Quick Sort

Quick Sort is a **Divide and Conquer** sorting algorithm.
Quick Sort works by **choosing a pivot element**, then **partitioning** the array so that:

- Elements **smaller than the pivot** go to the left

- Elements **greater than the pivot** go to the right

Then it **recursively applies** the same logic to the left and right subarrays.

Think of it like picking one guy as a reference point, and shoving everyone shorter to one side, taller to the other, then repeat inside each group.

🔹 Algorithmic Steps (pseudocode)

```
Algorithm: QuickSort(A, low, high)
Input: Array A from index low to high
Output: Sorted array A

1. if low < high then
2.     p ← Partition(A, low, high)
3.     QuickSort(A, low, p - 1)
4.     QuickSort(A, p + 1, high)
5. end if
```

Partition procedure (for step 5):

//This is the Lomuto partition, which picks the last element always

```
Algorithm: Partition(A, low, high)

1. pivot ← A[high]

2. i ← low - 1

3. for j ← low to high - 1 do

4.     if A[j] ≤ pivot then

5.         i ← i + 1

6.         swap A[i] and A[j]

7. end for

8. swap A[i + 1] and A[high]

9. return i + 1
```

Pivot Choice: If we accidentally pick the largest or the smallest element as a pivot (which happens in almost-sorted arrays), then Quick Sort tends to $O(n^2)$ which is equal to Bubble Sort!

🔹 Implementation in C++

```cpp
#include <iostream>

using namespace std;


// Partition function (Lomuto)

int partition(int arr[], int low, int high) {

    int pivot = arr[high];   // pivot element

    int i = low - 1;


    for (int j = low; j < high; j++) {

        if (arr[j] <= pivot) {

            i++;

            swap(arr[i], arr[j]);

        }

    }


    swap(arr[i + 1], arr[high]);

    return i + 1;

}


// Recursive Quick Sort

void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int p = partition(arr, low, high);

        quickSort(arr, low, p - 1);   // left subarray
        quickSort(arr, p + 1, high);  // right subarray
    }
}

int main() {
    int arr[] = {10, 7, 8, 9, 1, 5};
    int n = sizeof(arr) / sizeof(arr[0]);

    quickSort(arr, 0, n - 1);

    cout << "Sorted array: ";
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;

    return 0;
}
```

🔹 Time & Space Complexity

Case	Time Complexity
Best	O(n log n)
Average	O(n log n)
Worst	$O(n^2)$ (if we pick a bad pivot)

Space Complexity: O(n) - uses temporary arrays during merging.
Stable? ✅ Yes
Adaptive? ❌ No - still divides even if already sorted.

🔹 Properties

Property	Quick Sort
In-place	Yes
Stable	No
Divide & Conquer	Yes
Extra memory	Minimal (log n for stack space)
Practical speed	Very fast

Real-world note:

Quick Sort (with good pivot selection like *randomized* or *median-of-three*) is what most libraries prefer internally.