SUMMARY OF SORTING ALGORITHMS

Algorithm	Best Case	Average Case	Worst Case	Space Complexity	Stable	When to Use
Bubble Sort	O(n)	O(n^2)	O(n^2)	O(1)	Yes	**Educational purposes, small datasets:** Due to its simplicity, Bubble Sort is often used to demonstrate basic sorting concepts. It's not practical for large datasets but can be useful for small arrays or lists where efficiency is not a concern. Best used when the dataset is very small and the cost of code simplicity is prioritized over performance.
Insertion Sort	O(n)	O(n^2)	O(n^2)	O(1)	Yes	**Nearly sorted or small datasets:** Insertion Sort works efficiently on small datasets and is adaptive to nearly sorted data, offering a fast O(n) performance in the best case. It's a good choice when you know the data is almost sorted or when you need an easy-to-implement algorithm for small sets.
Selection Sort	O(n^2)	O(n^2)	O(n^2)	O(1)	No	**Small datasets or when memory writes are costly:** Selection Sort can be useful when the cost of swaps is high because it performs at most n swaps. It's not recommended for large datasets due to its O(n^2) time complexity, but it's easy to implement and has a simple structure, making it useful in limited scenarios.
Merge Sort	O(n log n)	O(n log n)	O(n log n)	O(n)	Yes	**Large datasets, stability required:** Merge Sort is excellent for large datasets where stable sorting is required. It's also effective for sorting linked lists due to its sequential access pattern. Its consistent O(n log n) performance makes it a go-to choice for situations requiring reliable and predictable efficiency.
Quick Sort	O(n log n)	O(n log n)	O(n^2)	O(log n)	No	**General-purpose, large datasets:** Quick Sort is typically the fastest algorithm for large datasets, with an average O(n log n) performance. While its worst case is O(n^2), this can be mitigated using techniques like randomized pivot selection. It's not stable but performs well in most real-world cases, making it widely used for general sorting tasks.
Heap Sort	O(n log n)	O(n log n)	O(n log n)	O(1)	No	**Memory-efficient, large datasets:** Heap Sort is valuable when you need O(n log n) performance and memory efficiency. Since it's an in-place algorithm, it's a great choice when memory is limited. However, it's not stable and is usually outperformed by Quick Sort in most cases. Best used in systems with tight memory constraints or where worst-case performance guarantees are needed.