***Naive 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.

**1.  Naive partition**

**Algorithm:**

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

1. Make a Temporary array temp[r-l+1] length

2. Choose last element as a pivot element

3. Run two loops:

-> Store all the elements in the temp array that are less than pivot element

-> Store the pivot element

-> Store all the elements in the temp array that are greater than pivot element.

4.Update all the elements of arr[] with the temp[] array

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

If r > l

1. Find the partition point of the array

m = Naivepartition(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 naive partition

// in quick sort

import java.io.\*;

import java.util.\*;

public class GFG {

static int partition(int a[], int start, int high)

{

// Creating temporary

int temp[] = new int[(high - start) + 1];

// Choosing a pivot

int pivot = a[high];

int index = 0;

// smaller number

for (int i = start; i <= high; ++i) {

if (a[i] < pivot)

{

temp[index++] = a[i];

}

}

// pivot position

int position = index;

// Placing the pivot to its original position

temp[index++] = pivot;

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

{

if (a[i] > pivot)

{

temp[index++] = a[i];

}

}

// Change the original array

for (int i = start; i <= high; ++i) {

a[i] = temp[i - start];

}

// return the position of the pivot

return position;

}

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

{

if (start < end) {

int point = partition(numbers, start, end);

quicksort(numbers, start, point - 1);

quicksort(numbers, point + 1, end);

}

}

// Function to print the array

static void print(int numbers[])

{

for (int a : numbers)

{

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

}

}

public static void main(String[] args)

{

int numbers[] = { 3, 2, 1, 78, 9798, 97 };

// rearrange using naive partition

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

print(numbers);

}

}

**Output**

1 2 3 78 97 9798