Sorting Notes

1.Bubble Sort – Bubble Sort is a simple comparison-based sorting algorithm that repeatedly swaps adjacent elements if they are in the wrong order. This process continues until the array is sorted.

Working Principle:

-Traverse the array from the first element to the last.

-Compare adjacent elements; if the first is greater than the second, swap them.

-After each full pass, the largest unsorted element moves to its correct position at the end.

-The process is repeated for the remaining unsorted elements until the entire array is sorted.

Best Case (Already Sorted): O(n)
Average Case: O(n^2)
Worst Case (Reversed Order): O(n^2)
Space Complexity:O(1) (In-place sorting)

-Optimized Bubble Sort:

-If no swaps occur in a pass, the array is already sorted, and further

passes are unnecessary.
    -We introduce a flag to check whether swaps were made in an iteration. If no swaps were made, we terminate the algorithm early.
    -This optimisation improves performance in the best case to O(n).

-Recursive Bubble Sort:
    Instead of using loops, recursion is used to perform passes.
    In each recursive call, one pass is completed, reducing the problem size.
    Base case: When the array size becomes 1, it is already sorted.
    Complexity remains O(n^2), but recursion introduces an extra space overhead due to the call stack.

2. Selection Sort
    -Selection Sort is a simple sorting algorithm that divides the array into two parts: sorted and unsorted.
    It repeatedly selects the smallest (or largest) element from the unsorted part and swaps it with the first element of the unsorted section.

Working Principle:

-Start from the first element and find the smallest element in the entire array.

-Swap it with the first element.

-Move to the second element and find the next smallest element from the remaining unsorted part.

-Repeat the process until all elements are sorted.

Time Complexity:
Best Case: O(n^2)
Average Case: O(n^2)
Worst Case: O(n^2)
Space Complexity: O(1) (In-place sorting)

-Recursive Selection Sort:

Instead of using loops, recursion is used to find the minimum element and swap it with the first element in each recursive call.
Base case: When the array size becomes 1, it is already sorted.
Complexity remains O(n^2), but recursion introduces extra space overhead.

3.MergeSort

-MergeSort is a divide and conquer sorting algorithm that divides an array

into two halves, sorts each half recursively, and then merges them in a sorted manner.

Working Principle:

-Divide: Split the array into two halves.
-Conquer: Recursively sort both halves.
-Merge: Merge the sorted halves into a single sorted array.

Time Complexity:Best,Avg,Worst Case: O(nlogn)
Space : O(n)

Key Points:
Stable sorting algorithm (maintains the order of equal elements).
Preferred for linked lists since merging can be done in O(1) space.
Not an in-place sorting algorithm due to additional space usage.
Works well for large datasets and external sorting (handling large files).

4.QuickSort

Pivot - choose any element, then after the first pass the left side contains elements less than pivot and right side elements greater than pivot. Pivot

placed at correct position

QuickSort is a divide and conquer sorting algorithm that selects a pivot element and partitions the array such that elements smaller than the pivot appear on the left and larger elements on the right. The process is then recursively applied to the left and right subarrays.

Working Principle:
-Choose a pivot element. (Different strategies exist: first element, last element, middle element, or random selection).
-Partition the array so that elements smaller than the pivot are moved to its left and larger elements to its right.
-Recursively apply the same logic to the left and right partitions.
-Base case: When a subarray has one or zero elements, it is already sorted.

Time Complexity: Best,Avg Case: O(nlogn)
Worst case scenario: O(n^2)
Space Complexity (In-place version): O(logn) (for recursion stack)

Key Takeaways:
-Bubble Sort and Selection Sort are

inefficient for large datasets due to O(n^2) complexity.
-QuickSort is efficient but can degrade to O(n^2) if pivot selection is poor.
-MergeSort always runs in O(nlogn) time but uses extra space.
-For small datasets, QuickSort is often preferred due to its in-place nature and better cache performance.

5.Hybrid Sorting Algorithms – TimSort (Merge+Insertion) partially sorted data.