🔍 Search Algorithms

Algorithm	**Time (Best / Avg / Worst)**	**Space**	**Approach Summary**	**Concept Used**
Linear Search	O(1) / O(n) / O(n)	O(1)	Traverse each element until the key is found.	Brute Force
Binary Search	O(1) / O(log n) / O(log n)	O(1)	Repeatedly divide sorted array in half.	Divide and Conquer
Ternary Search	$O(\log_3 n)$ / $O(\log_3 n)$ / $O(\log_3 n)$	O(1)	Split array into 3 parts and check mid-points.	Divide and Conquer
Jump Search	$O(\sqrt{n})$ / $O(\sqrt{n})$ / O(n)	O(1)	Jump ahead by $\sqrt{n}$ blocks, then do linear search.	Block Skipping
Exponential Search	O(log i) / O(log i) / O(log i)	O(1)	Find range exponentially, then binary search.	Binary + Exponential
Interpolation Search	O(1) / O(log log n) / O(n)	O(1)	Estimate mid using value-based formula (only for uniformly distributed arrays).	Value Prediction
Fibonacci Search	O(log n) / O(log n) / O(log n)	O(1)	Use Fibonacci numbers to split search range.	Fibonacci Numbers

---

🔃 Sorting Algorithms

Algorithm	**Time (Best / Avg / Worst)**	**Space**	**Approach Summary**	**Concept Used**
Bubble Sort	$O(n)$ / $O(n^2)$ / $O(n^2)$	O(1)	Repeatedly swap adjacent elements if they’re in wrong order.	Brute Force
Selection Sort	$O(n^2)$ / $O(n^2)$ / $O(n^2)$	O(1)	Select the smallest element & put it in the correct position.	Selection-based
Insertion Sort	$O(n)$ / $O(n^2)$ / $O(n^2)$	O(1)	Build sorted array one item at a time.	Incremental Building
Merge Sort	O(n log n) / O(n log n) / O(n log n)	O(n)	Divide array and merge in sorted order.	Divide and Conquer
Quick Sort	$O(n \log n)$ / $O(n \log n)$ / $O(n^2)$	O(log n)	Pick pivot, partition elements around pivot, and sort halves.	Divide and Conquer

Algorithm	**Time (Best / Avg / Worst)**	**Space**	**Approach Summary**	**Concept Used**
Heap Sort	O(n log n) / O(n log n) / O(n log n)	O(1)	Build max heap and remove top one by one.	Heap Tree
Counting Sort	O(n + k) / O(n + k) / O(n + k)	O(k)	Count frequency of elements, then rebuild array.	Counting (Non-Comparison)
Radix Sort	O(nk) / O(nk) / O(nk)	O(n + k)	Sort digits from least to most significant (uses Counting Sort).	Digit-wise Grouping
Bucket Sort	O(n + k) / O(n) / O(n^2)	O(n + k)	Distribute elements in buckets and sort each one.	Divide and Distribute
Shell Sort	O(n log n) / O(n $\log^2$ n) / O(n^2)	O(1)	Gap-based insertion sort for distant elements.	Gap Reduction
Tim Sort	O(n) / O(n log n) / O(n log n)	O(n)	Hybrid of Merge and Insertion sort (used in Python & Java).	Hybrid Sorting