**Quick Sort Explanation**

Quick Sort is a **divide-and-conquer** algorithm that works by selecting a **pivot** element from the array and partitioning the other elements into two sub-arrays according to whether they are less than or greater than the pivot. The sub-arrays are then sorted recursively.

The basic steps involved in Quick Sort are:

1. **Pick a pivot**: Choose an element from the array (typically the last element).
2. **Partitioning**: Rearrange the array so that elements smaller than the pivot are on the left, and elements greater than the pivot are on the right.
3. **Recursion**: Recursively apply the above steps to the left and right sub-arrays.

**Steps to Illustrate Quick Sort**

1. **Initial Array**: [9, 7, 5, 11, 12, 2, 14, 3, 10, 6]
2. **Choose a pivot**: Select the last element, 6.
3. **Partition the array**: Rearrange it to get elements smaller than 6 on the left and greater than 6 on the right.
4. **Recursively apply** the above steps on the left and right sub-arrays.

**Final Sorted Array**: [2, 3, 5, 6, 7, 9, 10, 11, 12, 14]

**Graph of Partitioning Example**

Below is a visual representation of the partitioning process in quick sort. Let's visualize each step:

**Initial array:**

[9, 7, 5, 11, 12, 2, 14, 3, 10, 6]

**Step 1: Choose pivot (6) and partition**

After partitioning:

[5, 2, 3, 6, 12, 7, 14, 11, 10, 9]

^ Pivot here

**Step 2: Recursively apply quick sort to the left ([5, 2, 3]) and right ([12, 7, 14, 11, 10, 9]) sub-arrays**

For left sub-array [5, 2, 3], pivot is 3:

[2, 3, 5]

For right sub-array [12, 7, 14, 11, 10, 9], pivot is 9:

[7, 9, 14, 11, 10, 12]

**Step 3: Continue recursively until sorted**

Final sorted array:

[2, 3, 5, 6, 7, 9, 10, 11, 12, 14]

**Python Code for Quick Sort**

Here’s the Python function to implement Quick Sort:

def quick\_sort(arr):

"""

Function to implement Quick Sort algorithm.

Args:

arr: List of elements to be sorted.

Returns:

Sorted list.

"""

# Base case: if the list is of length 0 or 1, it's already sorted

if len(arr) <= 1:

return arr

# Recursive case: choose pivot and partition the array

pivot = arr[-1] # We take the last element as the pivot

left = [] # List to store elements smaller than pivot

right = [] # List to store elements greater than pivot

middle = [] # List to store elements equal to pivot

# Partition the array into left, middle (pivot), and right sub-arrays

for elem in arr:

if elem < pivot:

left.append(elem)

elif elem > pivot:

right.append(elem)

else:

middle.append(elem)

# Recursively apply quick sort to left and right sub-arrays

return quick\_sort(left) + middle + quick\_sort(right)

# Example usage:

arr = [9, 7, 5, 11, 12, 2, 14, 3, 10, 6]

sorted\_arr = quick\_sort(arr)

print("Sorted Array:", sorted\_arr)

**Explanation of the Code:**

* **Base case**: If the array length is 0 or 1, it is already sorted, so we return it directly.
* **Partitioning**: We iterate over the array and compare each element with the pivot. Elements smaller than the pivot go to the left list, elements greater than the pivot go to the right list, and elements equal to the pivot go to the middle list.
* **Recursive calls**: We recursively apply quick sort to the left and right sub-arrays and combine them with the middle array (which contains the pivot).

**Time Complexity:**

* **Best case**: O(nlogn)O(n \log n)O(nlogn)
* **Worst case**: O(n2)O(n^2)O(n2) (when the pivot is always the smallest or largest element)
* **Average case**: O(nlogn)O(n \log n)O(nlogn)