

```
import java.lang.Math;
import java.util.Map;
import java.util.HashMap;
```

```
public class QuickSort {
```

```
    int getRandomNumber(int min, int max) {
        // Get a random number
        return (int) (Math.random() * (max - min)) + min;
    }
```

```
    int[] getInitializedRandomArray(int[] arr, int size) {
        // Inserting random elements into the array
        for(int i=0; i<size; i++) {
            arr[i] = this.getRandomNumber(0, 500);
        }
    }
```

```
    return arr;
}
```

```
    static void printArray(int[] arr, int size) {
        for(int j=0; j<size; j++) {
            System.out.print(arr[j]);
```

```
            // To not print ',' after the final element
            if (j!=size-1) {
                System.out.print(", ");
            }
        }
    }
```

```
    System.out.println("");
}
```

```
    static void swap(int[] arr, int i, int j)
    // Swap two elements of an array of index i and j
    {
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }
```

```
    static int partitionWithLastElementAsPivot(int[] arr, int
begin, int end, int size, int[] counter) {
        // Perform partition of an array with last element as
pivot
```

```
int pivot = arr[end];
```

```
int i = begin - 1;
```

```
for (int j = begin; j <= end - 1; j++) {  
    if (arr[j] < pivot) {  
        i++;  
        swap(arr, i, j);  
        counter[0]++;  
    }  
}
```

```
swap(arr, i + 1, end);  
counter[0]++;
```

```
System.out.println("After partition");  
printArray(arr, size);  
System.out.println("");
```

```
return (i + 1);  
}
```

```
static void quickSortUsingLastElementAsPivot(int[] arr, int  
begin, int end, int size, int[] counter) {  
    // Quick sort using last element as pivot  
    if (begin < end) {  
        int pi = partitionWithLastElementAsPivot(arr, begin,  
end, size, counter);
```

```
        quickSortUsingLastElementAsPivot(arr, begin, pi-1,  
size, counter);
```

```
        quickSortUsingLastElementAsPivot(arr, pi+1, end,  
size, counter);  
    }  
}
```

```
static int hoarePartition(int[] arr, int begin, int end, int  
size, int[] counter) {  
    // Perform partition using Hoare  
    int pivot = arr[begin];  
    int i = begin - 1, j = end + 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) {
            System.out.println("After partition");
            printArray(arr, size);
            System.out.println("");
            return j;
        }
        counter[0]++;
        swap(arr, i, j);
    }
}

```

```

static void hoareQuickSort(int[] arr, int begin, int end,
int size, int[] counter) {
    // Perform quick sort using Hoare
    if (begin < end) {
        int pi = hoarePartition(arr, begin, end, size,
counter);

```

```

        hoareQuickSort(arr, begin, pi, size, counter);

```

```

        hoareQuickSort(arr, pi+1, end, size, counter);
    }
}

```

```

public static void main(String[] args) {
    QuickSort qs = new QuickSort();

```

```

        Map<Integer, Integer>
inputToActualCountMapWorstCaseArray = new HashMap<>();

```

```

        // Worst Case Arrays

```

```

        // Sorted in ascending order of length = 14
        int[] arr1 = {15, 22, 31, 45, 66, 70, 82, 91, 102, 111,
125, 140, 178, 195};

        // Sorted in descending order of length = 23
        int[] arr2 = {310, 300, 290, 273, 251, 242, 226, 208,
170, 150, 90, 81, 65, 44, 30, 21, 19, 11, 8, 5};

        System.out.println("Original first array");
        printArray(arr1, arr1.length);

        int[] counter = {0};

        System.out.println("");
        quickSortUsingLastElementAsPivot(arr1, 0, arr1.length -
1, arr1.length, counter);

        System.out.println("Array length: " + arr1.length + "
Counter: " + counter[0]);

        inputToActualCountMapWorstCaseArray.put(arr1.length,
counter[0]);

        System.out.println("Sorted first array");
        printArray(arr1, arr1.length);

System.out.println("=====
=====");

        System.out.println("Original second array");
        printArray(arr2, arr2.length);

        counter[0] = 0;

        System.out.println("");
        quickSortUsingLastElementAsPivot(arr2, 0, arr2.length -
1, arr2.length, counter);

        System.out.println("Array length: " + arr2.length + "
Counter: " + counter[0]);

        inputToActualCountMapWorstCaseArray.put(arr2.length,
counter[0]);

        System.out.println("Sorted second array");

```

```
printArray(arr2, arr2.length);
```

```
System.out.println("=====");
```

```
int j = 1;
int[] randomArray = new int[30];
int randomArrayLength;
Map<Integer, Integer> inputToActualCountMapRandomArray =
new HashMap<>();
```

```
// Quick sort for random array
while (j < 4) {
    counter[0] = 0;
    randomArrayLength = qs.getRandomNumber(10, 20);
```

```
    randomArray =
qs.getInitializedRandomArray(randomArray, randomArrayLength);
```

```
    System.out.println("Original random array: " + j);
    printArray(randomArray, randomArrayLength);
```

```
    System.out.println("");
    quickSortUsingLastElementAsPivot(randomArray, 0,
randomArrayLength - 1, randomArrayLength, counter);
```

```
    System.out.println("Array length: " +
randomArrayLength + " Counter: " + counter[0]);
```

```
inputToActualCountMapRandomArray.put(randomArrayLength,
counter[0]);
```

```
    System.out.println("Sorted random array: " + j);
    printArray(randomArray, randomArrayLength);
```

```
System.out.println("=====");
```

```
    j++;
}
```

```
int k = 1;
```

```

        Map<Integer, Integer>
inputToActualCountMapHoarePartition = new HashMap<>();

        // Quick sort for random array using Hoare partition
        while(k < 3) {
            counter[0] = 0;
            randomArrayLength = qs.getRandomNumber(10, 20);

            randomArray =
qs.getInitializedRandomArray(randomArray, randomArrayLength);

            System.out.println("Original random array: " + j);
            printArray(randomArray, randomArrayLength);

            System.out.println("");
            hoareQuickSort(randomArray, 0, randomArrayLength -
1, randomArrayLength, counter);

inputToActualCountMapHoarePartition.put(randomArrayLength,
counter[0]);

            System.out.println("Sorted random array: " + k);
            printArray(randomArray, randomArrayLength);

System.out.println("=====
=====");
            k++;
        }

        System.out.println("\n Quick sort for worst case arrays
table\n");

        System.out.println(String.format("%10s %25s %10s %23s
%10s", "N", "|", "Actual Count", "|", "T(N) = (n * n)"));
        System.out.println(String.format("%s",
"-----"
));

        inputToActualCountMapWorstCaseArray.forEach((input,
count) -> {
            System.out.println(String.format("%10d %25s %10d
%25s %10d", input, "|", count, "|", (input * input)));
        });

```

```
System.out.println("\n Quick sort for random arrays  
table\n");
```

```
System.out.println(String.format("%10s %25s %10s %23s  
%10s", "N", "|", "Actual Count", "|", "T(N) = (n logn)"));  
System.out.println(String.format("%s",  
"-----"  
));
```

```
inputToActualCountMapRandomArray.forEach((input, count)  
-> {  
    System.out.println(String.format("%10d %25s %10d  
%25s %10d", input, "|", count, "|", (input * (int)  
(Math.log(input) / Math.log(2)))));  
});
```

```
System.out.println("\n Quick sort using Hoare Partition  
table\n");
```

```
System.out.println(String.format("%10s %25s %10s %23s  
%10s", "N", "|", "Actual Count", "|", "T(N) = (n logn)"));  
System.out.println(String.format("%s",  
"-----"  
));
```

```
inputToActualCountMapHoarePartition.forEach((input,  
count) -> {  
    System.out.println(String.format("%10d %25s %10d  
%25s %10d", input, "|", count, "|", (input * (int)  
(Math.log(input) / Math.log(2)))));  
});  
}  
}
```