Sorting – Quick Sort

1. Introduction

Quick Sort is a highly efficient and widely used **comparison-based sorting algorithm** that follows the **Divide and Conquer** strategy.

It works by selecting a **pivot element** and arranging the array so that all elements smaller than the pivot come before it, and all elements larger come after it.

Quick Sort is preferred in practice because of its **fast average-case performance**.

---

2. What is Quick Sort?

Quick Sort works in three main steps:

1. **Choose a Pivot**

   Select an element from the array (first, last, middle, or random).

2. **Partition**

   Rearrange the array so:

   - Elements smaller than pivot go to the left
   - Elements greater than pivot go to the right

3. **Recursively Sort**

   Apply Quick Sort to the left and right subarrays.

---

3. Why Do We Learn Quick Sort?

Quick Sort is important because:

- It is one of the fastest sorting algorithms
- Used in real-world systems and libraries
- Demonstrates divide-and-conquer clearly
- Commonly asked in interviews

- Efficient for large datasets

4. Basic Idea Behind Quick Sort

The idea is:

- Pick a pivot element
- Place the pivot at its correct position
- Ensure left side contains smaller elements
- Ensure right side contains larger elements
- Recursively apply the same logic to subarrays

5. Logic for Quick Sort (Plain English)

1. Select a pivot element from the array
2. Rearrange elements so smaller elements are on the left and larger on the right
3. Place the pivot in its correct position
4. Apply Quick Sort on the left subarray
5. Apply Quick Sort on the right subarray
6. Repeat until subarrays contain one or zero elements

6. Visualization of Quick Sort

Consider the array:

```
[10, 7, 8, 9, 1, 5]
```

Choose pivot = 5

Partition:

```
[1] [5] [10, 7, 8, 9]
```

Recursive sorting:

```
[1]  +  [7, 8, 9, 10]
```

Final sorted array:

```
[1, 5, 7, 8, 9, 10]
```

---

7. Pivot Selection Techniques

Common pivot choices:

- First element
- Last element
- Middle element
- Random element (best in practice)

Choosing a good pivot improves performance.

---

8. Time and Space Complexity

Case	**Time Complexity**
Best Case	$O(n \log n)$
Average Case	$O(n \log n)$
Worst Case	$O(n^2)$

- **Space Complexity:** $O(\log n)$

  (Recursive stack space)

---

9. Advantages of Quick Sort

- Very fast on average
- In-place sorting (no extra array needed)

- Cache-friendly
- Widely used in practice

10. Limitations of Quick Sort

- Worst-case time complexity is $O(n^2)$
- Performance depends on pivot choice
- Not a stable sorting algorithm
- Recursive implementation can cause stack overflow

11. Real-World Applications

- Used in system libraries
- Sorting large datasets
- Competitive programming
- Database query optimization
- Real-time systems

12. Comparison with Merge Sort

Feature	Quick Sort	Merge Sort
Time Complexity	O(n log n) avg	O(n log n)
Space Usage	Low	High
Stability	Not Stable	Stable
Speed	Faster in practice	Slower

13. Summary

- Quick Sort uses pivot and partition
- Divide-and-conquer based algorithm

- Very fast on average
- Worst case is $O(n^2)$
- Commonly used in real systems
