Quick Sort

📘 **Definition:**
Quick Sort is a **divide and conquer** algorithm that selects a **pivot** element, partitions the array into two parts (left: smaller, right: larger), and then recursively sorts the two parts.

💡 **Real-Life Analogy:**

Socho tumhare paas ek group of people hai jinko **height ke hisaab se line mein lagana hai**.

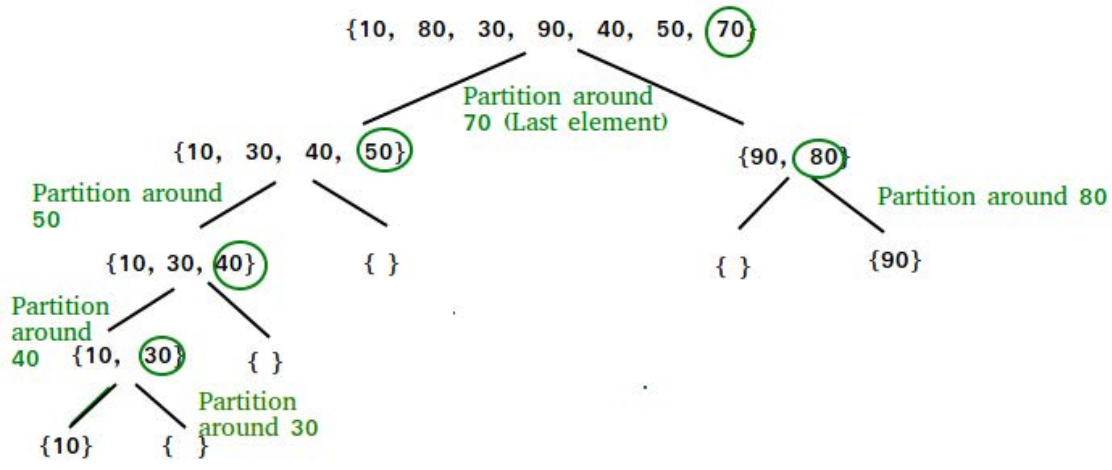
1. Tum randomly ek aadmi ko **pivot** choose karte ho.
2. Baaki sabse compare karte ho:
    - Jo pivot se chhote hain → uske left mein
    - Jo bade hain → right mein
3. Ab dono sides ko **alag-alag sort karo** — yahi hota hai recursion.

✅ **C Code: Quick Sort (with Partition Logic)**

```
#include <stdio.h>


// Partition function to place pivot correctly
int partition(int arr[], int low, int high) {
   int pivot = arr[high];  // last element as pivot
   int i = low - 1;       // index of smaller element

   for(int j = low; j < high; j++) {
      if(arr[j] < pivot) {
         i++;
         // Swap arr[i] and arr[j]
         int temp = arr[i];
         arr[i] = arr[j];
         arr[j] = temp;
      }
   }
```



```
    // Place pivot in the correct position
    int temp = arr[i + 1];
    arr[i + 1] = arr[high];
    arr[high] = temp;
    return i + 1; // New pivot position
}

// Recursive quick sort function
void quickSort(int arr[], int low, int high) {
    if(low < high) {
        int pi = partition(arr, low, high);  // Partitioning index
        quickSort(arr, low, pi - 1);         // Left of pivot
        quickSort(arr, pi + 1, high);        // Right of pivot
    }
}

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

    quickSort(arr, 0, size - 1);

    printf("Sorted array: ");
    for(int i = 0; i < size; i++) {
        printf("%d ", arr[i]);
    }

    return 0;
}
```

🧠 Step-by-Step Explanation

1. Pick a **pivot** (usually last element).
2. Partition:
    - Place elements < pivot to its left
    - pivot to its right
3. Recursively apply Quick Sort to left and right subarrays.

📜 Pseudocode (For Notes/Exams)

```
quickSort(arr, low, high):
  if low < high:
    pi = partition(arr, low, high)
    quickSort(arr, low, pi - 1)
    quickSort(arr, pi + 1, high)

partition(arr, low, high):
  pivot = arr[high]
  i = low - 1
  for j = low to high-1:
    if arr[j] < pivot:
      i++
      swap arr[i] with arr[j]
  swap arr[i+1] with arr[high]
  return i + 1
```

⚡ Time Complexity

Case	Time
Best Case	O(n log n) ✅
Average Case	O(n log n) ✅
Worst Case	$O(n^2)$ ❌ (when array is already sorted and bad pivot is chosen)
Space	O(log n) due to recursion stack

✅ **Use When:**

- Fast and efficient sorting is required
- Memory is limited (in-place)
- Data is large and randomly distributed
