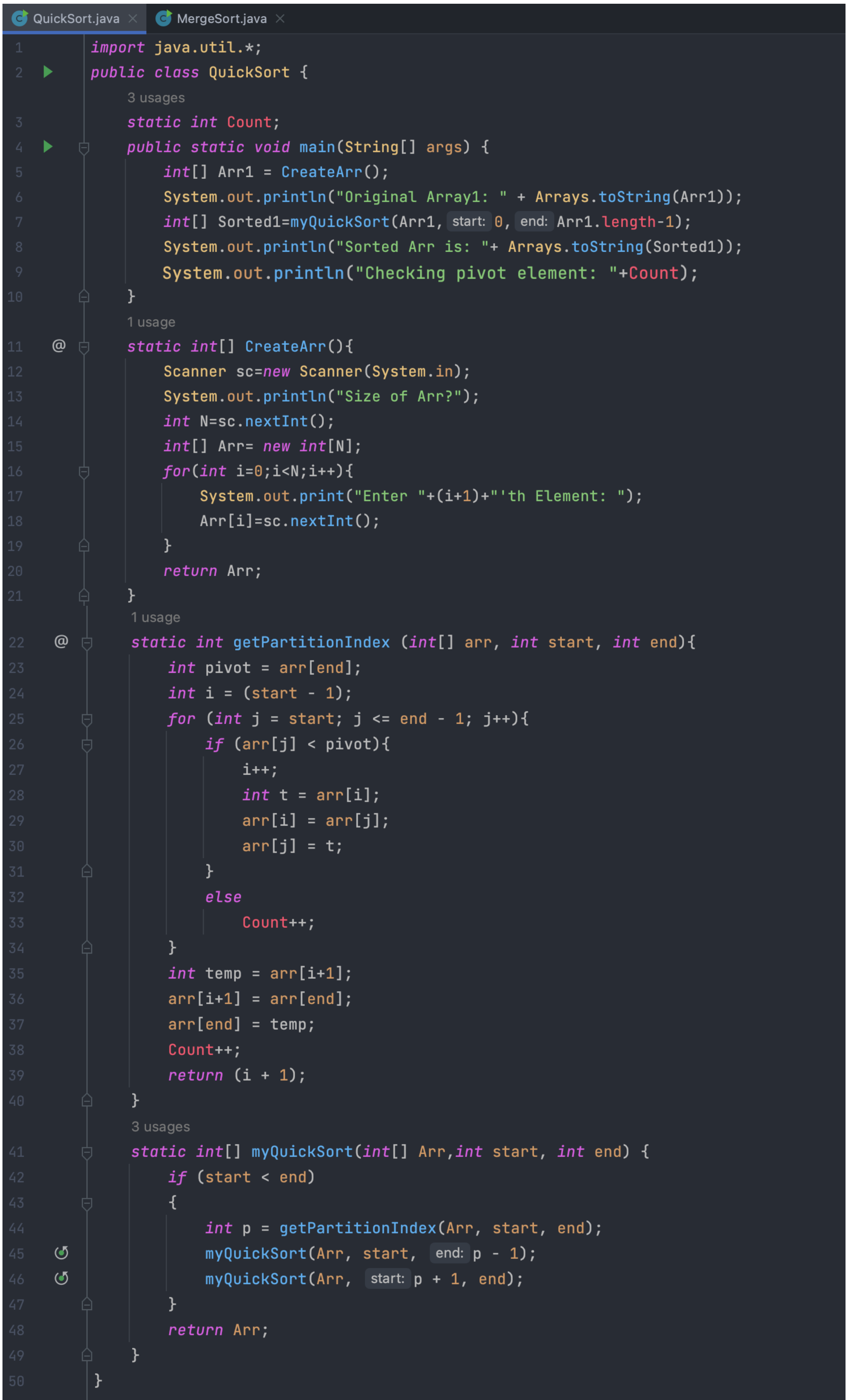
Assignment 6

Sayak Ghorai || BT21GCS004 || B2 || Design and Analysis of Algorithm

Q: Write an algo for Quick sort and Merge sort for an array of integers.

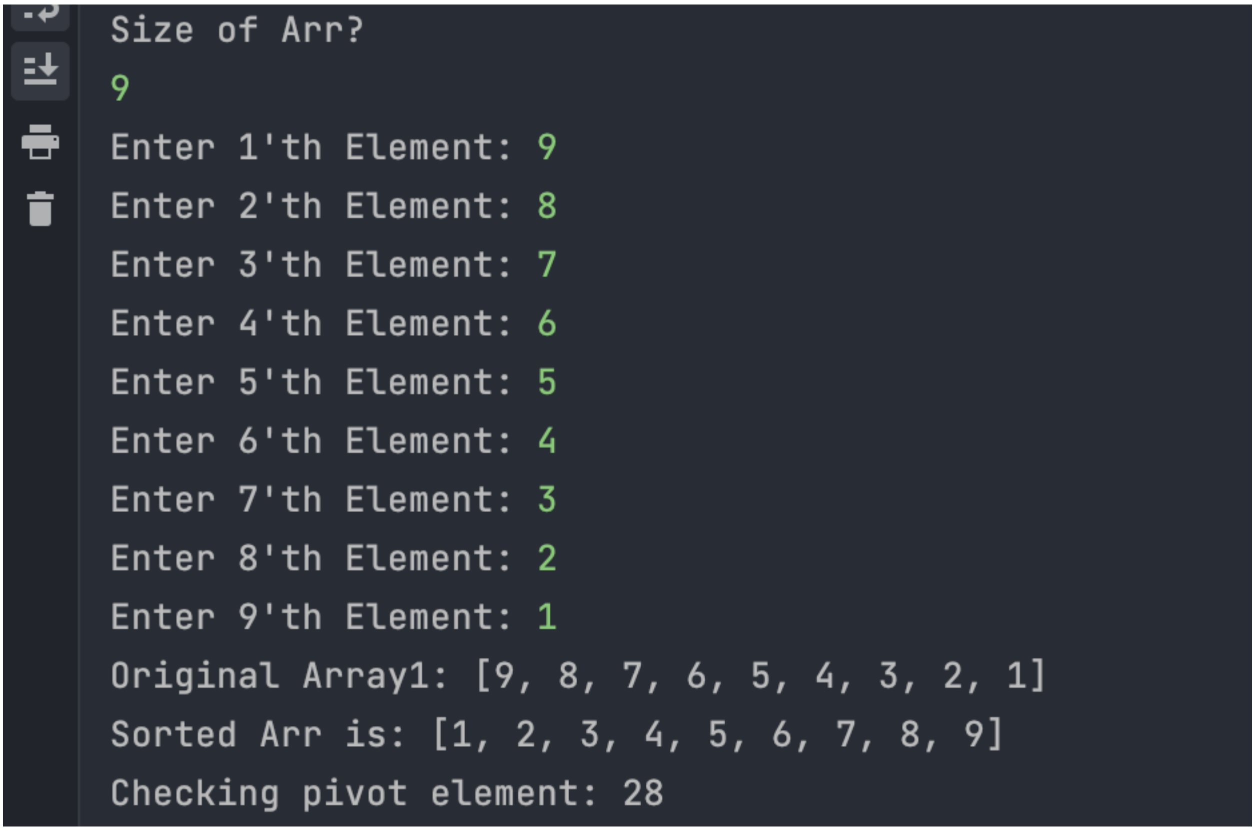
Code for Quick Sort:



```
import java.util.*;
public class QuickSort {
    3 usages
    static int Count;
    public static void main(String[] args) {
        int[] Arr1 = CreateArr();
        System.out.println("Original Array1: " + Arrays.toString(Arr1));
        int[] Sorted1=myQuickSort(Arr1, start: 0, end: Arr1.length-1);
        System.out.println("Sorted Arr is: "+ Arrays.toString(Sorted1));
        System.out.println("Checking pivot element: "+Count);
    }
    1 usage
    static int[] CreateArr(){
        Scanner sc=new Scanner(System.in);
        System.out.println("Size of Arr?");
        int N=sc.nextInt();
        int[] Arr= new int[N];
        for(int i=0;i<N;i++){
            System.out.print("Enter "+(i+1)+"'th Element: ");
            Arr[i]=sc.nextInt();
        }
        return Arr;
    }
    1 usage
    static int getPartitionIndex (int[] arr, int start, int end){
        int pivot = arr[end];
        int i = (start - 1);
        for (int j = start; j <= end - 1; j++){
            if (arr[j] < pivot){
                i++;
                int t = arr[i];
                arr[i] = arr[j];
                arr[j] = t;
            }
            else
                Count++;
        }
        int temp = arr[i+1];
        arr[i+1] = arr[end];
        arr[end] = temp;
        Count++;
        return (i + 1);
    }
    3 usages
    static int[] myQuickSort(int[] Arr,int start, int end) {
        if (start < end)
        {
            int p = getPartitionIndex(Arr, start, end);
            myQuickSort(Arr, start, end: p - 1);
            myQuickSort(Arr, start: p + 1, end);
        }
        return Arr;
    }
}
```

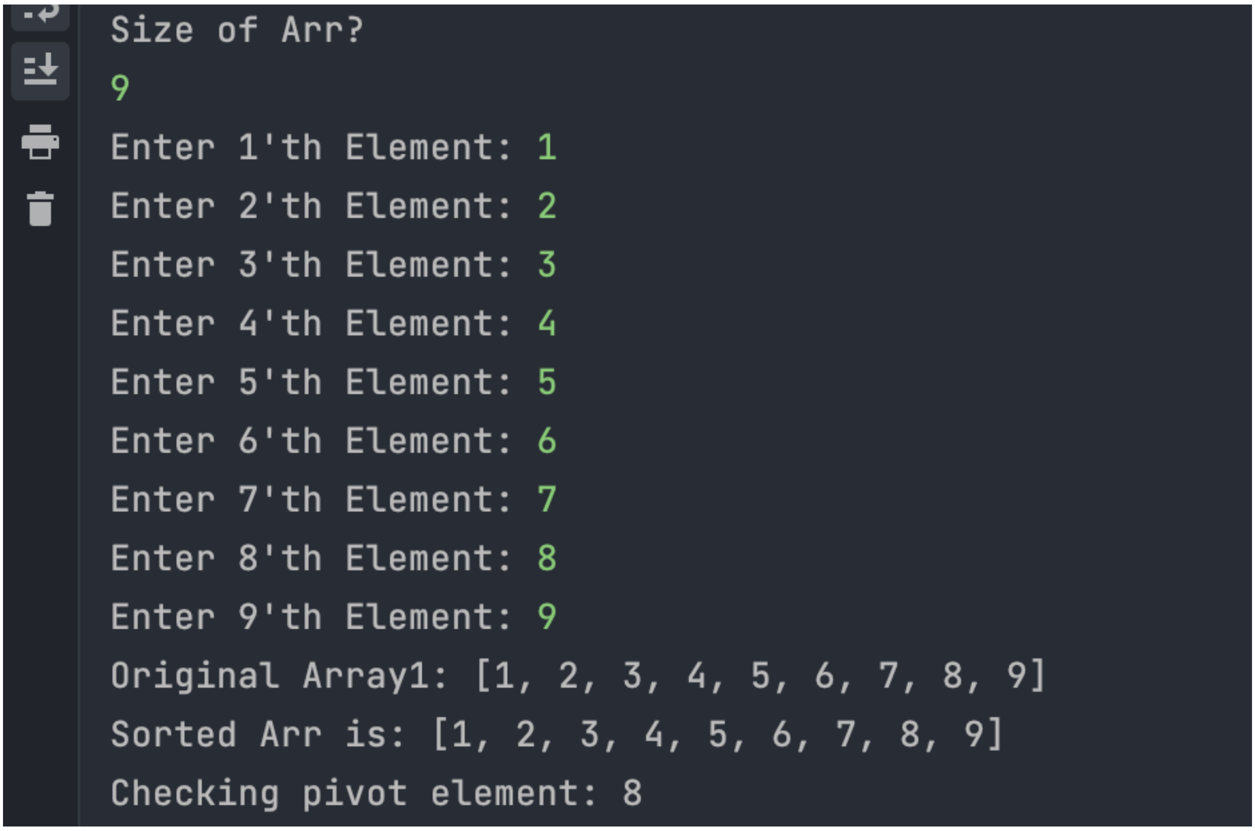
Output of Quick Sort:

Worst Case: Takes N^2 Steps » O(N^2)



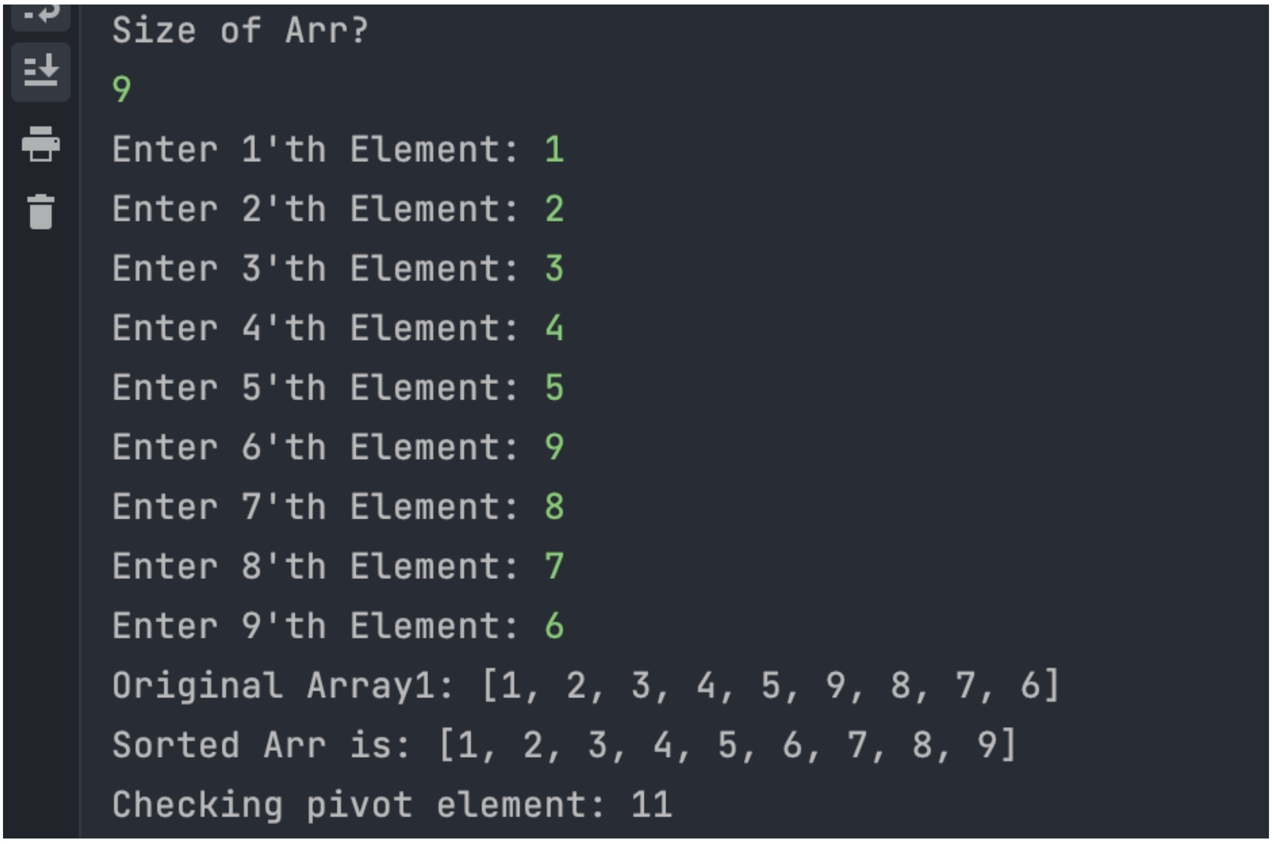
```
Size of Arr?
9
Enter 1'th Element: 9
Enter 2'th Element: 8
Enter 3'th Element: 7
Enter 4'th Element: 6
Enter 5'th Element: 5
Enter 6'th Element: 4
Enter 7'th Element: 3
Enter 8'th Element: 2
Enter 9'th Element: 1
Original Array1: [9, 8, 7, 6, 5, 4, 3, 2, 1]
Sorted Arr is: [1, 2, 3, 4, 5, 6, 7, 8, 9]
Checking pivot element: 28
```

Best Case: Takes Nxlog(N) Steps » O(Nxlog(N))



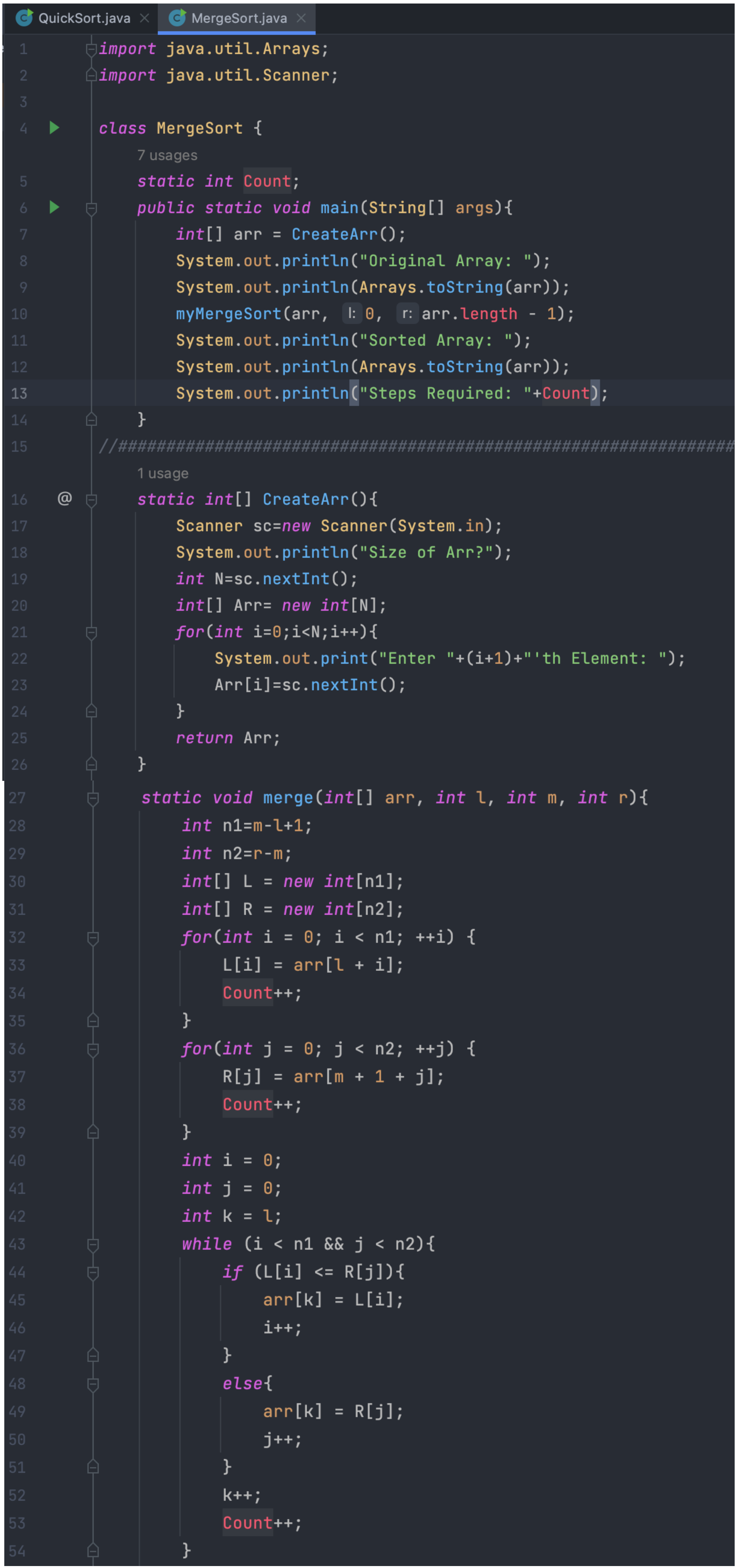
```
Size of Arr?
9
Enter 1'th Element: 1
Enter 2'th Element: 2
Enter 3'th Element: 3
Enter 4'th Element: 4
Enter 5'th Element: 5
Enter 6'th Element: 6
Enter 7'th Element: 7
Enter 8'th Element: 8
Enter 9'th Element: 9
Original Array1: [1, 2, 3, 4, 5, 6, 7, 8, 9]
Sorted Arr is: [1, 2, 3, 4, 5, 6, 7, 8, 9]
Checking pivot element: 8
```

Average Case:Takes Nxlog(N) Steps » O(Nxlog(N))

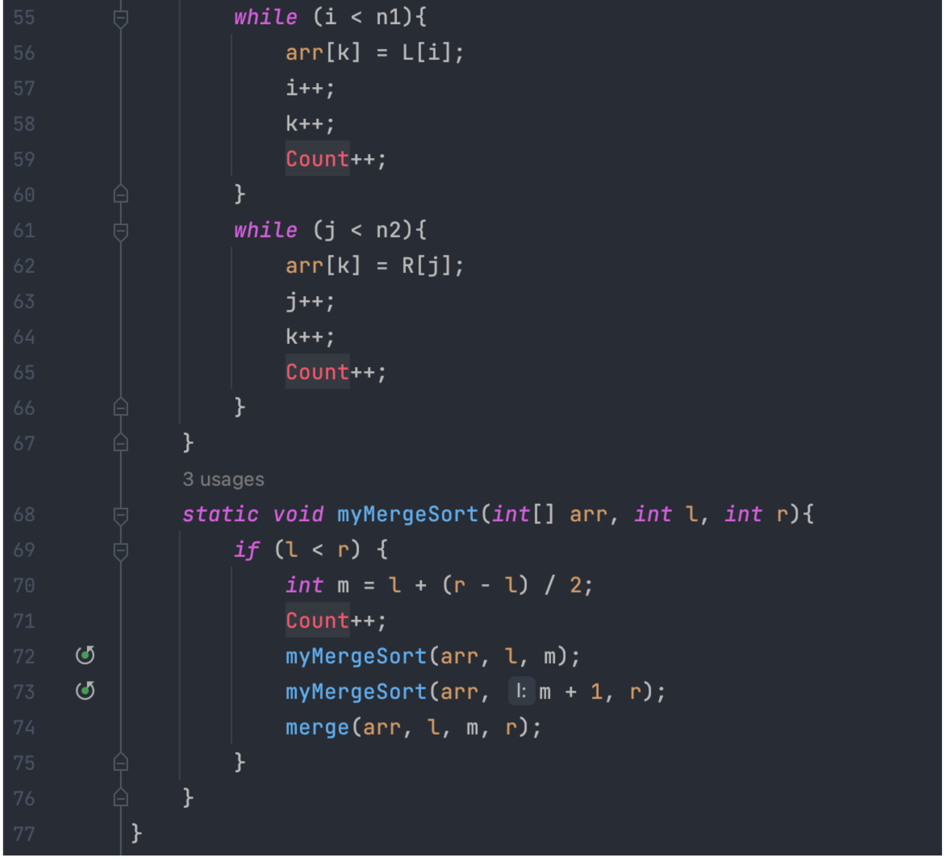


```
Size of Arr?
9
Enter 1'th Element: 1
Enter 2'th Element: 2
Enter 3'th Element: 3
Enter 4'th Element: 4
Enter 5'th Element: 5
Enter 6'th Element: 9
Enter 7'th Element: 8
Enter 8'th Element: 7
Enter 9'th Element: 6
Original Array1: [1, 2, 3, 4, 5, 9, 8, 7, 6]
Sorted Arr is: [1, 2, 3, 4, 5, 6, 7, 8, 9]
Checking pivot element: 11
```

Code for Merge Sort:



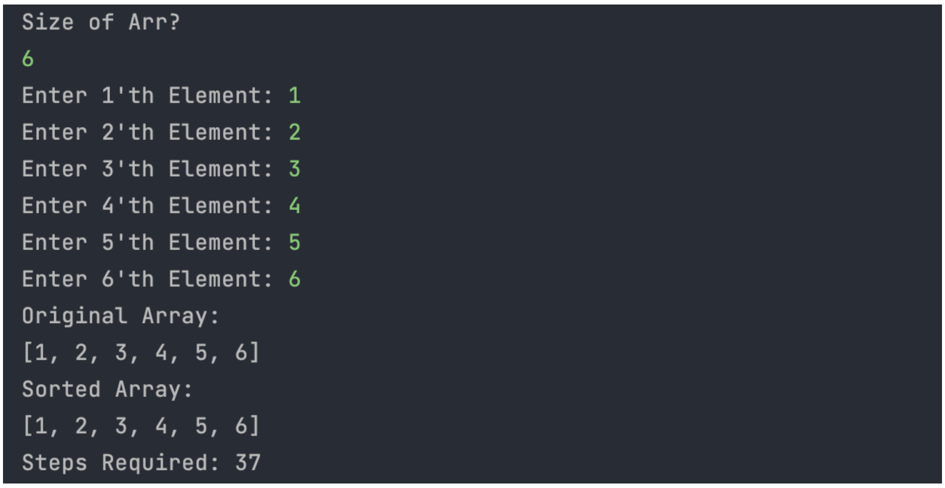
```
import java.util.Arrays;
import java.util.Scanner;

class MergeSort {
    static int Count;
    public static void main(String[] args){
        int[] arr = CreateArr();
        System.out.println("Original Array: ");
        System.out.println(Arrays.toString(arr));
        myMergeSort(arr, l: 0, r: arr.length - 1);
        System.out.println("Sorted Array: ");
        System.out.println(Arrays.toString(arr));
        System.out.println("Steps Required: "+Count);
    }
//################################################################
    static int[] CreateArr(){
        Scanner sc=new Scanner(System.in);
        System.out.println("Size of Arr?");
        int N=sc.nextInt();
        int[] Arr= new int[N];
        for(int i=0;i<N;i++){
            System.out.print("Enter "+(i+1)+"'th Element: ");
            Arr[i]=sc.nextInt();
        }
        return Arr;
    }
    static void merge(int[] arr, int l, int m, int r){
        int n1=m-l+1;
        int n2=r-m;
        int[] L = new int[n1];
        int[] R = new int[n2];
        for(int i = 0; i < n1; ++i) {
            L[i] = arr[l + i];
            Count++;
        }
        for(int j = 0; j < n2; ++j) {
            R[j] = arr[m + 1 + j];
            Count++;
        }
        int i = 0;
        int j = 0;
        int k = l;
        while (i < n1 && j < n2){
            if (L[i] <= R[j]){
                arr[k] = L[i];
                i++;
            }
            else{
                arr[k] = R[j];
                j++;
            }
            k++;
            Count++;
        }
```



```
        while (i < n1){
            arr[k] = L[i];
            i++;
            k++;
            Count++;
        }
        while (j < n2){
            arr[k] = R[j];
            j++;
            k++;
            Count++;
        }
    }
    3 usages
    static void myMergeSort(int[] arr, int l, int r){
        if (l < r) {
            int m = l + (r - l) / 2;
            Count++;
            myMergeSort(arr, l, m);
            myMergeSort(arr, l: m + 1, r);
            merge(arr, l, m, r);
        }
    }
}
```

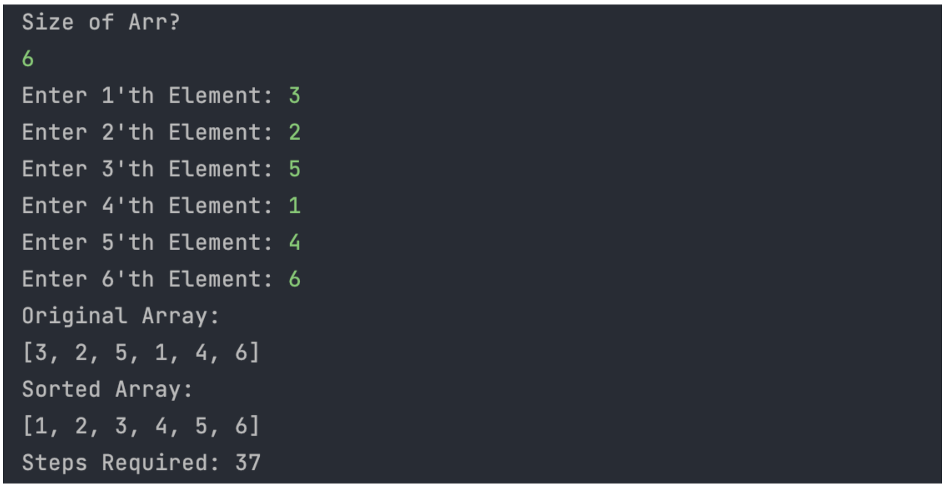
Output for Merge Sort:

Best Case:Takes Nxlog(N) Steps » O(Nxlog(N))



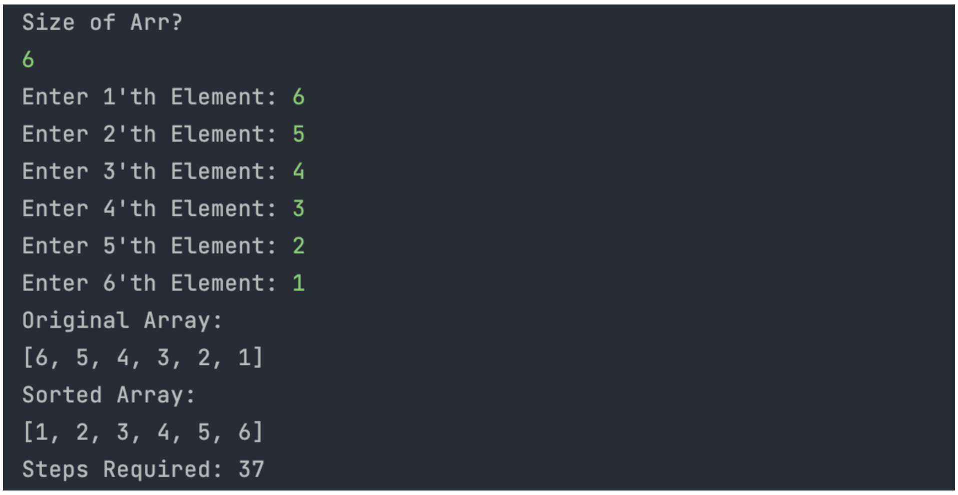
```
Size of Arr?
6
Enter 1'th Element: 1
Enter 2'th Element: 2
Enter 3'th Element: 3
Enter 4'th Element: 4
Enter 5'th Element: 5
Enter 6'th Element: 6
Original Array:
[1, 2, 3, 4, 5, 6]
Sorted Array:
[1, 2, 3, 4, 5, 6]
Steps Required: 37
```

Average Case:Takes Nxlog(N) Steps » O(Nxlog(N))



```
Size of Arr?
6
Enter 1'th Element: 3
Enter 2'th Element: 2
Enter 3'th Element: 5
Enter 4'th Element: 1
Enter 5'th Element: 4
Enter 6'th Element: 6
Original Array:
[3, 2, 5, 1, 4, 6]
Sorted Array:
[1, 2, 3, 4, 5, 6]
Steps Required: 37
```

Worst Case:Takes Nxlog(N) Steps » O(Nxlog(N))



```
Size of Arr?
6
Enter 1'th Element: 6
Enter 2'th Element: 5
Enter 3'th Element: 4
Enter 4'th Element: 3
Enter 5'th Element: 2
Enter 6'th Element: 1
Original Array:
[6, 5, 4, 3, 2, 1]
Sorted Array:
[1, 2, 3, 4, 5, 6]
Steps Required: 37
```

Analysis:

As we can see,
in Quick sort, Worst case is O(N^2) and in Merge sort, it is O(Nxlog(N)).
So for large number of unsorted inputs, Merge sort is better than Quick sort as (N^2 >> Nxlog(N));

In Quick sort, the Best case is O(Nxlog(N)), same as Merge sort. So it dosen't affect.

In Quick sort, the Average case is O(Nxlog(N)), same as Merge sort. So it dosen't affect.

So, for small unsorrted inputs, both performs well. But in case of large inputs, Merge sort is a better choice.