Hoare 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.

Hoare's Partition

Hoare’s Partition Scheme works by initializing two indexes that start at two ends, the two indexes move toward each other until an inversion is (A smaller value on the left side and a greater value on the right side) found. When an inversion is found, two values are swapped and the process is repeated.

Algorithm:

```
Hoarepartition(arr[], lo, hi)

   pivot = arr[lo]
   i = lo - 1  // Initialize left index
   j = hi + 1  // Initialize right index

   // Find a value in left side greater
   // than pivot
   do
      i = i + 1
   while arr[i] < pivot

   // Find a value in right side smaller
   // than pivot
   do
      j--;
   while (arr[j] > pivot);

   if i >= j then
      return j

   swap arr[i] with arr[j]



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

```
If r > l
     1. Find the partition point of the array
              m =Hoarepartition(a,l,r)
     2. Call Quicksort for less than partition point
             Call Quicksort(arr, l, m)
     3. Call Quicksort for greater than the partition point
             Call Quicksort(arr, m+1, r)
```

Dash

All

Articles

Videos

Problems

Quiz

Java

```
 // Java implementation of QuickSort
// using Hoare's partition scheme

import java.io.*;

class GFG {

    // This function takes first element as pivot, and
    // places all the elements smaller than the pivot on the
    // left side and all the elements greater than the pivot
```

Courses Tutorials Jobs Practice Contests

```
    {
        int pivot = arr[low];
        int i = low - 1, j = high + 1;
        while (true)
        {
            // Find leftmost element greater
            // than or equal to pivot
            do {
                i++;
            } while (arr[i] < pivot);

            // Find rightmost element smaller
            // than or equal to pivot
            do {
                j--;
            } while (arr[j] > pivot);

            // If two pointers met.
            if (i >= j)
                return j;

            // swap(arr[i], arr[j]);
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;

        }
    }

    // The main function that
    // implements QuickSort
    // arr[] --> Array to be sorted,
    // low --> Starting index,
    // high --> Ending index
    static void quickSort(int[] arr, int low, int high)
    {
        if (low < high) {

            // pi is partitioning index,
            // arr[p] is now at right place
            int pi = partition(arr, low, high);

            // Separately sort elements before
            // partition and after partition
            quickSort(arr, low, pi);
            quickSort(arr, pi + 1, high);
        }
    }

    // Function to print an array
    static void printArray(int[] arr, int n)
    {
        for (int i = 0; i < n; ++i)
            System.out.print(" " + arr[i]);

        System.out.println();
    }

    // Driver Code
    static public void main(String[] args)
    {
        int[] arr = { 10, 17, 18, 9, 11, 15 };
        int n = arr.length;
        quickSort(arr, 0, n - 1);

        printArray(arr, n);
    }
}
```

Output

```
9 10 11 15 17 18
```

Mark as Read

Report An Issue

If you are facing any issue on this page. Please let us know.