	- Efficient for small datasets or nearly sorted arrays (O(n) time complexity in these cases). - Low space complexity (O(1)). - Stable sorting algorithm.	- Performance can degrade to O(n^2) for large datasets and randomly ordered arrays. - Makes comparisons and shifts elements, which can be inefficient for very large arrays.
count Sort	- Excellent performance for data with a limited range of values (O(n + k) time complexity, where k is the range). - Efficient for counting occurrences and placing elements directly in sorted positions.	- Space complexity can be high (O(k)) for datasets with a large range of values. - Not suitable for general-purpose sorting due to the requirement of a limited value range. - Not stable sorting algorithm (equal elements might not preserve their order).