```
/**
 * Solomiya Pobutska
 * CISC3130
 * Assignment #6
 * */

import java.util.Scanner;
public class Sorting {
    private static Scanner in = new Scanner(System.in);
    private static int quickComp = 0, quickInterChange = 0,
mergeComp = 0, mergeInterChange = 0;
    private static int[] takingDataFromUser(int n) {
        // This function is used to take the data from the user
and returns the array
        int temp[] = new int[n];
        System.out.println("Values: ");
        for (int i = 0; i < n; i++) temp[i] = in.nextInt();
        return temp;
    }

    private static void bubbleSort(int a[], int n) {
        // This function is used to sort the array using bubble
sort
        int temp, comp = 0, interChange = 0;
        for (int i = 0; i < n - 1; i++) {
            for (int j = 0; j < n - i - 1; j++) {
                comp += 1;
                if (a[j] > a[j + 1]) {
```

```
                    // Swapping the elements
                    interChange += 1;
                    temp = a[j];
                    a[j] = a[j + 1];
                    a[j + 1] = temp;
                }
            }
        }
        System.out.println("Comparisions:" + comp);
        System.out.println("InterChanges:" + interChange);
    }


    private static int partition(int a[], int low, int high) {
  // This function is used to ensure that all elements which are
left to pivot is less than the pivot and right elements is to
greater
        int temp, pivot = a[high];
        int i = low - 1;
        for (int j = low; j <= high - 1; j++) {
            quickComp += 1;
            if (a[j] <= pivot) {
                // Swapping the elements
                quickInterChange += 1;
                i++;
                temp = a[i];
                a[i] = a[j];
                a[j] = temp;
            }
        }
        temp = a[i + 1];
```

```
    a[i + 1] = a[high];
    a[high] = temp;
    return i + 1;
}

private static void quickSort(int a[], int low, int high) {
          if (low < high) {
        int p = partition(a, low, high);
        // Calling partition for getting pivot element
        quickSort(a, low, p - 1);
        quickSort(a, p + 1, high);
        // Calling quickSort recursively to sort the array

    }
}

private static void merge(int a[], int l, int m, int r) {
    int i, j, k;
    int n1 = m - l + 1, n2 = r - m;
    int L[] = new int[n1], R[] = new int[n2];
    for (i = 0; i < n1; i++) L[i] = a[l + i];
    for (j = 0; j < n2; j++) R[j] = a[m + j + 1];
    i = 0;
    j = 0;
    k = l;
    mergeComp += 1;
    while (i < n1 && j < n2) {
        mergeInterChange += 1;
        // Swapping the elements
```

```
        if (L[i] <= R[j]) a[k++] = L[i++];
        else a[k++] = R[j++];
    }
    while (i < n1) a[k++] = L[i++];
    while (j < n2) a[k++] = R[j++];
}

private static void ms(int a[], int l, int r) {
        if (l < r) {
        int m = l + (r - l) / 2;
        ms(a, l, m);
        // Calling ms recursively to sort the array
        ms(a, m + 1, r);
        merge(a, l, m, r);
        // Calling merge for merging two sorted sub arrays
    }
}

public static void main(String args[]) {
    int n;
    System.out.print("Size of the array is ");
    n = in.nextInt();
    int input[] = new int[n];
    // Displaying the results
    input = takingDataFromUser(n);
    System.out.println("Bubble Sort");
    bubbleSort(input, n);
    System.out.println("Quick Sort");
    quickSort(input, 0, n - 1);
```

```
        System.out.println("Comparisions: " + quickComp);
        System.out.println("InterChanges: " + quickInterChange);
        System.out.println("Merge Sort");
        ms(input, 0, n - 1);
        System.out.println("Comparisions: " + mergeComp);
        System.out.println("InterChanges: " + mergeInterChange);
    }
}
```

Output:

```
10 NUMBERS IN ALMOST SORTED ORDER
Size of the array is 10
Values:
1 2 3 5 4 6 8 7 9 10

Bubble Sort
Comparisions: 45
InterChanges: 2

Quick Sort
Comparisions: 45
InterChanges: 45

Merge Sort
Comparisions: 9
InterChanges: 19

10 NUMBERS IN RANDOM ORDER
Size of the array is 10
Values:
4 6 7 3 5 9 0 3 1 9

Bubble Sort
Comparisions: 45
InterChanges: 23

Quick Sort
Comparisions: 45
InterChanges: 45
```

Merge Sort
Comparisons: 9
InterChanges: 19

10 NUMBERS IN REVERSE ORDER
Size of the array is 10
Values:
10 9 8 7 6 5 4 3 2 1

Bubble Sort
Comparisions: 45
InterChanges: 45

Quick Sort
Comparisions: 45
InterChanges: 45

Merge Sort
Comparisions: 9
InterChanges: 19

50 NUMBERS IN ALMOST SORTED ORDER
Size of the array is 50
Values:
1 3 2 5 4 7 6 8 9 10 12 11 13 14 15 17 16 18 19 21 20 22 23 24 25 27
26 28 29 30 31 32 33 35 34 36 39 38 37 40 41 42 43 44 45 46 47 49 48
50

Bubble Sort
Comparisions: 1225
InterChanges: 12

Quick Sort
Comparisions: 1225
InterChanges: 1225

Merge Sort
Comparisions: 49
InterChanges: 153

50 NUMBERS IN RANDOM ORDER
Size of the array is 50
Values:
45 32 15 1 42 2 26 39 13 4 27 46 29 19 44 47 5 9 33 24 35 34 40 30 16
22 23 48 43 20 36 6 49 10 14 31 12 37 38 7 17 28 50 11 25 3 8 18 41 21

Bubble Sort
Comparisions: 1225

InterChanges: 637

Quick Sort
Comparisions: 1225
InterChanges: 1225

Merge Sort
Comparisions: 49
InterChanges: 153

50 NUMBERS IN REVERSE ORDER
Size of the array is 50
Values:
50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

Bubble Sort
Comparisions: 1225
InterChanges: 1225

Quick Sort
Comparisions: 1225
InterChanges: 1225

Merge Sort
Comparisions: 49
InterChanges: 153

100 NUMBERS IN ALMOST SORTED ORDER
Size of the array is 100
Values:
1 5 4 3 2 6 7 8 9 10 12 11 15 13 15 14 16 17 18 19 20 21 22 23 24 25 27 26 28 29 30 31 32 34 33 36 35 37 38, 39, 40 41 43 42 45 44 46 47 48 49 50 52 51 54 53 55 56 57 58 59 60 61 63 65 64 62 66 67 68 70 69 71 72 73 75 74 76 77 78 79 80 81 83 82 85 84 87 86 89 88 90 93 91 92 94 95 96 97 100 99 98

Bubble Sort
Comparisions: 4950
InterChanges: 19

Quick Sort
Comparisions: 4950
InterChanges: 4950

Merge Sort
Comparisions: 99
InterChanges: 356

100 NUMBERS IN RANDOM ORDER
Size of the array is 100
Values:
9 6 4 3 2 1 5 8 10 56 57 58 59 60 14 16 17 18 19 20 21 22 23 24 25 27
26 28 29 30 31 32 34 33 36 35 37 38, 39, 40 41 43 42 45 44 46 47 48 49
50 52 51 54 53 55 12 11 15 13 15 14 61 63 65 64 62 66 67 68 70 69 71
72 73 75 74 76 77 78 79 80 81 83 82 85 84 87 86 89 88 90 93 91 92 94
95 100 96 99 98 97 95

Bubble Sort
Comparisions: 4950
InterChanges: 2567

Quick Sort
Comparisions: 4950
InterChanges: 4950

Merge Sort
Comparisions: 99
InterChanges: 356

100 NUMBERS IN REVERSE ORDER
Size of the array is 100
Values:
100 99 98 97 96 95 94 93 92 91 90 89 88 87 86 85 84 83 82 81 80 79 78
77 76 75 74 73 72 71 70 69 68 67 65 64 63 62 61 60 59 58 57 56 55 54
53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31
30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6
5 4 3 2 1

Bubble Sort
Comparisions: 4950
InterChanges: 4950

Quick Sort
Comparisions: 4950
InterChanges: 4950

Merge Sort
Comparisions: 99
InterChanges: 356