```java
import java.util.Random;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;

import java.util.Arrays;
import java.util.Collections;


public class TestRandom {

    public static int count = 0;

    public static void main(String[] args) {

        int[] all = {
            10,
            25,
            35,
            45,
            60,
            70,
            85,
            95
        };

        for (int a: all) {



            System.out.print("N = " + a + "   -   ");

            List < Integer > test = new ArrayList < Integer > ();

            for (int i = 0; i < a; i++) {

                int ele = getR(5, 70);

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

                test.add(ele);

            }
```

```
        int t = test.size();

        int[] test_array = new int[t];

        int i = 0;

        /** List to array **/
        for (int test_ele: test) {
            test_array[i] = test_ele;
            i++;
        }

        /** Original Data **/
        quickSort(test_array);


        count = 0;
        quickSort(test_array);


        System.out.println(" ");

        System.out.print("Sorted  -  ");
        for (int test_array_ele: test_array) {

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

        }

        System.out.println(" ");

        System.out.println("Worst Case: N^2 = " + a * a);
        System.out.println("Average Case: N^LogN = " + a * log2(a));

        System.out.println("Actual Count : = " + count);



        System.out.println(" ");


}
```

```
}

/** reverse the array **/
public static int[] reverseArray(int[] a) {

    int L = 0;
    int R = a.length - 1;

    while (L <= R) {

        int m = a[L];

        a[L] = a[R];

        a[R] = m;

        L++;
        R--;

    }
    return a;
}

/** get random number **/
private static int getR(int min, int max) {

    Random r = new Random();
    return r.nextInt((max - min) + 1) + min;
}

/** get Log N based on 2 **/
public static int log2(int N) {

    int result = (int)(Math.log(N) / Math.log(2));

    return result;
}

/** quick sort **/
public static void quickSort(int[] a) {


    int left = 0;
    int right = a.length - 1;
```

```
    quickSort(a, left, right);

}

private static void quickSort(int[] a, int low, int high) {

    ++count;

    if (low < high) {

        int pivot = partition(a, low, high); // divide

        quickSort(a, low, pivot - 1); // recursively process low part
        quickSort(a, pivot + 1, high); // recursively process high part

    }
}

/** Regular Partition **/
private static int partition_bk(int[] a, int low, int high) {

    int pivot = a[low];

    while (low < high) {

        ++count;

        while (low < high && a[high] >= pivot) {

            ++count;

            --high;
        }

        a[low] = a[high];


        while (low < high && a[low] <= pivot) {

            ++count;

            ++low;
        }
```

```
            a[high] = a[low]; //swap

        }

        a[low] = pivot; // low/high is the right position of pivot
        return low; // index of pivot
    }

    /** Hoare - Partition **/
    private static int partition(int[] A, int p, int r) {

        int x = A[p];

        int i = p - 1;
        int j = r + 1;

        while (true) {

            count++;

            do {
                i++;
                count++;
            } while (A[i] < x);

            do {
                j--;
                count++;
            } while (A[j] > x);

            if (i < j) {
                int tmp = A[i];
                A[i] = A[j];
                A[j] = tmp;
            } else return j;
        }

    }

}
```