Quick Sort Algorithm Overview

Here's a **complete and clean summary** of the entire **Quick Sort explanation** video:

🔷 What is Quick Sort?

Quick Sort is a **Divide and Conquer** sorting algorithm that:

- Sorts an array in **ascending** or **descending** order.
- Works by **choosing a pivot**, placing it in its **correct sorted position**, and **recursively sorting** elements to the left and right.

🔷 Quick Sort Steps (High-Level Intuition)

1. **Pick a Pivot**
    - Choose any element: first, last, middle, or random.
    - In this tutorial: **first element** is chosen as the pivot.
2. **Place Pivot in Its Correct Position**
    - All elements **smaller** than the pivot go to the **left**.
    - All elements **greater** than the pivot go to the **right**.
    - This divides the array into two subarrays (left and right of pivot).
3. **Recursively Apply Quick Sort**
    - Sort the left subarray.
    - Sort the right subarray.
    - Stop when the subarray has $\leqslant 1$ element (already sorted).

🔷 Key Terms

- **Partition Index**: Final position of the pivot where left < pivot < right.

- **Divide and Conquer**:
  - Divide the array using pivot (partitioning).
  - Conquer by recursively sorting subarrays.

🔷 How Partitioning Works (Two Pointer Approach)

1. Initialize two pointers:
   - `i` at `low` (start of array)
   - `j` at `high` (end of array)
2. While `i <= j`:
   - Move `i` right until `arr[i] > pivot`
   - Move `j` left until `arr[j] < pivot`
   - If `i < j`: swap `arr[i]` and `arr[j]`
   - If `i >= j`: swap pivot with `arr[j]` → Now pivot is at correct place
3. Return `j` as the **partition index**.

🔷 Pseudo Code

```cpp
void quickSort(vector<int>& arr, int low, int high) {
    if (low < high) {
        int pIndex = partition(arr, low, high);
        quickSort(arr, low, pIndex - 1);    // Left part
        quickSort(arr, pIndex + 1, high);    // Right part
    }
}

int partition(vector<int>& arr, int low, int high) {
    int pivot = arr[low];
    int i = low + 1;
    int j = high;
```

```
    while (i <= j) {
        while (i <= high && arr[i] <= pivot) i++;
        while (j >= low && arr[j] > pivot) j--;
        if (i < j) swap(arr[i], arr[j]);
    }

    swap(arr[low], arr[j]); // Place pivot at correct position
    return j;
}
```

🔷 Time and Space Complexity

- **Time Complexity:**
  - **Best/Average Case:** `O(N log N)`
  - **Worst Case** (pivot at ends or sorted array): `O(N^2)`
- **Space Complexity:**
  - `O(log N)` for recursive calls (stack space)
  - No extra array (unlike Merge Sort)

🔷 Why Quick Sort?

✅ Faster than Merge Sort in practice (no extra space)
✅ Efficient for large datasets
⚠ But worst-case is `O(N²)` if pivot selection is poor

🔷 To Practice Further

As an **assignment**, try:

> ✍ Implementing **Quick Sort in descending order** by tweaking comparisons (`<` becomes `>` and vice versa)

Let me know if you want a C++, Python, or Java version of this implementation!