***Lomuto Partition***

Quicksort is a Divide and Conquer Algorithm that is used for sorting the elements. In this algorithm, we choose a pivot and partitions the given array according to the pivot. Quicksort algorithm is a mostly used algorithm because this algorithm is cache-friendly and performs in-place sorting of the elements means no extra space requires for sorting the elements.

**Note:**

Quicksort algorithm is generally unstable algorithm because quick sort cannot be able to maintain the relative   
order of the elements.

**Three partitions are possible for the Quicksort algorithm:**

1. **Naive partition:** In this partition helps to maintain the relative order of the elements but this partition takes O(n) extra space.
2. **Lomuto partition:**In this partition, The last element chooses as a pivot in this partition. The pivot acquires its required position after partition but more comparison takes place in this partition.
3. **Hoare's partition:** In this partition, The first element chooses as a pivot in this partition. The pivot displaces its required position after partition but less comparison takes place as compared to the Lomuto partition.

**Lomuto partition** 

* **Lomuto’s Partition Algorithm (**[**unstable**](https://www.geeksforgeeks.org/stability-in-sorting-algorithms/)**algorithm)**

**Lomutopartition(arr[], lo, hi)**

pivot = arr[hi]

i = lo // place for swapping

for j := lo to hi – 1 do

if arr[j] <= pivot then

swap arr[i] with arr[j]

i = i + 1

swap arr[i] with arr[hi]

return i

**QuickSort(arr[], l, r)**

If r > l

1. Find the partition point of the array

m =Lomutopartition(a,l,r)

2. Call Quicksort for less than partition point

Call Quicksort(arr, l, m-1)

3. Call Quicksort for greater than the partition point

Call Quicksort(arr, m+1, r)

Java

// Java program to demonstrate the Lomuto partition

// in quick sort

import java.util.\*;

public class GFG {

static int sort(int numbers[], int start, int last)

{

int pivot = numbers[last];

int index = start - 1;

int temp = 0;

for (int i = start; i < last; ++i)

{

if (numbers[i] < pivot) {

++index;

// swap the position

temp = numbers[index];

numbers[index] = numbers[i];

numbers[i] = temp;

}

}

int pivotposition = ++index;

temp = numbers[index];

numbers[index] = pivot;

numbers[last] = temp;

return pivotposition;

}

static void quicksort(int numbers[], int start, int end)

{

if (start < end)

{

int pivot\_position = sort(numbers, start, end);

quicksort(numbers, start, pivot\_position - 1);

quicksort(numbers, pivot\_position + 1, end);

}

}

static void print(int numbers[])

{

for (int a : numbers) {

System.out.print(a + " ");

}

}

public static void main(String[] args)

{

int numbers[] = { 4, 5, 1, 2, 4, 5, 6 };

quicksort(numbers, 0, numbers.length - 1);

print(numbers);

}

}

**Output**

1 2 4 4 5 5 6