CS162

ASSIGNMENT 9

NAME:

ARCHIT AGRAWAL

ROLL NO. :

202052307

SECTION:

A

Question

1. **Code and compare the running time of Quick and Randomised Quick and Counting sort.**

   Do as directed :

   a. Take an array of at least 100 numbers.
   b. Implement Quick and Randomised Quick and Counting sort.
   c. Sort them using Quick and Randomised Quick and Counting sort.
   d. Make the following measurements :
      i. Time for sorting when numbers are in random order.
      ii. Time for sorting when numbers are in ascending order.
      iii. Time for sorting when numbers are in descending order.
   e. Repeat these measurements for 10 times and take out the average value.
   f. Fill the table below with your readings and attach it with your submission file.

Note: Do not use any built in sorting libraries.

S.No.	Sorting Algorithm	Theoretical Time Complexity	Number of elements in the Array	Average Time (in ms) Random	Average Time (in ms) Ascending	Average Time (in ms) Descending
1.	Quick Sort					
2.	Randomised Quick Sort					
3.	Counting Sort					

Description -: The following program asks the user to input the size of array and asks to decide in what order(ascending, descending or random) the user wants to pass the array to sorting methods (quick sort, randomised quick sort, counting sort).

The array is generated using random function, hence it is not sorted.

- If the user wants a random sorted to be passed to the sorting methods(quick sort, randomised quick sort, counting sort), this array can directly be passed.

- If the user wants an ascending sorted array, this array will be first sorted in ascending order and then it will be passed to the sorting methods.
- If the user wants a descending sorted array, this array will be first sorted in descending order and then it will be passed to the sorting methods.

The program will calculate time taken to sort the array for quick sort, randomised quick sort and counting sort 10 times and gives the output as average time taken to do so for all the sorting methods.

CODE

```
/* This code is written to compare the average time for quick sort, randomised
 quick sort and counting sort algorithms
   You will be asked to enter the size of array as input
   You will be asked to decide the order of input array
   an array will be generated randomly of that size and time will be calculate
d for all the algorithms (this will take place 10 times)
   the average of the time for all the methods will be calculated and printed
 */
import java.util.*;
import java.util.Random;
import java.util.Arrays;
import java.util.Collections;

public class SortingMethods{

    public static void swap(Integer arr[], int i, int j){
        int temp = arr[j];
        arr[j] = arr[i];
        arr[i] = temp;
    }

    public static int partition(Integer arr[], int l, int h){
        int pivot = arr[l];

        int i = h + 1;
```

```
        for(int j = h; j > l; j--){
            if(arr[j] > pivot){
                i--;
                swap(arr, i, j);
            }
        }

        swap(arr, i - 1, l);
        return i - 1;
    }

    public static void quickSort(Integer arr[], int low, int high){

        if(low < high){
            int x = partition(arr, low, high);
            quickSort(arr, low, x - 1);
            quickSort(arr, x + 1, high);
        }

    }

    public static void random(Integer arr[], int low, int high){

        Random rand = new Random();
        int pivot = rand.nextInt(high - low) + low;

        swap(arr, pivot, high);
    }

    public static int partitionRandom(Integer arr[], int l, int h){

        random(arr, l, h);
        int pivot = arr[l];

        int i = h + 1;

        for(int j = h; j > l; j--){
            if(arr[j] > pivot){
                i--;
                swap(arr, i, j);
            }
        }

        swap(arr, i - 1, l);
        return i - 1;
    }
```

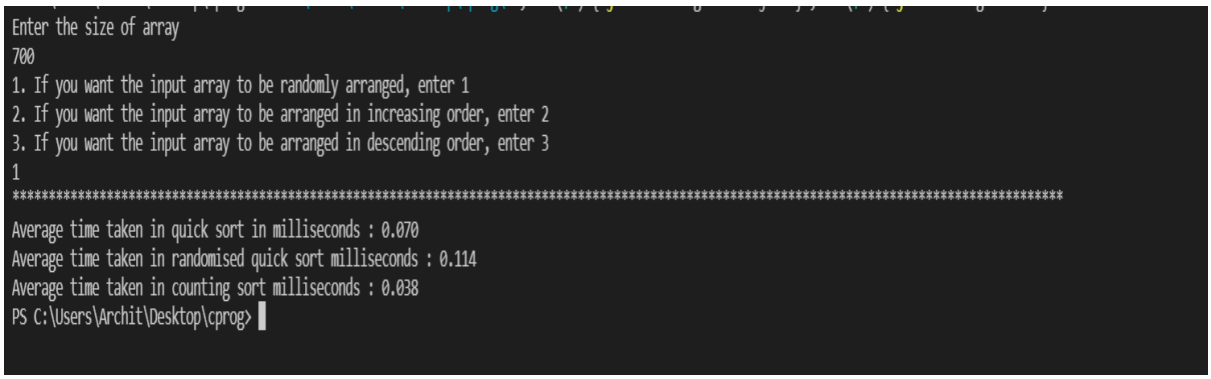
```
    public static double randomisedQuickSort(Integer arr[], int low, int high)
{
        double start = System.nanoTime();

        if(low < high){
            int x = partitionRandom(arr, low, high);
            randomisedQuickSort(arr, low, x - 1);
            randomisedQuickSort(arr, x + 1, high);
        }

        double end = System.nanoTime();
        return (end - start)/1000000.0;
    }

    public static double countingSort(Integer[] arr){

        double start = System.nanoTime();
        int max = arr[0];
        int min = arr[0];

        for(int i : arr){
            if(i > max) max = i;
            if(i < min) min = i;
        }

        int range = max - min + 1;
        int[] freq = new int[range];

        for(int i : arr){
            freq[i - min]++;
        }

        for(int i = 1; i < range; i++){
            freq[i] += freq[i - 1];
        }

        int[] sortedArr = new int[arr.length];
        for(int i = arr.length - 1; i >= 0; i--){
            sortedArr[freq[arr[i] - min] - 1] = arr[i];
            freq[arr[i] - min]--;
        }

        for(int i = 0; i < arr.length; i++){
            arr[i] = sortedArr[i];
        }

        double end = System.nanoTime();
        return (end - start)/1000000.0;
```

```
    }

    public static void main(String []args){
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the size of array");
        int n = sc.nextInt();

        System.out.println("1. If you want the input array to be randomly arra
nged, enter 1");
        System.out.println("2. If you want the input array to be arranged in i
ncreasing order, enter 2");
        System.out.println("3. If you want the input array to be arranged in d
escending order, enter 3");
        int order = sc.nextInt();

        double avg_time_quick_sort = 0.0;
        double avg_time_randomised_quick_sort = 0.0;
        double avg_time_counting_sort = 0.0;

        int t = 1;
        while(t > 0) {  //while loop is used to run quick/randomised quick/cou
nting sort for different arrays and compute the average.
            Integer[] a = new Integer[n];
            Integer[] b = new Integer[n]; //a copy of a[]
            Integer[] c = new Integer[n]; //another copy of a[]
            Random rand = new Random();

            for (int i = 0; i < n; i++) {
                a[i] = rand.nextInt(9000) + 1000;
                //System.out.print(a[i] + " ");
            }
            for(int i = 0; i < n; i++){
                b[i] = a[i];
                c[i] = a[i];
            }
            //a[] will be quick sorted
            //b[] will be randomised quick sorted
            //c[] will be counting sorted
            // as random will give a new number everytime that is why
            //copies of a[] is created to ensure that both the sorting
            //methods gets the same array

            if (order == 2) {
                Arrays.sort(a);
                Arrays.sort(b);
            } else if (order == 3) {
                Arrays.sort(a, Collections.reverseOrder());
                Arrays.sort(b, Collections.reverseOrder());
```

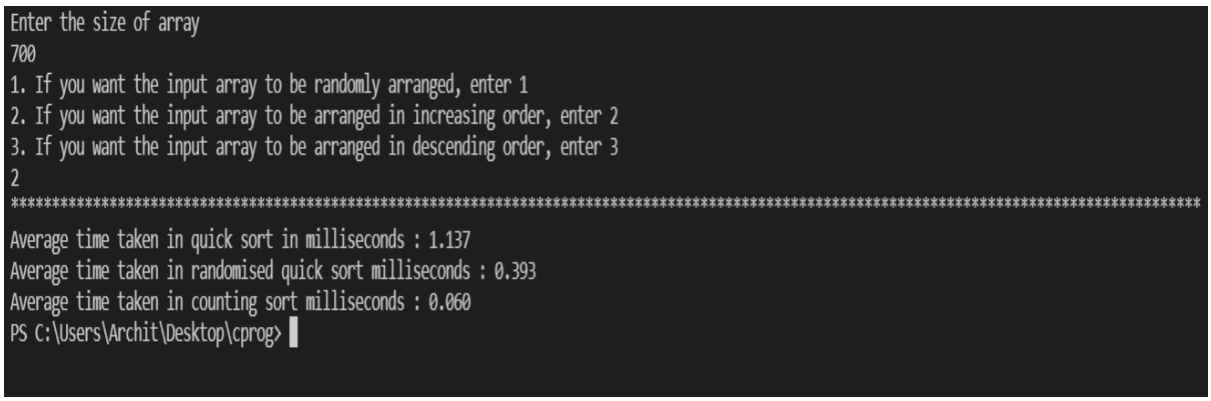
```
            }

            double start = System.nanoTime();
            quickSort(a, 0, a.length - 1);
            double end = System.nanoTime();

            double time_in_quick = (end - start)/1000000.0;

            start = System.nanoTime();
            randomisedQuickSort(b, 0, b.length - 1);
            end = System.nanoTime();

            double time_in_random_quick = (end - start)/1000000.0;

            double time_in_counting = countingSort(c);

            avg_time_quick_sort += time_in_quick;
            avg_time_randomised_quick_sort += time_in_random_quick;
            avg_time_counting_sort += time_in_counting;

            t--;
        }

        System.out.println("**********************************************
**************************************************************************
*****************");
        System.out.printf("Average time taken in quick sort in milliseconds :
%.3f ",(avg_time_quick_sort/10.0));
        System.out.println();
        System.out.printf("Average time taken in randomised quick sort millise
conds : %.3f ",(avg_time_randomised_quick_sort/10.0));
        System.out.println();
        System.out.printf("Average time taken in counting sort milliseconds :
%.3f ",(avg_time_counting_sort/10.0));
    }
}
```

OUTPUT

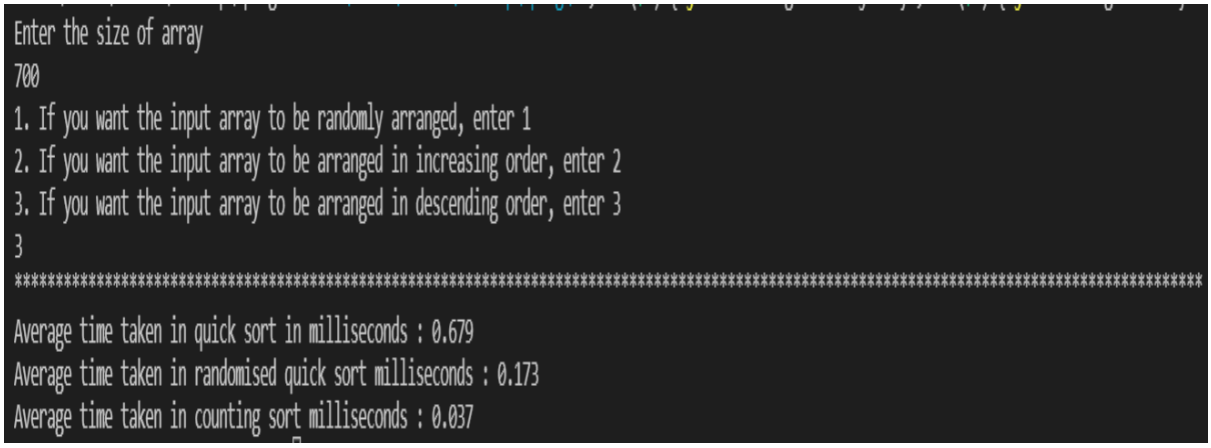
- ***For random ordered input array***



- ***For ascending ordered input array***



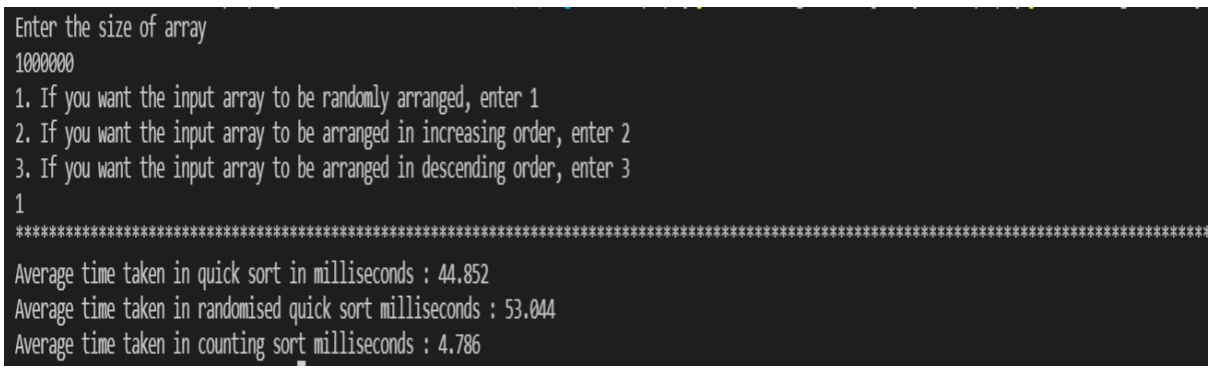
- ***For descending ordered input array***



The above data is tabularized below.

S.No.	Sorting Algorithm	Theoretical Time Complexity	Number of elements in the array	Average Time (in ms) for Random Order Input	Average Time (in ms) for Ascending Order Input	Average Time (in ms) for Descending Order Input
1.	Quick Sort	Worst = n^2 Avg = $n \log n$ Best = $n \log n$	700	0.070	1.137	0.679
2.	Randomised Quick Sort	Worst = n^2 Avg = $n \log n$ Best = $n \log n$	700	0.114	0.393	0.173
3.	Counting Sort	$O(n)$	700	0.038	0.060	0.037

Note **-:** Counting sort is most effective when there are a lot of numbers in a small range of numbers. For example, in the code provided above, the range of numbers was from 1000 to 10000. From the above table, it can be seen that there is not a very huge difference in time for all the algorithms when 700 numbers between 1000 to 10000 are sorted. Let us now take 10 lakh numbers between 1000 and 10000.



As you can observe, the time take by counting sort is almost 10 times lesser than time taken by quicksort when a large number of numbers are taken in a smaller range of numbers