Quick Sort

- Quick Sort follows the **Divide and Conquer** approach.
- It selects a **pivot**, partitions the array around the pivot, and recursively sorts the left and right sub-arrays.

Steps of Quick Sort:

1. **Divide:** Choose a pivot and partition the array into elements **smaller than** and **greater than** the pivot.
2. **Conquer (Sort Recursively):** Apply Quick Sort recursively on the left and right sub-arrays.
3. **Combine:** Since sorting happens in-place, no explicit merging is needed.

Example

Consider the array: [8, 3, 7, 4, 9, 2, 6, 5]

Step 1: First Partition

Pivot = **6**

- Left: [3, 4, 2, 5] (elements < 6)
- Middle: [6]
- Right: [8, 7, 9] (elements > 6)

Step 2: Sorting Left [3, 4, 2, 5]

Pivot = **4**

- Left: [3, 2]
- Middle: [4]
- Right: [5]

Step 3: Sorting [3, 2]

Pivot = **3**

- Left: [2]
- Middle: [3]
- Right: []

Sorted left partition: [2, 3, 4, 5] ✅

Step 4: Sorting Right [8, 7, 9]

Pivot = **7**

- Left: []
- Middle: [7]
- Right: [8, 9]

Step 5: Sorting [8, 9]

Pivot = **8**

- Left: []
- Middle: [8]
- Right: [9]

Sorted right partition: [7, 8, 9] ✅

Final Sorted Array:

Merging all partitions: **[2, 3, 4, 5, 6, 7, 8, 9]**

Time Complexity

- **Best Case: O(n log n)**
- **Average Case: O(n log n)**
- **Worst Case: $O(n^2)$** (If pivot selection is poor)
-

Why Use Quick Sort?

✅ **Faster for most cases** (In-place sorting, unlike Merge Sort).

✅ **No extra space needed** (Except for recursive calls).

✅ **Good for large datasets with efficient pivot selection**.

❌ **Not stable** (May not preserve order of equal elements).

❌ **Worst case $O(n^2)$ if pivot selection is bad**.