CM2303 – Algorithms and Data Structure

A run-time analysis of insertion sort and counting sort

By Zak-Sum Yeung

Contents

Introduction 2
Pseudocode and Implementation 3
Testing 11
Experimental Setup and Results 22
    Insertion Sort 22
    Counting Sort 27
Conclusions and Discussion 29
Source Code 30

Introduction

The aim of this report is to be able to analyse and compare the theoretical run-time complexities compared to the actual run-time complexities of the two different sorting algorithms, 'Insertion Sort' and 'Counting Sort'. To find the real-time analysis of both sorting algorithms, we would need to work out the run-time, and find out the best case, worst case for Insertion Sort as well as the average case for Counting Sort.

To work out the run-time and being able to analyse our data, we would need to use the 'Big O' notation to describe the relationship between the time, T(n), taken by either of the two algorithms and the number of inputs, n. For Insertion Sort, the best-case input has a linear run- time of O(n), where the first remaining element of the input compares to the last element of the sorted array. However, compared to the worst case, the array is in reverse order, where in each set, each element is the smallest of the elements before it. It also has a quadratic run- time of $O(n^2)$, where it will repeat the inner loop by scanning and moving the sorted section of the array before inserting the next element. And as for the average case, it is also quadratic, causing a disadvantage for sorting large arrays as it is almost as bad as the worst case.

On the other hand, for Counting Sort, it is just made of simple 'for loops' making it more straightforward to analyse as the run- time is linear in the number of items. When k is small, it fields a good performance as to when k is large, then it yields a bad performance, with the time complexity of each Best and Worst case is at O(n+k). The important property of Counting sort is that is stable, where the numbers with the same values appear in the output array in the same order that they do the input array.

Case	Insertion Sort	Counting Sort
Best	O(n)	O(n + k)
Average	$O(n^2)$	O(n + k)
Worst	$O(n^2)$	O(n + k)

Pseudocode and Implementation

Insertion Sort (Pseudocode)

Algorithm Routine

1. Select the next element and compare the previous element.
2. If selected element is smaller than previous element, then swap. Else move onto the next element.
3. Repeat the second step until all elements in the list are in order.
4. End of Algorithm

Algorithm insertionSort(A, n)
Input: an array A storing n integers
Output: array A sorted in non–descending order
for $i \leftarrow 1$ **to** $(n-1)$ **do**
  $item \leftarrow A[i]$
  $j \leftarrow i-1$
  while $j \geq 0$ **and** $A[j] > item$ **do**
    $A[j+1] \leftarrow A[j]$
    $j \leftarrow j-1$
  end while
  $A[j+1] \leftarrow item$
end for

Insertion Sort (Implementation)

```java
//ALGORITHM FOR INSERTION SORT//
public static void insertionSort (int a[], int n) {
  long insertionStart = System.nanoTime();
  for(int i = 1; i<n; i++) {
    int item = a[i];
    int j = i-1;
    while (j>=0 && a[j]>item) {
      a[j+1] = a[j];
      j = j-1;
    }
    a[j+1] = item;
  }
  long insertionEnd =  System.nanoTime();
}
```

Counting Sort (Pseudocode)

Algorithm Routine

1. Create an array C for frequencies, index from 0 to k
2. Fill 0 to array C
3. Count the frequencies of elements in A and store the result in the C Array
4. Modify C, where it stores the occurrences in the output array
5. Copy elements in A
6. Output B array with proper position (output array starts at index 0)

Algorithm countingSort(A, B, n, k)
Input: array A, with n elements, each with value from 0 to $k-1$
Output: sorted array B
for $i \leftarrow 0$ **to** $k-1$ **do**
  $C[i] \leftarrow 0$
end for
for $j \leftarrow 0$ **to** $n-1$ **do**
  $C[A[j]] \leftarrow C[A[j]] + 1$
end for
for $i \leftarrow 1$ **to** $k-1$ **do**
  $C[i] \leftarrow C[i] + C[i-1]$
end for
for $j \leftarrow n-1$ **downto** 0 **do**
  $B[C[A[j]]-1] \leftarrow A[j]$
  $C[A[j]] \leftarrow C[A[j]] - 1$
end for

Counting Sort Max value (Pseudocode)

getMax (array)
Input: Pass in array 'array' to get the max value
Output: return integer max
Initialize the first value of the array as max
max ← input array [0]
For i ← 1 to i← length of array input do
    If array input [i] > max
        set max = array input[i]
    End if
End For

Counting Sort (Implementation)

```
//ALGORITHM FOR COUNTING SORT//
public static void countingSort (int a[], int b[], int n, int k) {
  int[] c = new int[k+1];
  for(int j = 0; j<(n); j++) {
    c[a[j]] = c[a[j]]+1;
  }
  for(int i=1; i<(k); i++) {
    c[i] = c[i]+c[i-1];
  }
  for(int j = (n-1);j<=0; j--) {
    b[c[a[j]]-1] = a[j];
    c[a[j]] = c[a[j]]-1;
  }
}

  public static int getMax(int[] arr_input){
      //set first value in array as the max
      int max = arr_input[0];
      for(int i = 1;i<arr_input.length;i++){
          //if the value in the array is bigger, set new value as max
          if(arr_input[i]>max){
              max = arr_input[i];
          }
      }
      return max; //return the max integer
  }
```

Procedures used to generate input data

Random Array (Pseudocode)

getRandom(n)
Input: create an array length of ‘n’ integers
Output: random array of integers created with length ‘n’
Create new array of integers ‘list’ of length ‘n’
For i← 0 to i← length of the array ‘list’
    list[i] ← random number between * n
End For

Random Array (Implementation)

```
public static int[] getRandom(int n) {  //pass the lenght of array to be created
//create a new list of array with bunch of random 'n' integers
int[] list = new int[n];
for (int i = 0; i<list.length; i++){
    //fill the new array with random integers
    list[i] = (int)(Math.random()*n);
}
    return list;
}
```

Sorted Array (Pseudocode)

Input: A list of unsorted integer arrays filled with random integers.
Output: A sorted array of random integers in ascending order
sort(array, ’a’)

Sorted Array (Implementation)

```
public static int[] sort(int[] a){
//List of unsorted random integers
Arrays.sort(a); //Sort array 'a'
//Return a list of sroted arrays in ascending order
return a;
}
```

Reverse Sort (Pseudocode)

Input: List of unsorted random arrays to be sorted.
Output: List of array sorted in descending order
For i← 0 to i← length of array
    For j← i + 1 to j ← length of array
        If array[i] < array[j]
            Temp = array[i]
            array[i] = array[j]
            array[j] = temp
        End If
    End For
End for

Reverse Sort (Implementation)

```
public static int[] reversesort(int[] b){
    //Create a new array of reversed integers with a length of input arrays
    for(int z = 0; z < (b.length/2); z++){
        int temp = b[z];
        //reverse array
        b[z] = b[b.length -1 -z];
        b[b.length -1 - z] = temp;
    }
    return b;
}
```

Insertion Sort Average Case (Implementation)

```
//create a loop of array length starting from 1000 to 20000, incrementing by 1000
for (int value_avg = 1000; value_avg<=20000; value_avg+=1000) {
// for (int value_avg = 5; value_avg<=10; value_avg+=5) {
    System.out.println(value_avg);
    //create a loop of data points from 1-5
    for (int h = 1; h<=5; h++) {
        System.out.println(h);
        int[] rand_arr = getRandom(value_avg); //declaring array in average case
        int n = rand_arr.length;
        //using nanoTime to get the current time
        long insertionStart = System.nanoTime();
        //Run the insertion method
        insertionSort(rand_arr, n);
        long insertionEnd =  System.nanoTime();
        //Find the time taken and records the time for the algorithm
        System.out.println("Average_Insertion_Run-time "+ (insertionEnd - insertionStart));
    }
}
```

Insertion Sort Best Case (Implementation)

```
//create a loop of array length starting from 1000 to 20000, incrementing by 1000
for (int value_best = 1000; value_best<=20000; value_best+=1000) {
// for (int value_best = 5; value_best<=10; value_best+=5) {
        System.out.println(value_best);
        //create a loop of data points from 1-5
        for (int h = 1; h<=5; h++) {
            System.out.println(h);
            int[] rand_arr = getRandom(value_best); //declaring array in best case
            int n = rand_arr.length;
            //using nanoTime to get the current time
            Arrays.sort(rand_arr);
            long insertionStart = System.nanoTime();
            //Run the insertion method
            insertionSort(rand_arr, n);
            long insertionEnd =  System.nanoTime();
            //Find the time taken and records the time for the algorithm
            System.out.println("Best_Insertion_Run-time "+(insertionEnd - insertionStart));
        }
    }
```

Insertion Sort Worst Case (Implementation)

```
    int[] randomReverseSorted_array;//initialising the array
    //create a loop of array length starting from 1000 to 20000, incrementing by 1000
    for (int value_worst = 1000; value_worst<=20000; value_worst+=1000) {
    // for (int value_worst = 5; value_worst<=10; value_worst+=5) {
        System.out.println(value_worst);
        //create a loop of data points from 1-5
        for (int h = 1; h<=5; h++) {
            System.out.println(h);
            int[] rand_arr = getRandom(value_worst); //declaring array in worst case
            //Reverse the unsorted array to make it sorted in reverse descending order
            randomReverseSorted_array = reversesort(rand_arr);
            int n = rand_arr.length;
            //using nanoTime to get the current time
            long insertionStart = System.nanoTime();
            insertionSort(randomReverseSorted_array, n);
            long insertionEnd =  System.nanoTime();
            //Find the time taken and records the time for the algorithm
            System.out.println("Worst_Insertion_Run-time "+(insertionEnd - insertionStart));
        }
    }
}
```

Counting Sort Bad Performance (Implementation)

```
//Generate data for Counting Worse Case//
public static void countingTest_worstcase() {
    int[] rand_arr ;
    //create a loop of array length starting from 1000 to 20000, incrementing by 1000
    for (int value_worst_C = 1000; value_worst_C<=20000; value_worst_C+=1000) {
    // for (int value_worst_C = 5; value_worst_C<=5; value_worst_C+=5) {
        System.out.println(value_worst_C);
        //create a loop of data points from 1-5
        for (int h = 1; h<=5; h++) {
          System.out.println(h);
            rand_arr = getRandom(value_worst_C); //declaring array in worst counting case
            //Empty array for storing random integers
            int[] empty_array = new int[value_worst_C];
            //Length of array is calculated
            int n = rand_arr.length;
            int k_value = (n*n/*+getMax(rand_arr)*//*Get max value*/);
            //using nanoTime to get the current time
            long countingStart = System.nanoTime();
            countingSort(rand_arr, empty_array, n, k_value); //Run mehtod for bad performance
            long countingEnd =  System.nanoTime();
            //Find the time taken and records the time for the algorithm
            System.out.println("Worst_Counting_Run-time "+(countingEnd - countingStart));
        }
    }
}
```

Counting Sort Good Performance (Implementation)

```
//Generate data for Counting Best Case//
public static void countingTest_bestcase() {
    int[] rand_arr ;
    //create a loop of array length starting from 1000 to 20000, incrementing by 1000
    for (int value_best_C = 1000; value_best_C<=20000; value_best_C+=1000) {
    // for (int value_best_C = 5; value_best_C<=5; value_best_C+=5) {
        System.out.println(value_best_C);
        //create a loop of data points from 1-5
        for (int h = 1; h<=5; h++) {
            System.out.println(h);
            rand_arr = getRandom(value_best_C); //declaring array in best counting case
            //Empty array for storing random integers
            int[] empty_array = new int[value_best_C];
            //Length of array is calculated
            int n = rand_arr.length;
            int k_value = getMax(rand_arr); //getting the highest number in the array 'rand_arr'
            //using nanoTime to get the current time
            long countingStart = System.nanoTime();
            countingSort(rand_arr, empty_array, n, k_value); //run method for Good performance
            long countingEnd = System.nanoTime();
            //Find the time taken and records the time for the algorithm
            System.out.println("Best_Counting_Run-time "+(countingEnd - countingStart));
        }
    }
}
```

Nano speed Timing Method (System.nanoTime())

```
//Using 'System.nanoTime()' to get the current run-time
//Gets the system time in Nano seconds and assigns it to the start time
long countingStart = System.nanoTime();
Sort_algorithm(a,n); //run method for performance
//Gets system time after running algorithm method
long countingEnd = System.nanoTime();
//Find the time taken and records the time for the algorithm
System.out.println("Run-time "+(algEnd - algStart)/*Find the actual run-time*/);
```

When trying to find the run-time of your best/average/worst case for each sorting algorithm, I decided to use the function provided by Java, 'System.nanoTime()' to be able to give me a precise calculation of the run-time in Nanoseconds. To get the best result of your run-time, I decided to place this function just before compiling my method as seen in the above codes, and placing it just after the method has been calculated to avoid extra nanoseconds being added and affecting the results and losing efficiency.

When the start and the end time has been recorded, I will need to calculate the run-time by finding the difference between the end time and the start time, as shown above.

However, by using this function, it also affected the accuracy of my results due to the which operating system was running as well as the type of processing power speed. For example, from using 'Microsoft Windows OS', it would have many background applications running even when the windows have been closed or from ending the tasks, which would vary the measurements of my results.

Testing

When testing my Algorithm, I have tested for the Best, Average and Worst Case for Insertion sort of 3 different array sizes, ranging from 5 to 15 where it increments 5 each time is runs for Insertion Sort. As for Counting Sort Worst Performance, I will also have 3 different array sizes, starting from 3 to 9 where it increments by 3 each time. And as for Counting Sort Best Performance, I will also be starting from 5 to 15 where it increments by 5 each time. I decided to run each test 5 times for each different array size to ensure that this test is done fairly showing zero errors occurring. As these tests are all random numbers, it would make a non- biased test therefore showing an honest result.

Insertion Sort Average Case

Array Length: 5	Array Length: 10	Array Length: 15
Test: 1	Test: 1	Test: 1
Input: [2, 1, 3, 0, 2]	Input: [3, 7, 3, 2, 9, 5, 2, 3, 3, 7]	Input: [2, 11, 8, 7, 6, 14, 0, 12, 0, 5, 10, 2, 14, 0, 13]
[2, 2, 3, 0, 2]	[3, 7, 3, 2, 9, 5, 2, 3, 3, 7]	[2, 11, 8, 7, 6, 14, 0, 12, 0, 5, 10, 2, 14, 0, 13]
[1, 2, 3, 0, 2]	[3, 7, 7, 2, 9, 5, 2, 3, 3, 7]	[2, 11, 11, 7, 6, 14, 0, 12, 0, 5, 10, 2, 14, 0, 13]
[1, 1, 2, 3, 2]	[3, 3, 3, 7, 9, 5, 2, 3, 3, 7]	[2, 8, 8, 11, 6, 14, 0, 12, 0, 5, 10, 2, 14, 0, 13]
[0, 1, 2, 3, 3]	[2, 3, 3, 7, 9, 5, 2, 3, 3, 7]	[2, 7, 7, 8, 11, 14, 0, 12, 0, 5, 10, 2, 14, 0, 13]
Result: [0, 1, 2, 2, 3]	[2, 3, 3, 7, 7, 9, 2, 3, 3, 7]	[2, 6, 7, 8, 11, 14, 0, 12, 0, 5, 10, 2, 14, 0, 13]
Test: 2	[2, 3, 3, 3, 5, 7, 9, 3, 3, 7]	[2, 2, 6, 7, 8, 11, 14, 12, 0, 5, 10, 2, 14, 0, 13]
Input: [2, 3, 0, 1, 4]	[2, 2, 3, 3, 5, 5, 7, 9, 3, 7]	[0, 2, 6, 7, 8, 11, 14, 14, 0, 5, 10, 2, 14, 0, 13]
[2, 3, 0, 1, 4]	[2, 2, 3, 3, 3, 5, 5, 7, 9, 7]	[0, 2, 2, 6, 7, 8, 11, 12, 14, 5, 10, 2, 14, 0, 13]
[2, 2, 3, 1, 4]	[2, 2, 3, 3, 3, 3, 5, 7, 9, 9]	[0, 0, 2, 6, 6, 7, 8, 11, 12, 14, 10, 2, 14, 0, 13]
[0, 2, 2, 3, 4]	Result: [2, 2, 3, 3, 3, 3, 5, 7, 7, 9]	[0, 0, 2, 5, 6, 7, 8, 11, 11, 12, 14, 2, 14, 0, 13]
[0, 1, 2, 3, 4]	Test: 2	[0, 0, 2, 5, 5, 6, 7, 8, 10, 11, 12, 14, 14, 0, 13]
Result: [0, 1, 2, 3, 4]	Input: [7, 1, 3, 7, 7, 8, 3, 2, 4, 7]	[0, 0, 2, 2, 5, 6, 7, 8, 10, 11, 12, 14, 14, 0, 13]
Test: 3	[7, 7, 3, 7, 7, 8, 3, 2, 4, 7]	[0, 0, 2, 2, 2, 5, 6, 7, 8, 10, 11, 12, 14, 14, 13]
Input: [2, 2, 1, 0, 4]	[1, 7, 7, 7, 7, 8, 3, 2, 4, 7]	[0, 0, 0, 2, 2, 5, 6, 7, 8, 10, 11, 12, 14, 14, 14]
[2, 2, 1, 0, 4]	[1, 3, 7, 7, 7, 8, 3, 2, 4, 7]	Result: [0, 0, 0, 2, 2, 5, 6, 7, 8, 10, 11, 12, 13, 14, 14]
[2, 2, 2, 0, 4]	[1, 3, 7, 7, 7, 8, 3, 2, 4, 7]	Test: 2
[1, 1, 2, 2, 4]	[1, 3, 7, 7, 7, 8, 3, 2, 4, 7]	Input: [9, 6, 11, 9, 12, 12, 8, 1, 11, 8, 1, 0, 8, 3, 4]
[0, 1, 2, 2, 4]	[1, 3, 7, 7, 7, 7, 8, 2, 4, 7]	[9, 9, 11, 9, 12, 12, 8, 1, 11, 8, 1, 0, 8, 3, 4]
Result: [0, 1, 2, 2, 4]	[1, 3, 3, 3, 7, 7, 7, 8, 4, 7]	[6, 9, 11, 9, 12, 12, 8, 1, 11, 8, 1, 0, 8, 3, 4]
Test: 4	[1, 2, 3, 3, 7, 7, 7, 7, 8, 7]	[6, 9, 11, 11, 12, 12, 8, 1, 11, 8, 1, 0, 8, 3, 4]
Input: [3, 2, 1, 1, 0]	[1, 2, 3, 3, 4, 7, 7, 7, 8, 8]	[6, 9, 9, 11, 12, 12, 8, 1, 11, 8, 1, 0, 8, 3, 4]
[3, 3, 1, 1, 0]	Result: [1, 2, 3, 3, 4, 7, 7, 7, 7, 8]	[6, 9, 9, 11, 12, 12, 8, 1, 11, 8, 1, 0, 8, 3, 4]
[2, 2, 3, 1, 0]	Test: 3	[6, 9, 9, 9, 11, 12, 12, 1, 11, 8, 1, 0, 8, 3, 4]
[1, 2, 2, 3, 0]	Input: [6, 1, 7, 5, 3, 9, 5, 0, 1, 5]	[6, 6, 8, 9, 9, 11, 12, 12, 11, 8, 1, 0, 8, 3, 4]
[1, 1, 1, 2, 3]	[6, 6, 7, 5, 3, 9, 5, 0, 1, 5]	[1, 6, 8, 9, 9, 11, 12, 12, 12, 8, 1, 0, 8, 3, 4]
Result: [0, 1, 1, 2, 3]	[1, 6, 7, 5, 3, 9, 5, 0, 1, 5]	[1, 6, 8, 9, 9, 9, 11, 11, 12, 12, 1, 0, 8, 3, 4]
Test: 5	[1, 6, 6, 7, 3, 9, 5, 0, 1, 5]	[1, 6, 6, 8, 8, 9, 9, 11, 11, 12, 12, 0, 8, 3, 4]
Input: [3, 3, 3, 3, 0]	[1, 5, 5, 6, 7, 9, 5, 0, 1, 5]	[1, 1, 1, 6, 8, 8, 9, 9, 11, 11, 12, 12, 8, 3, 4]
[3, 3, 3, 3, 0]	[1, 3, 5, 6, 7, 9, 5, 0, 1, 5]	[0, 1, 1, 6, 8, 8, 9, 9, 9, 11, 11, 12, 12, 3, 4]
[3, 3, 3, 3, 0]	[1, 3, 5, 6, 6, 7, 9, 0, 1, 5]	[0, 1, 1, 6, 6, 8, 8, 8, 9, 9, 11, 11, 12, 12, 4]
[3, 3, 3, 3, 0]	[1, 1, 3, 5, 5, 6, 7, 9, 1, 5]	[0, 1, 1, 3, 6, 6, 8, 8, 8, 9, 9, 11, 11, 12, 12]

[3, 3, 3, 3, 3] Result: [0, 3, 3, 3, 3]	[0, 1, 3, 3, 5, 5, 6, 7, 9, 5] [0, 1, 1, 3, 5, 5, 6, 6, 7, 9] Result: [0, 1, 1, 3, 5, 5, 5, 6, 7, 9] Test: 4 Input: [8, 6, 5, 7, 3, 3, 7, 4, 6, 6] [8, 8, 5, 7, 3, 3, 7, 4, 6, 6] [6, 6, 8, 7, 3, 3, 7, 4, 6, 6] [5, 6, 8, 8, 3, 3, 7, 4, 6, 6] [5, 5, 6, 7, 8, 3, 7, 4, 6, 6] [3, 5, 5, 6, 7, 8, 7, 4, 6, 6] [3, 3, 5, 6, 7, 8, 8, 4, 6, 6] [3, 3, 5, 5, 6, 7, 7, 8, 6, 6] [3, 3, 4, 5, 6, 7, 7, 7, 8, 6] [3, 3, 4, 5, 6, 6, 7, 7, 7, 8] Result: [3, 3, 4, 5, 6, 6, 6, 7, 7, 8] Test: 5 Input: [9, 8, 7, 9, 9, 5, 6, 7, 8, 5] [9, 9, 7, 9, 9, 5, 6, 7, 8, 5] [8, 8, 9, 9, 9, 5, 6, 7, 8, 5] [7, 8, 9, 9, 9, 5, 6, 7, 8, 5] [7, 8, 9, 9, 9, 5, 6, 7, 8, 5] [7, 7, 8, 9, 9, 9, 6, 7, 8, 5] [5, 7, 7, 8, 9, 9, 9, 7, 8, 5] [5, 6, 7, 8, 8, 9, 9, 9, 8, 5] [5, 6, 7, 7, 8, 9, 9, 9, 9, 5] [5, 6, 6, 7, 7, 8, 8, 9, 9, 9] Result: [5, 5, 6, 7, 7, 8, 8, 9, 9, 9]	Result: [0, 1, 1, 3, 4, 6, 8, 8, 8, 9, 9, 11, 11, 12, 12] Test: 3 Input: [6, 3, 14, 9, 12, 4, 12, 9, 8, 12, 3, 11, 7, 11, 13] [6, 6, 14, 9, 12, 4, 12, 9, 8, 12, 3, 11, 7, 11, 13] [3, 6, 14, 9, 12, 4, 12, 9, 8, 12, 3, 11, 7, 11, 13] [3, 6, 14, 14, 12, 4, 12, 9, 8, 12, 3, 11, 7, 11, 13] [3, 6, 9, 14, 14, 4, 12, 9, 8, 12, 3, 11, 7, 11, 13] [3, 6, 6, 9, 12, 14, 12, 9, 8, 12, 3, 11, 7, 11, 13] [3, 4, 6, 9, 12, 14, 14, 9, 8, 12, 3, 11, 7, 11, 13] [3, 4, 6, 9, 12, 12, 12, 14, 8, 12, 3, 11, 7, 11, 13] [3, 4, 6, 9, 9, 9, 12, 12, 14, 12, 3, 11, 7, 11, 13] [3, 4, 6, 8, 9, 9, 12, 12, 14, 14, 3, 11, 7, 11, 13] [3, 4, 4, 6, 8, 9, 9, 12, 12, 12, 14, 11, 7, 11, 13] [3, 3, 4, 6, 8, 9, 9, 12, 12, 12, 12, 14, 7, 11, 13] [3, 3, 4, 6, 8, 8, 9, 9, 11, 12, 12, 12, 14, 11, 13] [3, 3, 4, 6, 7, 8, 9, 9, 11, 12, 12, 12, 12, 14, 13] [3, 3, 4, 6, 7, 8, 9, 9, 11, 11, 12, 12, 12, 14, 14] Result: [3, 3, 4, 6, 7, 8, 9, 9, 11, 11, 12, 12, 12, 13, 14] Test: 4 Input: [5, 9, 7, 3, 3, 10, 13, 9, 9, 3, 5, 12, 0, 8, 0] [5, 9, 7, 3, 3, 10, 13, 9, 9, 3, 5, 12, 0, 8, 0] [5, 9, 9, 3, 3, 10, 13, 9, 9, 3, 5, 12, 0, 8, 0] [5, 5, 7, 9, 3, 10, 13, 9, 9, 3, 5, 12, 0, 8, 0] [3, 5, 5, 7, 9, 10, 13, 9, 9, 3, 5, 12, 0, 8, 0] [3, 3, 5, 7, 9, 10, 13, 9, 9, 3, 5, 12, 0, 8, 0] [3, 3, 5, 7, 9, 10, 13, 9, 9, 3, 5, 12, 0, 8, 0] [3, 3, 5, 7, 9, 10, 10, 13, 9, 3, 5, 12, 0, 8, 0] [3, 3, 5, 7, 9, 9, 10, 10, 13, 3, 5, 12, 0, 8, 0] [3, 3, 5, 5, 7, 9, 9, 9, 10, 13, 5, 12, 0, 8, 0] [3, 3, 3, 5, 7, 7, 9, 9, 9, 10, 13, 12, 0, 8, 0] [3, 3, 3, 5, 5, 7, 9, 9, 9, 10, 13, 13, 0, 8, 0] [3, 3, 3, 3, 5, 5, 7, 9, 9, 9, 10, 12, 13, 8, 0] [0, 3, 3, 3, 5, 5, 7, 9, 9, 9, 9, 10, 12, 13, 0] [0, 3, 3, 3, 3, 5, 5, 7, 8, 9, 9, 9, 10, 12, 13] Result: [0, 0, 3, 3, 3, 5, 5, 7, 8, 9, 9, 9, 10, 12, 13] Test: 5 Input: [4, 2, 3, 11, 8, 3, 6, 14, 6, 13, 12, 13, 3, 14, 11] [4, 4, 3, 11, 8, 3, 6, 14, 6, 13, 12, 13, 3, 14, 11] [2, 4, 4, 11, 8, 3, 6, 14, 6, 13, 12, 13, 3, 14, 11] [2, 3, 4, 11, 8, 3, 6, 14, 6, 13, 12, 13, 3, 14, 11] [2, 3, 4, 11, 11, 3, 6, 14, 6, 13, 12, 13, 3, 14, 11] [2, 3, 4, 4, 8, 11, 6, 14, 6, 13, 12, 13, 3, 14, 11] [2, 3, 3, 4, 8, 8, 11, 14, 6, 13, 12, 13, 3, 14, 11] [2, 3, 3, 4, 6, 8, 11, 14, 6, 13, 12, 13, 3, 14, 11] [2, 3, 3, 4, 6, 8, 8, 11, 14, 13, 12, 13, 3, 14, 11] [2, 3, 3, 4, 6, 6, 8, 11, 14, 14, 12, 13, 3, 14, 11] [2, 3, 3, 4, 6, 6, 8, 11, 13, 13, 14, 13, 3, 14, 11] [2, 3, 3, 4, 6, 6, 8, 11, 12, 13, 14, 14, 3, 14, 11]

		[2, 3, 3, 4, 4, 6, 6, 8, 11, 12, 13, 13, 14, 14, 11] [2, 3, 3, 3, 4, 6, 6, 8, 11, 12, 13, 13, 14, 14, 11] [2, 3, 3, 3, 4, 6, 6, 8, 11, 12, 12, 13, 13, 14, 14] Result: [2, 3, 3, 3, 4, 6, 6, 8, 11, 11, 12, 13, 13, 14, 14]

Insertion Sort Best Case

Array Length: 5 Test: 1 Input: [1, 1, 2, 2, 3] [1, 1, 2, 2, 3] [1, 1, 2, 2, 3] [1, 1, 2, 2, 3] [1, 1, 2, 2, 3] Result: [1, 1, 2, 2, 3] Test: 2 Input: [0, 0, 1, 2, 3] [0, 0, 1, 2, 3] [0, 0, 1, 2, 3] [0, 0, 1, 2, 3] [0, 0, 1, 2, 3] Result: [0, 0, 1, 2, 3] Test: 3 Input: [1, 1, 2, 2, 3] [1, 1, 2, 2, 3] [1, 1, 2, 2, 3] [1, 1, 2, 2, 3] [1, 1, 2, 2, 3] Result: [1, 1, 2, 2, 3] Test: 4 Input: [1, 1, 2, 3, 4] [1, 1, 2, 3, 4] [1, 1, 2, 3, 4] [1, 1, 2, 3, 4] [1, 1, 2, 3, 4] Result: [1, 1, 2, 3, 4] Test: 5 Input: [0, 3, 3, 3, 3] [0, 3, 3, 3, 3] [0, 3, 3, 3, 3] [0, 3, 3, 3, 3] [0, 3, 3, 3, 3] Result: [0, 3, 3, 3, 3]	Array Length: 10 Test: 1 Input: [0, 2, 2, 3, 4, 4, 5, 5, 6, 8] [0, 2, 2, 3, 4, 4, 5, 5, 6, 8] [0, 2, 2, 3, 4, 4, 5, 5, 6, 8] [0, 2, 2, 3, 4, 4, 5, 5, 6, 8] [0, 2, 2, 3, 4, 4, 5, 5, 6, 8] [0, 2, 2, 3, 4, 4, 5, 5, 6, 8] [0, 2, 2, 3, 4, 4, 5, 5, 6, 8] [0, 2, 2, 3, 4, 4, 5, 5, 6, 8] [0, 2, 2, 3, 4, 4, 5, 5, 6, 8] [0, 2, 2, 3, 4, 4, 5, 5, 6, 8] Result: [0, 2, 2, 3, 4, 4, 5, 5, 6, 8] Test: 2 Input: [0, 3, 5, 5, 5, 6, 8, 8, 8, 9] [0, 3, 5, 5, 5, 6, 8, 8, 8, 9] [0, 3, 5, 5, 5, 6, 8, 8, 8, 9] [0, 3, 5, 5, 5, 6, 8, 8, 8, 9] [0, 3, 5, 5, 5, 6, 8, 8, 8, 9] [0, 3, 5, 5, 5, 6, 8, 8, 8, 9] [0, 3, 5, 5, 5, 6, 8, 8, 8, 9] [0, 3, 5, 5, 5, 6, 8, 8, 8, 9] [0, 3, 5, 5, 5, 6, 8, 8, 8, 9] [0, 3, 5, 5, 5, 6, 8, 8, 8, 9] Result: [0, 3, 5, 5, 5, 6, 8, 8, 8, 9] Test: 3 Input: [0, 0, 5, 5, 6, 6, 7, 9, 9, 9] [0, 0, 5, 5, 6, 6, 7, 9, 9, 9] [0, 0, 5, 5, 6, 6, 7, 9, 9, 9] [0, 0, 5, 5, 6, 6, 7, 9, 9, 9] [0, 0, 5, 5, 6, 6, 7, 9, 9, 9] [0, 0, 5, 5, 6, 6, 7, 9, 9, 9] [0, 0, 5, 5, 6, 6, 7, 9, 9, 9] [0, 0, 5, 5, 6, 6, 7, 9, 9, 9] [0, 0, 5, 5, 6, 6, 7, 9, 9, 9] [0, 0, 5, 5, 6, 6, 7, 9, 9, 9] Result: [0, 0, 5, 5, 6, 6, 7, 9, 9, 9] Test: 4	Array Length: 15 Test: 1 Input: [0, 2, 2, 2, 3, 4, 4, 4, 6, 7, 8, 8, 8, 13, 14] [0, 2, 2, 2, 3, 4, 4, 4, 6, 7, 8, 8, 8, 13, 14] [0, 2, 2, 2, 3, 4, 4, 4, 6, 7, 8, 8, 8, 13, 14] [0, 2, 2, 2, 3, 4, 4, 4, 6, 7, 8, 8, 8, 13, 14] [0, 2, 2, 2, 3, 4, 4, 4, 6, 7, 8, 8, 8, 13, 14] [0, 2, 2, 2, 3, 4, 4, 4, 6, 7, 8, 8, 8, 13, 14] [0, 2, 2, 2, 3, 4, 4, 4, 6, 7, 8, 8, 8, 13, 14] [0, 2, 2, 2, 3, 4, 4, 4, 6, 7, 8, 8, 8, 13, 14] [0, 2, 2, 2, 3, 4, 4, 4, 6, 7, 8, 8, 8, 13, 14] [0, 2, 2, 2, 3, 4, 4, 4, 6, 7, 8, 8, 8, 13, 14] [0, 2, 2, 2, 3, 4, 4, 4, 6, 7, 8, 8, 8, 13, 14] [0, 2, 2, 2, 3, 4, 4, 4, 6, 7, 8, 8, 8, 13, 14] [0, 2, 2, 2, 3, 4, 4, 4, 6, 7, 8, 8, 8, 13, 14] [0, 2, 2, 2, 3, 4, 4, 4, 6, 7, 8, 8, 8, 13, 14] [0, 2, 2, 2, 3, 4, 4, 4, 6, 7, 8, 8, 8, 13, 14] Result: [0, 2, 2, 2, 3, 4, 4, 4, 6, 7, 8, 8, 8, 13, 14] Test: 2 Input: [1, 1, 2, 3, 3, 3, 4, 4, 6, 7, 7, 12, 13, 14, 14] [1, 1, 2, 3, 3, 3, 4, 4, 6, 7, 7, 12, 13, 14, 14] [1, 1, 2, 3, 3, 3, 4, 4, 6, 7, 7, 12, 13, 14, 14] [1, 1, 2, 3, 3, 3, 4, 4, 6, 7, 7, 12, 13, 14, 14] [1, 1, 2, 3, 3, 3, 4, 4, 6, 7, 7, 12, 13, 14, 14] [1, 1, 2, 3, 3, 3, 4, 4, 6, 7, 7, 12, 13, 14, 14] [1, 1, 2, 3, 3, 3, 4, 4, 6, 7, 7, 12, 13, 14, 14] [1, 1, 2, 3, 3, 3, 4, 4, 6, 7, 7, 12, 13, 14, 14] [1, 1, 2, 3, 3, 3, 4, 4, 6, 7, 7, 12, 13, 14, 14] [1, 1, 2, 3, 3, 3, 4, 4, 6, 7, 7, 12, 13, 14, 14] [1, 1, 2, 3, 3, 3, 4, 4, 6, 7, 7, 12, 13, 14, 14] [1, 1, 2, 3, 3, 3, 4, 4, 6, 7, 7, 12, 13, 14, 14] [1, 1, 2, 3, 3, 3, 4, 4, 6, 7, 7, 12, 13, 14, 14] [1, 1, 2, 3, 3, 3, 4, 4, 6, 7, 7, 12, 13, 14, 14] [1, 1, 2, 3, 3, 3, 4, 4, 6, 7, 7, 12, 13, 14, 14] Result: [1, 1, 2, 3, 3, 3, 4, 4, 6, 7, 7, 12, 13, 14, 14] Test: 3 Input: [0, 1, 2, 2, 3, 3, 4, 4, 6, 7, 7, 9, 9, 11, 12] [0, 1, 2, 2, 3, 3, 4, 4, 6, 7, 7, 9, 9, 11, 12]

	Input: [0, 1, 1, 2, 3, 4, 4, 4, 7, 8] [0, 1, 1, 2, 3, 4, 4, 4, 7, 8] [0, 1, 1, 2, 3, 4, 4, 4, 7, 8] [0, 1, 1, 2, 3, 4, 4, 4, 7, 8] [0, 1, 1, 2, 3, 4, 4, 4, 7, 8] [0, 1, 1, 2, 3, 4, 4, 4, 7, 8] [0, 1, 1, 2, 3, 4, 4, 4, 7, 8] [0, 1, 1, 2, 3, 4, 4, 4, 7, 8] [0, 1, 1, 2, 3, 4, 4, 4, 7, 8] [0, 1, 1, 2, 3, 4, 4, 4, 7, 8] Result: [0, 1, 1, 2, 3, 4, 4, 4, 7, 8] Test: 5 Input: [2, 2, 3, 3, 3, 5, 6, 7, 7, 8] [2, 2, 3, 3, 3, 5, 6, 7, 7, 8] [2, 2, 3, 3, 3, 5, 6, 7, 7, 8] [2, 2, 3, 3, 3, 5, 6, 7, 7, 8] [2, 2, 3, 3, 3, 5, 6, 7, 7, 8] [2, 2, 3, 3, 3, 5, 6, 7, 7, 8] [2, 2, 3, 3, 3, 5, 6, 7, 7, 8] [2, 2, 3, 3, 3, 5, 6, 7, 7, 8] [2, 2, 3, 3, 3, 5, 6, 7, 7, 8] [2, 2, 3, 3, 3, 5, 6, 7, 7, 8] Result: [2, 2, 3, 3, 3, 5, 6, 7, 7, 8]	[0, 1, 2, 2, 3, 3, 4, 4, 6, 7, 7, 9, 9, 11, 12] [0, 1, 2, 2, 3, 3, 4, 4, 6, 7, 7, 9, 9, 11, 12] [0, 1, 2, 2, 3, 3, 4, 4, 6, 7, 7, 9, 9, 11, 12] [0, 1, 2, 2, 3, 3, 4, 4, 6, 7, 7, 9, 9, 11, 12] [0, 1, 2, 2, 3, 3, 4, 4, 6, 7, 7, 9, 9, 11, 12] [0, 1, 2, 2, 3, 3, 4, 4, 6, 7, 7, 9, 9, 11, 12] [0, 1, 2, 2, 3, 3, 4, 4, 6, 7, 7, 9, 9, 11, 12] [0, 1, 2, 2, 3, 3, 4, 4, 6, 7, 7, 9, 9, 11, 12] [0, 1, 2, 2, 3, 3, 4, 4, 6, 7, 7, 9, 9, 11, 12] [0, 1, 2, 2, 3, 3, 4, 4, 6, 7, 7, 9, 9, 11, 12] [0, 1, 2, 2, 3, 3, 4, 4, 6, 7, 7, 9, 9, 11, 12] [0, 1, 2, 2, 3, 3, 4, 4, 6, 7, 7, 9, 9, 11, 12] [0, 1, 2, 2, 3, 3, 4, 4, 6, 7, 7, 9, 9, 11, 12] Result: [0, 1, 2, 2, 3, 3, 4, 4, 6, 7, 7, 9, 9, 11, 12] Test: 4 Input: [0, 0, 0, 2, 2, 3, 6, 8, 9, 9, 10, 11, 13, 14, 14] [0, 0, 0, 2, 2, 3, 6, 8, 9, 9, 10, 11, 13, 14, 14] [0, 0, 0, 2, 2, 3, 6, 8, 9, 9, 10, 11, 13, 14, 14] [0, 0, 0, 2, 2, 3, 6, 8, 9, 9, 10, 11, 13, 14, 14] [0, 0, 0, 2, 2, 3, 6, 8, 9, 9, 10, 11, 13, 14, 14] [0, 0, 0, 2, 2, 3, 6, 8, 9, 9, 10, 11, 13, 14, 14] [0, 0, 0, 2, 2, 3, 6, 8, 9, 9, 10, 11, 13, 14, 14] [0, 0, 0, 2, 2, 3, 6, 8, 9, 9, 10, 11, 13, 14, 14] [0, 0, 0, 2, 2, 3, 6, 8, 9, 9, 10, 11, 13, 14, 14] [0, 0, 0, 2, 2, 3, 6, 8, 9, 9, 10, 11, 13, 14, 14] [0, 0, 0, 2, 2, 3, 6, 8, 9, 9, 10, 11, 13, 14, 14] [0, 0, 0, 2, 2, 3, 6, 8, 9, 9, 10, 11, 13, 14, 14] [0, 0, 0, 2, 2, 3, 6, 8, 9, 9, 10, 11, 13, 14, 14] [0, 0, 0, 2, 2, 3, 6, 8, 9, 9, 10, 11, 13, 14, 14] [0, 0, 0, 2, 2, 3, 6, 8, 9, 9, 10, 11, 13, 14, 14] Result: [0, 0, 0, 2, 2, 3, 6, 8, 9, 9, 10, 11, 13, 14, 14] Test: 5 Input: [0, 4, 5, 6, 6, 7, 7, 7, 8, 9, 10, 10, 12, 13, 13] [0, 4, 5, 6, 6, 7, 7, 7, 8, 9, 10, 10, 12, 13, 13] [0, 4, 5, 6, 6, 7, 7, 7, 8, 9, 10, 10, 12, 13, 13] [0, 4, 5, 6, 6, 7, 7, 7, 8, 9, 10, 10, 12, 13, 13] [0, 4, 5, 6, 6, 7, 7, 7, 8, 9, 10, 10, 12, 13, 13] [0, 4, 5, 6, 6, 7, 7, 7, 8, 9, 10, 10, 12, 13, 13] [0, 4, 5, 6, 6, 7, 7, 7, 8, 9, 10, 10, 12, 13, 13] [0, 4, 5, 6, 6, 7, 7, 7, 8, 9, 10, 10, 12, 13, 13] [0, 4, 5, 6, 6, 7, 7, 7, 8, 9, 10, 10, 12, 13, 13] [0, 4, 5, 6, 6, 7, 7, 7, 8, 9, 10, 10, 12, 13, 13] [0, 4, 5, 6, 6, 7, 7, 7, 8, 9, 10, 10, 12, 13, 13] [0, 4, 5, 6, 6, 7, 7, 7, 8, 9, 10, 10, 12, 13, 13] [0, 4, 5, 6, 6, 7, 7, 7, 8, 9, 10, 10, 12, 13, 13] [0, 4, 5, 6, 6, 7, 7, 7, 8, 9, 10, 10, 12, 13, 13] [0, 4, 5, 6, 6, 7, 7, 7, 8, 9, 10, 10, 12, 13, 13] Result: [0, 4, 5, 6, 6, 7, 7, 7, 8, 9, 10, 10, 12, 13, 13]

Insertion Sort Worst Case

Array Length: 5	Array Length: 10	Array Length: 15
Test: 1	Test: 1	Test: 1
Input: [0, 4, 0, 2, 3]	Input: [8, 1, 1, 0, 0, 9, 9, 6, 5, 0]	Input: [4, 4, 2, 1, 3, 13, 13, 8, 13, 12, 4, 7, 5, 7, 13]
[0, 4, 0, 2, 3]	[8, 8, 1, 0, 0, 9, 9, 6, 5, 0]	[4, 4, 2, 1, 3, 13, 13, 8, 13, 12, 4, 7, 5, 7, 13]
[0, 4, 4, 2, 3]	[1, 8, 8, 0, 0, 9, 9, 6, 5, 0]	[4, 4, 4, 1, 3, 13, 13, 8, 13, 12, 4, 7, 5, 7, 13]
[0, 0, 4, 4, 3]	[1, 1, 1, 8, 0, 9, 9, 6, 5, 0]	[2, 2, 4, 4, 3, 13, 13, 8, 13, 12, 4, 7, 5, 7, 13]
[0, 0, 2, 4, 4]	[0, 1, 1, 1, 8, 9, 9, 6, 5, 0]	[1, 2, 4, 4, 4, 13, 13, 8, 13, 12, 4, 7, 5, 7, 13]
Result: [0, 0, 2, 3, 4]	[0, 0, 1, 1, 8, 9, 9, 6, 5, 0]	[1, 2, 3, 4, 4, 13, 13, 8, 13, 12, 4, 7, 5, 7, 13]
Test: 2	[0, 0, 1, 1, 8, 9, 9, 6, 5, 0]	[1, 2, 3, 4, 4, 13, 13, 8, 13, 12, 4, 7, 5, 7, 13]
Input: [2, 1, 3, 3, 4]	[0, 0, 1, 1, 8, 8, 9, 9, 5, 0]	[1, 2, 3, 4, 4, 13, 13, 13, 13, 12, 4, 7, 5, 7, 13]
[2, 2, 3, 3, 4]	[0, 0, 1, 1, 6, 6, 8, 9, 9, 0]	[1, 2, 3, 4, 4, 8, 13, 13, 13, 12, 4, 7, 5, 7, 13]
[1, 2, 3, 3, 4]	[0, 0, 1, 1, 1, 5, 6, 8, 9, 9]	[1, 2, 3, 4, 4, 8, 13, 13, 13, 13, 4, 7, 5, 7, 13]
[1, 2, 3, 3, 4]	Result: [0, 0, 0, 1, 1, 5, 6, 8, 9, 9]	[1, 2, 3, 4, 4, 8, 8, 12, 13, 13, 13, 7, 5, 7, 13]
[1, 2, 3, 3, 4]	Test: 2	[1, 2, 3, 4, 4, 4, 8, 8, 12, 13, 13, 13, 5, 7, 13]
Result: [1, 2, 3, 3, 4]	Input: [3, 0, 7, 5, 9, 4, 2, 5, 7, 9]	[1, 2, 3, 4, 4, 4, 7, 7, 8, 12, 13, 13, 13, 7, 13]
Test: 3	[3, 3, 7, 5, 9, 4, 2, 5, 7, 9]	[1, 2, 3, 4, 4, 4, 5, 7, 8, 8, 12, 13, 13, 13, 13]
Input: [0, 4, 3, 0, 0]	[0, 3, 7, 5, 9, 4, 2, 5, 7, 9]	[1, 2, 3, 4, 4, 4, 5, 7, 7, 8, 12, 13, 13, 13, 13]
[0, 4, 3, 0, 0]	[0, 3, 7, 7, 9, 4, 2, 5, 7, 9]	Result: [1, 2, 3, 4, 4, 4, 5, 7, 7, 8, 12, 13, 13, 13, 13]
[0, 4, 4, 0, 0]	[0, 3, 5, 7, 9, 4, 2, 5, 7, 9]	Test: 2
[0, 3, 3, 4, 0]	[0, 3, 5, 5, 7, 9, 2, 5, 7, 9]	Input: [13, 0, 8, 13, 8, 4, 14, 8, 2, 14, 4, 9, 6, 7, 6]
[0, 0, 3, 3, 4]	[0, 3, 3, 4, 5, 7, 9, 5, 7, 9]	[13, 13, 8, 13, 8, 4, 14, 8, 2, 14, 4, 9, 6, 7, 6]
Result: [0, 0, 0, 3, 4]	[0, 2, 3, 4, 5, 7, 7, 9, 7, 9]	[0, 13, 13, 13, 8, 4, 14, 8, 2, 14, 4, 9, 6, 7, 6]
Test: 4	[0, 2, 3, 4, 5, 5, 7, 9, 9, 9]	[0, 8, 13, 13, 8, 4, 14, 8, 2, 14, 4, 9, 6, 7, 6]
Input: [4, 3, 1, 3, 0]	[0, 2, 3, 4, 5, 5, 7, 7, 9, 9]	[0, 8, 13, 13, 13, 4, 14, 8, 2, 14, 4, 9, 6, 7, 6]
[4, 4, 1, 3, 0]	Result: [0, 2, 3, 4, 5, 5, 7, 7, 9, 9]	[0, 8, 8, 8, 13, 13, 14, 8, 2, 14, 4, 9, 6, 7, 6]
[3, 3, 4, 3, 0]	Test: 3	[0, 4, 8, 8, 13, 13, 14, 8, 2, 14, 4, 9, 6, 7, 6]
[1, 3, 4, 4, 0]	Input: [3, 8, 5, 8, 7, 6, 8, 2, 1, 4]	[0, 4, 8, 8, 13, 13, 13, 14, 2, 14, 4, 9, 6, 7, 6]
[1, 1, 3, 3, 4]	[3, 8, 5, 8, 7, 6, 8, 2, 1, 4]	[0, 4, 4, 8, 8, 8, 13, 13, 14, 14, 4, 9, 6, 7, 6]
Result: [0, 1, 3, 3, 4]	[3, 8, 8, 8, 7, 6, 8, 2, 1, 4]	[0, 2, 4, 8, 8, 8, 13, 13, 14, 14, 4, 9, 6, 7, 6]
Test: 5	[3, 5, 8, 8, 7, 6, 8, 2, 1, 4]	[0, 2, 4, 8, 8, 8, 8, 13, 13, 14, 14, 9, 6, 7, 6]
Input: [4, 3, 1, 0, 3]	[3, 5, 8, 8, 8, 6, 8, 2, 1, 4]	[0, 2, 4, 4, 8, 8, 8, 13, 13, 13, 14, 14, 6, 7, 6]
[4, 4, 1, 0, 3]	[3, 5, 7, 7, 8, 8, 8, 2, 1, 4]	[0, 2, 4, 4, 8, 8, 8, 8, 9, 13, 13, 14, 14, 7, 6]
[3, 3, 4, 0, 3]	[3, 5, 6, 7, 8, 8, 8, 2, 1, 4]	[0, 2, 4, 4, 6, 8, 8, 8, 8, 9, 13, 13, 14, 14, 6]
[1, 1, 3, 4, 3]	[3, 3, 5, 6, 7, 8, 8, 8, 1, 4]	[0, 2, 4, 4, 6, 7, 7, 8, 8, 8, 9, 13, 13, 14, 14]
[0, 1, 3, 4, 4]	[2, 2, 3, 5, 6, 7, 8, 8, 8, 4]	Result: [0, 2, 4, 4, 6, 6, 7, 8, 8, 8, 9, 13, 13, 14, 14]
Result: [0, 1, 3, 3, 4]	[1, 2, 3, 5, 5, 6, 7, 8, 8, 8]	Test: 3
	Result: [1, 2, 3, 4, 5, 6, 7, 8, 8, 8]	Input: [2, 8, 9, 14, 12, 9, 13, 12, 3, 14, 10, 4, 3, 11, 12]
	Test: 4	[2, 8, 9, 14, 12, 9, 13, 12, 3, 14, 10, 4, 3, 11, 12]
	Input: [5, 0, 8, 6, 4, 5, 3, 6, 0, 2]	[2, 8, 9, 14, 12, 9, 13, 12, 3, 14, 10, 4, 3, 11, 12]
	[5, 5, 8, 6, 4, 5, 3, 6, 0, 2]	[2, 8, 9, 14, 12, 9, 13, 12, 3, 14, 10, 4, 3, 11, 12]
	[0, 5, 8, 6, 4, 5, 3, 6, 0, 2]	[2, 8, 9, 14, 14, 9, 13, 12, 3, 14, 10, 4, 3, 11, 12]
	[0, 5, 8, 8, 4, 5, 3, 6, 0, 2]	[2, 8, 9, 12, 12, 14, 13, 12, 3, 14, 10, 4, 3, 11, 12]
	[0, 5, 5, 6, 8, 5, 3, 6, 0, 2]	[2, 8, 9, 9, 12, 14, 14, 12, 3, 14, 10, 4, 3, 11, 12]
	[0, 4, 5, 6, 6, 8, 3, 6, 0, 2]	[2, 8, 9, 9, 12, 13, 13, 14, 3, 14, 10, 4, 3, 11, 12]

	[0, 4, 4, 5, 5, 6, 8, 6, 0, 2] [0, 3, 4, 5, 5, 6, 8, 8, 0, 2] [0, 3, 3, 4, 5, 5, 6, 6, 8, 2] [0, 0, 3, 3, 4, 5, 5, 6, 6, 8] Result: [0, 0, 2, 3, 4, 5, 5, 6, 6, 8] Test: 5 Input: [4, 7, 3, 6, 7, 5, 7, 2, 2, 8] [4, 7, 3, 6, 7, 5, 7, 2, 2, 8] [4, 4, 7, 6, 7, 5, 7, 2, 2, 8] [3, 4, 7, 7, 7, 5, 7, 2, 2, 8] [3, 4, 6, 7, 7, 5, 7, 2, 2, 8] [3, 4, 6, 6, 7, 7, 7, 2, 2, 8] [3, 4, 5, 6, 7, 7, 7, 2, 2, 8] [3, 3, 4, 5, 6, 7, 7, 7, 2, 8] [2, 3, 3, 4, 5, 6, 7, 7, 7, 8] [2, 2, 3, 4, 5, 6, 7, 7, 7, 8] Result: [2, 2, 3, 4, 5, 6, 7, 7, 7, 8]	[2, 8, 8, 9, 9, 12, 12, 13, 14, 14, 10, 4, 3, 11, 12] [2, 3, 8, 9, 9, 12, 12, 13, 14, 14, 10, 4, 3, 11, 12] [2, 3, 8, 9, 9, 12, 12, 12, 13, 14, 14, 4, 3, 11, 12] [2, 3, 8, 8, 9, 9, 10, 12, 12, 13, 14, 14, 3, 11, 12] [2, 3, 4, 4, 8, 9, 9, 10, 12, 12, 13, 14, 14, 11, 12] [2, 3, 3, 4, 8, 9, 9, 10, 12, 12, 12, 13, 14, 14, 12] [2, 3, 3, 4, 8, 9, 9, 10, 11, 12, 12, 13, 13, 14, 14] Result: [2, 3, 3, 4, 8, 9, 9, 10, 11, 12, 12, 12, 13, 14, 14] Test: 4 Input: [8, 6, 3, 14, 2, 12, 7, 5, 12, 1, 9, 7, 8, 1, 11] [8, 8, 3, 14, 2, 12, 7, 5, 12, 1, 9, 7, 8, 1, 11] [6, 6, 8, 14, 2, 12, 7, 5, 12, 1, 9, 7, 8, 1, 11] [3, 6, 8, 14, 2, 12, 7, 5, 12, 1, 9, 7, 8, 1, 11] [3, 3, 6, 8, 14, 12, 7, 5, 12, 1, 9, 7, 8, 1, 11] [2, 3, 6, 8, 14, 14, 7, 5, 12, 1, 9, 7, 8, 1, 11] [2, 3, 6, 8, 8, 12, 14, 5, 12, 1, 9, 7, 8, 1, 11] [2, 3, 6, 6, 7, 8, 12, 14, 12, 1, 9, 7, 8, 1, 11] [2, 3, 5, 6, 7, 8, 12, 14, 14, 1, 9, 7, 8, 1, 11] [2, 2, 3, 5, 6, 7, 8, 12, 12, 14, 9, 7, 8, 1, 11] [1, 2, 3, 5, 6, 7, 8, 12, 12, 12, 14, 7, 8, 1, 11] [1, 2, 3, 5, 6, 7, 8, 8, 9, 12, 12, 14, 8, 1, 11] [1, 2, 3, 5, 6, 7, 7, 8, 9, 9, 12, 12, 14, 1, 11] [1, 2, 2, 3, 5, 6, 7, 7, 8, 8, 9, 12, 12, 14, 11] [1, 1, 2, 3, 5, 6, 7, 7, 8, 8, 9, 12, 12, 12, 14] Result: [1, 1, 2, 3, 5, 6, 7, 7, 8, 8, 9, 11, 12, 12, 14] Test: 5 Input: [6, 11, 6, 10, 12, 7, 6, 5, 14, 1, 9, 7, 13, 14, 14] [6, 11, 6, 10, 12, 7, 6, 5, 14, 1, 9, 7, 13, 14, 14] [6, 11, 11, 10, 12, 7, 6, 5, 14, 1, 9, 7, 13, 14, 14] [6, 6, 11, 11, 12, 7, 6, 5, 14, 1, 9, 7, 13, 14, 14] [6, 6, 10, 11, 12, 7, 6, 5, 14, 1, 9, 7, 13, 14, 14] [6, 6, 10, 10, 11, 12, 6, 5, 14, 1, 9, 7, 13, 14, 14] [6, 6, 7, 7, 10, 11, 12, 5, 14, 1, 9, 7, 13, 14, 14] [6, 6, 6, 6, 7, 10, 11, 12, 14, 1, 9, 7, 13, 14, 14] [5, 6, 6, 6, 7, 10, 11, 12, 14, 1, 9, 7, 13, 14, 14] [5, 5, 6, 6, 6, 7, 10, 11, 12, 14, 9, 7, 13, 14, 14] [1, 5, 6, 6, 6, 7, 10, 10, 11, 12, 14, 7, 13, 14, 14] [1, 5, 6, 6, 6, 7, 9, 9, 10, 11, 12, 14, 13, 14, 14] [1, 5, 6, 6, 6, 7, 7, 9, 10, 11, 12, 14, 14, 14, 14] [1, 5, 6, 6, 6, 7, 7, 9, 10, 11, 12, 13, 14, 14, 14] [1, 5, 6, 6, 6, 7, 7, 9, 10, 11, 12, 13, 14, 14, 14] Result: [1, 5, 6, 6, 6, 7, 7, 9, 10, 11, 12, 13, 14, 14, 14]

Counting Sort Worst Performance

Array Length: 3 Test: 1 Input: [0, 2, 1] Counting Array: [0, 0, 0, 0, 0, 0, 0, 0, 0] Loop[0, 2, 1] Loop[1, 2, 3, 3, 3, 3, 3, 3, 3, 0] Loop[1, 2, 3, 3, 3, 3, 3, 3, 3, 0] Loop[1, 2, 3, 3, 3, 3, 3, 3, 3, 0] Loop[0, 0, 0] Output[0, 0, 0] Test: 2 Input: [0, 1, 2] Counting Array: [0, 0, 0, 0, 0, 0, 0, 0, 0] Loop[0, 1, 2] Loop[1, 2, 3, 3, 3, 3, 3, 3, 3, 0] Loop[1, 2, 3, 3, 3, 3, 3, 3, 3, 0] Loop[1, 2, 3, 3, 3, 3, 3, 3, 3, 0] Loop[0, 0, 0] Output[0, 0, 0] Test: 3 Input: [1, 1, 2] Counting Array: [0, 0, 0, 0, 0, 0, 0, 0, 0] Loop[1, 1, 2] Loop[0, 2, 3, 3, 3, 3, 3, 3, 3, 0] Loop[0, 2, 3, 3, 3, 3, 3, 3, 3, 0] Loop[0, 2, 3, 3, 3, 3, 3, 3, 3, 0]	Array Length: 6 Test: 1 Input: [1, 3, 4, 4, 1, 3] Counting Array: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] Loop[1, 3, 4, 4, 1, 3] Loop[0, 2, 2, 4, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 0] Loop[0, 2, 2, 4, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 0] Loop[0, 2, 2, 4, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 0] Loop[0, 0, 0, 0, 0, 0] Output[0, 0, 0, 0, 0, 0] Test: 2 Input: [0, 5, 0, 1, 4, 5] Counting Array: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] Loop[0, 5, 0, 1, 4, 5] Loop[2, 3, 3, 3, 4, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 0] Loop[2, 3, 3, 3, 4, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 0] Loop[2, 3, 3, 3, 4, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 0] Loop[0, 0, 0, 0, 0, 0] Output[0, 0, 0, 0, 0, 0] Test: 3 Input: [1, 0, 4, 3, 5, 1]	Array Length: 9 Test: 1 Input: [1, 8, 4, 1, 7, 1, 2, 6, 4] Counting Array: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] Loop[1, 8, 4, 1, 7, 1, 2, 6, 4] Loop[0, 3, 4, 4, 6, 6, 7, 8, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 0] Loop[0, 3, 4, 4, 6, 6, 7, 8, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 0] Loop[0, 3, 4, 4, 6, 6, 7, 8, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 0] Loop[0, 0, 0, 0, 0, 0, 0, 0, 0] Output[0, 0, 0, 0, 0, 0, 0, 0, 0] Test: 2 Input: [5, 3, 0, 3, 6, 2, 4, 8, 4] Counting Array: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] Loop[5, 3, 0, 3, 6, 2, 4, 8, 4] Loop[1, 1, 2, 4, 6, 7, 8, 8, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 0] Loop[1, 1, 2, 4, 6, 7, 8, 8, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 0] Loop[1, 1, 2, 4, 6, 7, 8, 8, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9,

Loop[0, 0, 0]
Output[0, 0, 0]

Test: 4
Input: [2, 1, 1]
Counting Array: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

Loop[2, 1, 1]
Loop[0, 2, 3, 3, 3, 3, 3, 3, 3, 0]
Loop[0, 2, 3, 3, 3, 3, 3, 3, 3, 0]
Loop[0, 2, 3, 3, 3, 3, 3, 3, 3, 0]
Loop[0, 0, 0]
Output[0, 0, 0]

Test: 5
Input: [2, 2, 0]
Counting Array: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

Loop[2, 2, 0]
Loop[1, 1, 3, 3, 3, 3, 3, 3, 3, 0]
Loop[1, 1, 3, 3, 3, 3, 3, 3, 3, 0]
Loop[1, 1, 3, 3, 3, 3, 3, 3, 3, 0]
Loop[0, 0, 0]
Output[0, 0, 0]

Counting Array: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

Loop[1, 0, 4, 3, 5, 1]
Loop[1, 3, 3, 4, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 0]
Loop[1, 3, 3, 4, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 0]
Loop[1, 3, 3, 4, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 0]
Loop[0, 0, 0, 0, 0, 0]
Output[0, 0, 0, 0, 0, 0]

Test: 4
Input: [3, 3, 5, 2, 4, 2]
Counting Array: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

Loop[3, 3, 5, 2, 4, 2]
Loop[0, 0, 2, 4, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 0]
Loop[0, 0, 2, 4, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 0]
Loop[0, 0, 2, 4, 5, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 0]
Loop[0, 0, 0, 0, 0, 0]
Output[0, 0, 0, 0, 0, 0]

Test: 5
Input: [2, 5, 4, 2, 0, 5]
Counting Array: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

Loop[2, 5, 4, 2, 0, 5]

9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 0]
Loop[0, 0, 0, 0, 0, 0, 0, 0, 0]
Output[0, 0, 0, 0, 0, 0, 0, 0, 0]

Test: 3
Input: [8, 8, 2, 7, 8, 5, 4, 3, 8]
Counting Array: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

Loop[8, 8, 2, 7, 8, 5, 4, 3, 8]
Loop[0, 0, 1, 2, 3, 4, 4, 5, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 0]
Loop[0, 0, 1, 2, 3, 4, 4, 5, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 0]
Loop[0, 0, 1, 2, 3, 4, 4, 5, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 0]
Loop[0, 0, 0, 0, 0, 0, 0, 0, 0]
Output[0, 0, 0, 0, 0, 0, 0, 0, 0]

Test: 4
Input: [3, 4, 2, 3, 5, 1, 5, 7, 2]
Counting Array: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

Loop[3, 4, 2, 3, 5, 1, 5, 7, 2]
Loop[0, 1, 3, 5, 6, 8, 8, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 0]
Loop[0, 1, 3, 5, 6, 8, 8, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 0]
Loop[0, 1, 3, 5, 6, 8, 8, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 0]

	Loop[1, 1, 3, 3, 4, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 0] Loop[1, 1, 3, 3, 4, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 0] Loop[1, 1, 3, 3, 4, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 0] Loop[0, 0, 0, 0, 0, 0] Output[0, 0, 0, 0, 0, 0]	Loop[0, 0, 0, 0, 0, 0, 0, 0, 0] Output[0, 0, 0, 0, 0, 0, 0, 0, 0] Test: 5 Input: [4, 3, 0, 6, 7, 7, 3, 1, 3] Counting Array: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] Loop[4, 3, 0, 6, 7, 7, 3, 1, 3] Loop[1, 2, 2, 5, 6, 6, 7, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 0] Loop[1, 2, 2, 5, 6, 6, 7, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 0] Loop[1, 2, 2, 5, 6, 6, 7, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 0] Loop[0, 0, 0, 0, 0, 0, 0, 0, 0] Output[0, 0, 0, 0, 0, 0, 0, 0, 0]

Counting Sort Best Performance

Array Length: 5 Test: 1 Input: [2, 4, 0, 4, 3] Counting Array: [0, 0, 0, 0, 0] Loop[2, 4, 0, 4, 3] Loop[1, 1, 2, 3, 2] Loop[1, 1, 2, 3, 2] Loop[1, 1, 2, 3, 2] Loop[0, 0, 0, 0, 0] Output[0, 0, 0, 0, 0] Test: 2 Input: [4, 3, 2, 1, 0] Counting Array: [0, 0, 0, 0, 0]	Array Length: 10 Test: 1 Input: [8, 9, 5, 1, 3, 2, 8, 5, 4, 8] Counting Array: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0] Loop[8, 9, 5, 1, 3, 2, 8, 5, 4, 8] Loop[0, 1, 2, 3, 4, 6, 6, 6, 9, 1] Loop[0, 1, 2, 3, 4, 6, 6, 6, 9, 1] Loop[0, 1, 2, 3, 4, 6, 6, 6, 9, 1] Loop[0, 0, 0, 0, 0, 0, 0, 0, 0, 0] Output[0, 0, 0, 0, 0, 0, 0, 0, 0, 0] Test: 2 Input: [1, 5, 3, 2, 3, 9, 7, 0, 5, 6] Counting Array: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	Array Length: 15 Test: 1 Input: [11, 1, 3, 1, 11, 2, 11, 12, 3, 11, 12, 9, 2, 7, 8] Counting Array: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] Loop[11, 1, 3, 1, 11, 2, 11, 12, 3, 11, 12, 9, 2, 7, 8] Loop[0, 2, 4, 6, 6, 6, 6, 7, 8, 9, 9, 13, 2] Loop[0, 2, 4, 6, 6, 6, 6, 7, 8, 9, 9, 13, 2] Loop[0, 2, 4, 6, 6, 6, 6, 7, 8, 9, 9, 13, 2] Loop[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] Output[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] Test: 2 Input: [2, 11, 13, 0, 6, 7, 6, 13, 3, 11, 13, 11, 3, 4, 2] Counting Array: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] Loop[2, 11, 13, 0, 6, 7, 6, 13, 3, 11, 13, 11, 3, 4, 2]

		Loop[1, 1, 3, 5, 6, 6, 8, 9, 9, 9, 9, 12, 12, 3]
Loop[4, 3, 2, 1, 0]	Loop[1, 5, 3, 2, 3, 9, 7, 0, 5, 6]	Loop[1, 1, 3, 5, 6, 6, 8, 9, 9, 9, 9, 12, 12, 3]
Loop[1, 2, 3, 4, 1]	Loop[1, 2, 3, 5, 5, 7, 8, 9, 9, 1]	Loop[1, 1, 3, 5, 6, 6, 8, 9, 9, 9, 9, 12, 12, 3]
Loop[1, 2, 3, 4, 1]	Loop[1, 2, 3, 5, 5, 7, 8, 9, 9, 1]	Loop[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Loop[1, 2, 3, 4, 1]	Loop[1, 2, 3, 5, 5, 7, 8, 9, 9, 1]	Output[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Loop[0, 0, 0, 0, 0]	Loop[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	
Output[0, 0, 0, 0, 0]	Output[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	Test: 3
		Input: [4, 2, 5, 10, 5, 1, 10, 2, 5, 13, 12, 7, 11, 7, 6]
Test: 3	Test: 3	Counting Array: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Input: [3, 3, 2, 3, 0]	Input: [8, 9, 5, 1, 5, 8, 4, 9, 2, 5]	
Counting Array: [0, 0, 0, 0]	Counting Array: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	Loop[4, 2, 5, 10, 5, 1, 10, 2, 5, 13, 12, 7, 11, 7, 6]
		Loop[0, 1, 3, 3, 4, 7, 8, 10, 10, 10, 12, 13, 14, 1]
		Loop[0, 1, 3, 3, 4, 7, 8, 10, 10, 10, 12, 13, 14, 1]
Loop[3, 3, 2, 3, 0]	Loop[8, 9, 5, 1, 5, 8, 4, 9, 2, 5]	Loop[0, 1, 3, 3, 4, 7, 8, 10, 10, 10, 12, 13, 14, 1]
Loop[1, 1, 2, 3]	Loop[0, 1, 2, 2, 3, 6, 6, 6, 8, 2]	Loop[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Loop[1, 1, 2, 3]	Loop[0, 1, 2, 2, 3, 6, 6, 6, 8, 2]	Output[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Loop[1, 1, 2, 3]	Loop[0, 1, 2, 2, 3, 6, 6, 6, 8, 2]	
Loop[0, 0, 0, 0, 0]	Loop[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	Test: 4
Output[0, 0, 0, 0, 0]	Output[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	Input: [4, 9, 2, 4, 13, 0, 12, 9, 12, 2, 11, 10, 5, 7, 0]
		Counting Array: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Test: 4	Test: 4	
Input: [2, 4, 2, 4, 3]	Input: [7, 7, 6, 6, 8, 7, 4, 2, 5, 0]	Loop[4, 9, 2, 4, 13, 0, 12, 9, 12, 2, 11, 10, 5, 7, 0]
Counting Array: [0, 0, 0, 0, 0]	Counting Array: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	Loop[2, 2, 4, 4, 6, 7, 7, 8, 8, 10, 11, 12, 14, 1]
		Loop[2, 2, 4, 4, 6, 7, 7, 8, 8, 10, 11, 12, 14, 1]
		Loop[2, 2, 4, 4, 6, 7, 7, 8, 8, 10, 11, 12, 14, 1]
Loop[2, 4, 2, 4, 3]	Loop[7, 7, 6, 6, 8, 7, 4, 2, 5, 0]	Loop[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Loop[0, 0, 2, 3, 2]	Loop[1, 1, 2, 2, 3, 4, 6, 9, 1]	Output[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Loop[0, 0, 2, 3, 2]	Loop[1, 1, 2, 2, 3, 4, 6, 9, 1]	
Loop[0, 0, 2, 3, 2]	Loop[1, 1, 2, 2, 3, 4, 6, 9, 1]	Test: 5
Loop[0, 0, 0, 0, 0]	Loop[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	Input: [11, 10, 2, 4, 12, 0, 2, 13, 4, 8, 3, 7, 3, 12, 9]
Output[0, 0, 0, 0, 0]	Output[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	Counting Array: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Test: 5	Test: 5	Loop[11, 10, 2, 4, 12, 0, 2, 13, 4, 8, 3, 7, 3, 12, 9]
Input: [3, 3, 0, 4, 2]	Input: [0, 6, 4, 0, 3, 4, 5, 9, 1, 0]	Loop[1, 1, 3, 5, 7, 7, 7, 8, 9, 10, 11, 12, 14, 1]
Counting Array: [0, 0, 0, 0, 0]	Counting Array: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	Loop[1, 1, 3, 5, 7, 7, 7, 8, 9, 10, 11, 12, 14, 1]
		Loop[1, 1, 3, 5, 7, 7, 7, 8, 9, 10, 11, 12, 14, 1]
		Loop[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Loop[3, 3, 0, 4, 2]	Loop[0, 6, 4, 0, 3, 4, 5, 9, 1, 0]	Output[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Loop[1, 1, 2, 4, 1]	Loop[3, 4, 4, 5, 7, 8, 9, 9, 9, 1]	
Loop[1, 1, 2, 4, 1]	Loop[3, 4, 4, 5, 7, 8, 9, 9, 9, 1]	
Loop[1, 1, 2, 4, 1]	Loop[3, 4, 4, 5, 7, 8, 9, 9, 9, 1]	
Loop[0, 0, 0, 0, 0]	Loop[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	
Output[0, 0, 0, 0, 0]	Output[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]	

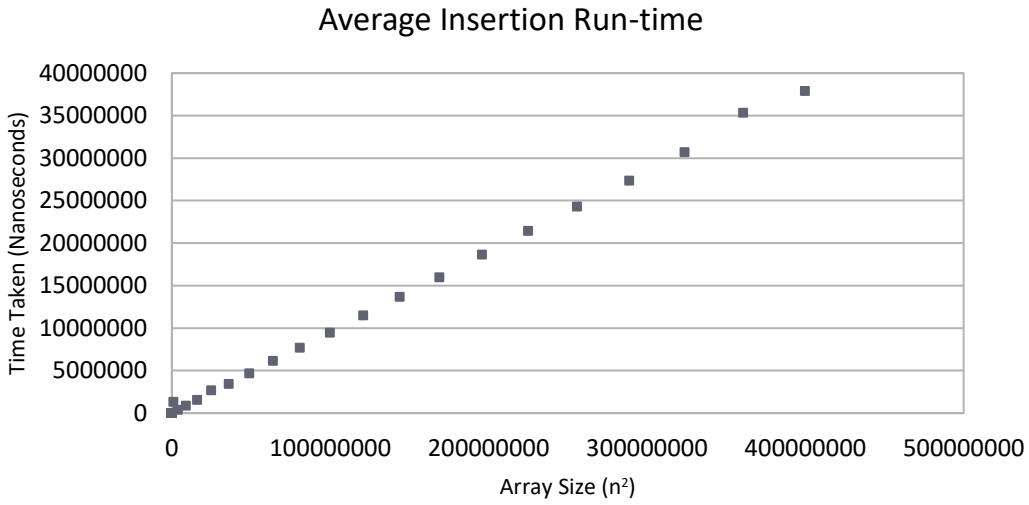
Experimental Setup and Results

Insertion Sort

Average Case

Insertion Sort Average Run-time

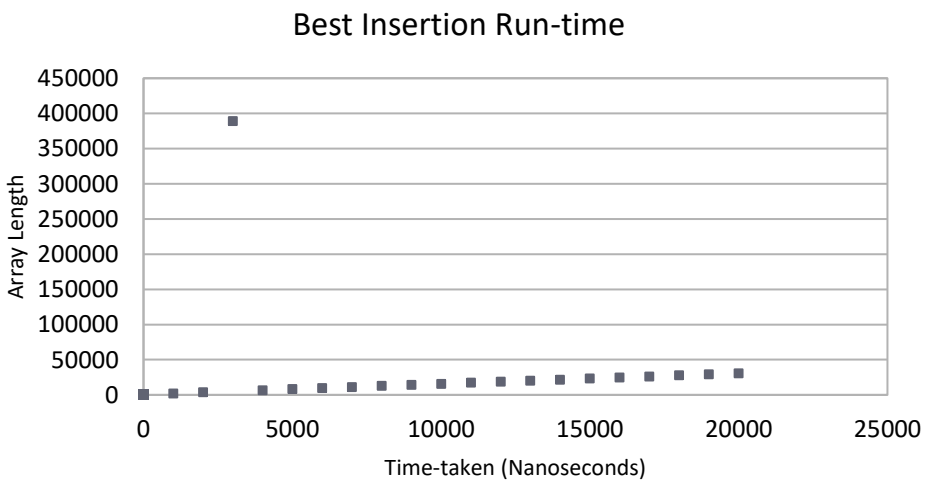
Array Size (n^2)	Test 1	Test 2	Test 3	Test 4	Test 5	Average
1000000	2170452	662585	732721	655522	2423505	1328957
4000000	390061	396841	386786	396478	396993	393431.8
9000000	872222	850104	866488	862650	867174	863727.6
1600000	1549978	1541562	1525472	1522555	1554235	1538760.4
2500000	2393083	2381080	2915883	2913193	2880501	2696748
3600000	3528593	3399449	3450134	3378017	3425165	3436271.6
4900000	4755479	4665585	4594047	4731031	4589002	4667028.8
6400000	6245322	6181503	6124583	6115536	6099164	6153221.6
81000000	7751664	7662837	7683907	7642542	7633305	7674851
100000000	9575966	9415590	9428357	9474617	9443061	9467518.2
121000000	11503919	11409829	11427196	11577672	11515341	11486791.4
144000000	13759729	13808753	13577091	13656810	13519069	13664290.4
169000000	15883986	15974555	16095562	15885466	16067123	15981338.4
196000000	18574626	18559142	18857633	18741408	18423099	18631181.6
225000000	21470529	21419775	21277723	21323638	21635047	21425342.4
256000000	24400548	24217225	24227696	24219190	24425082	24297948.2
289000000	27218171	27616497	27317100	27336305	27254445	27348503.6
324000000	30800409	30525608	30611896	30852539	30758206	30709731.6
361000000	33971239	33637549	34009702	36070537	38943648	35326535
400000000	37521407	38629380	37666988	37665410	37922556	37881148.2



Best Case

Insertion Sort Best Run-time

Array Size (n)	Test 1	Test 2	Test 3	Test 4	Test 5	Average
1000	1846	1796	1696	1709	1676	1744.6
2000	3219	3427	3259	3214	3191	3262
3000	4712	4675	4683	4701	1925367	388827.6
4000	6334	6371	6279	6318	6230	6306.4
5000	7756	7764	7728	7767	7751	7753.2
6000	9188	9200	9207	9205	9215	9203
7000	10788	10754	10755	10734	10802	10766.6
8000	12231	12279	12250	12233	12230	12244.6
9000	13915	13765	13790	13759	13811	13808
10000	15255	15305	15287	15313	15274	15286.8
11000	16829	16868	16833	16838	16862	16846
12000	18350	18382	18373	18346	18318	18353.8
13000	19882	19782	19834	19806	19816	19824
14000	21324	21312	21312	21302	21306	21311.2
15000	22792	22822	22827	22808	22806	22811
16000	24304	24322	24309	24323	24315	24314.6
17000	25726	25808	25796	25700	25754	25761.2
18000	27270	27239	27266	27233	27241	27249.8
19000	28732	28735	28752	28764	28770	28750.6
20000	30269	30259	30264	30287	30272	30270.2


Best Insertion Run-time
450000
400000
350000
300000
250000
200000
150000
100000
50000
0
Array Length
0
5000
10000
15000
20000
25000
Time-taken (Nanoseconds)

Worst Case

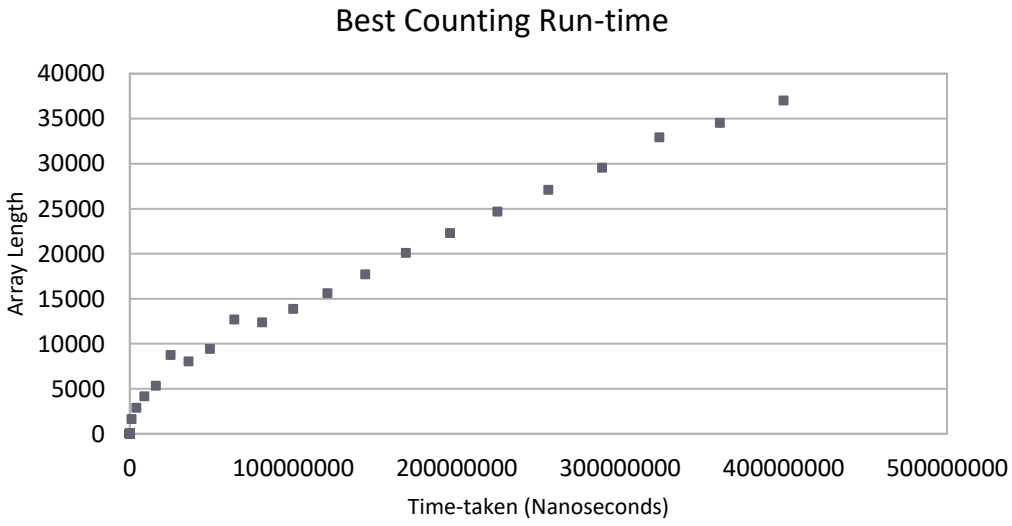
Insertion Sort Worst Run-time

Array Size (n^2)	Test 1	Test 2	Test 3	Test 4	Test 5	Average
1000000	103042	110372	100591	102732	101429	103633.2
4000000	391164	408051	389448	397660	407799	398824.4
9000000	863783	868957	927223	978488	891438	905983.8
1600000	1539756	1566535	1530237	1533881	1517769	1537635.6
2500000	2403214	2401342	2362789	2407735	2422579	2399531.8
3600000	3440064	3420607	3435264	3484405	3419565	3439981
4900000	4687267	4694046	4651487	4723205	4638907	4678982.4
6400000	611415	6062180	6036238	6178228	6056428	6088897.8
81000000	7699361	7662304	7732173	7767011	7698984	7711966.6
100000000	9465234	9421669	9459689	9513670	9648801	9501812.6
121000000	11553320	11496743	11493326	11413890	11475311	11486518
144000000	13399114	13627273	13795058	13771386		
169000000						
196000000						
225000000						
256000000						
289000000						
324000000						
361000000						
400000000						

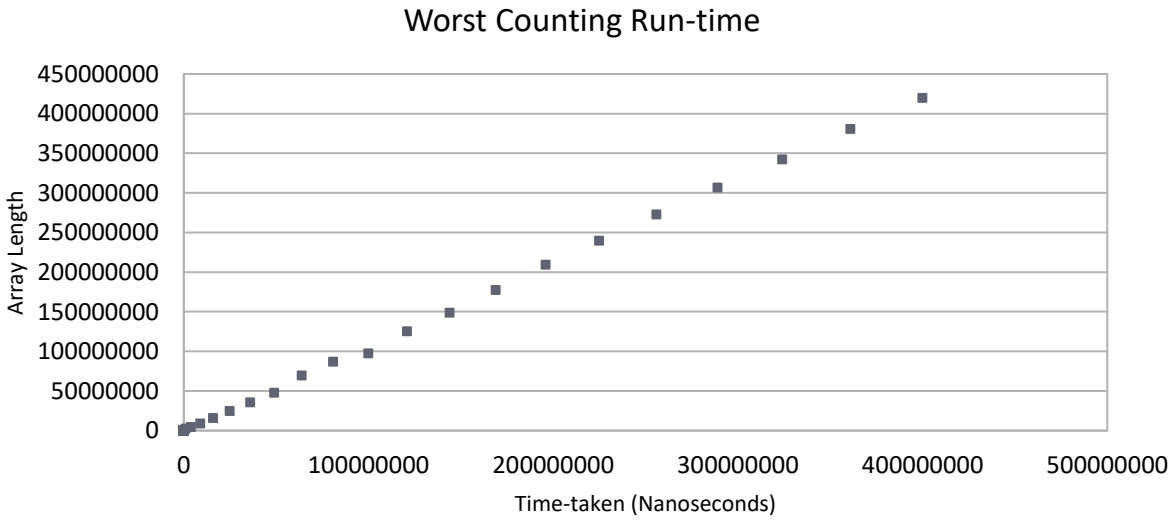

Worst Insertion Run-time
40000000
35000000
30000000
25000000
20000000
15000000
10000000
5000000
0
Array Length
0
100000000
200000000
300000000
400000000
500000000
Time-taken (Nanoseconds)

Counting Sort

Best Performance



Worst Performance



Conclusions and Discussion

From the experiments that have been conducted throughout. I have found they have been conducted satisfyingly.

The experiment has successfully shown the time complexity of the two different sorting algorithms with different cases. Where I conducted 20 data sets from input size 1000 to 20000 with random numbers, sorted and reversed sequence.

This mean that the experiment matched the expected and theoretical time complexity.

Case	Insertion Sort	Counting Sort
Best	O(n)	O(n + k)
Average	$O(n^2)$	O(n + k)
Worst	$O(n^2)$	O(n + k)

To finally conclude, there are no best sorting algorithms that would be able to point out the characteristics of the 2 tested algorithms in time complexity. Insertion sort is useful for large values of data and sorted input sequences compared to Counting sort. Where Counting sort could be defined as a stable sorting algorithm, but it is also efficient when comparing integer or an integer representable data, where maximum key value is small and handle unknown data order.

Source Code

```
import java.util.Arrays;

public class Sorting_alg {
	public static void main(String[] args){
		insertionTest();
		countingTest_worstcase();
  countingTest_bestcase();
	}

	public static int[] getRandom(int n) {	//pass the lenght of array to be created
	//create a new list of array with bunch of random 'n' integers
	int[] list = new int[n];
	for (int i = 0; i<list.length; i++){
			//fill the new array with random integers
	   list[i] = (int)(Math.random()*n);
	}
	   return list;
	}


 public static int[] sort(int[] a){
	//List of unsorted random integers
 Arrays.sort(a); //Sort array 'a'
	//Return a list of sroted arrays in ascending order
	return a;
 }

 public static int[] reversesort(int[] b){
			//Create a new array of reversed integers with a length of input arrays
   for(int z = 0; z < (b.length/2); z++){
      int temp = b[z];
						//reverse array
      b[z] = b[b.length -1 -z];
      b[b.length -1 - z] = temp;
   }
   return b;
 }
	////////////////////////////////////////////////////////////////////////////////
	/////////////////////////////////INSERTION SORT/////////////////////////////////
	////////////////////////////////////////////////////////////////////////////////

	//ALGORITHM FOR INSERTION SORT//
	public static void insertionSort (int a[], int n) {
		// System.out.println("Input: " +Arrays.toString(a));
		long insertionStart = System.nanoTime();
```

```
		for(int i = 1; i<n; i++) {
			int item = a[i];
			int j = i-1;
			while (j>=0 && a[j]>item) {
				a[j+1] = a[j];
				j = j-1;
			}
			// System.out.println(Arrays.toString(a));
			a[j+1] = item;
		}
		// System.out.println("Result: " +Arrays.toString(a));
		long insertionEnd =  System.nanoTime();
	}

	public static void insertionTest() {
			//create a loop of array length starting from 1000 to 20000, incrementing by
1000
    for (int value_avg = 1000; value_avg<=20000; value_avg+=1000) {
			//for (int value_avg = 5; value_avg<=15; value_avg+=5) {
      System.out.println("Array Length: " +value_avg);
                                    //create a loop of data points from 1-5
      for (int h = 1; h<=5; h++) {
        System.out.println("Test: " +h);
        int[] rand_arr = getRandom(value_avg); //declaring array in average case
        int n = rand_arr.length;
                                                    //using nanoTime to get the current time
        long insertionStart = System.nanoTime();
                                                    //Run the insertion method
        insertionSort(rand_arr, n);
        long insertionEnd =  System.nanoTime();
                                                    //Find the time taken and records the time
for the algorithm
        System.out.println("Average_Insertion_Run-time "+ (insertionEnd - insertionStart));
      }
    }

		//create a loop of array length starting from 1000 to 20000, incrementing by 1000
		for (int value_best = 1000; value_best<=20000; value_best+=1000) {
		//for (int value_best = 5; value_best<=15; value_best+=5) {
      System.out.println("Array Length: " +value_best);
                                    //create a loop of data points from 1-5
      for (int h = 1; h<=5; h++) {
        System.out.println("Test: " +h);
         int[] rand_arr = getRandom(value_best); //declaring array in best case
         int n = rand_arr.length;
                                                    //using nanoTime to get the current
time
         Arrays.sort(rand_arr);
```

```
            long insertionStart = System.nanoTime();
                                                                    //Run the insertion method
            insertionSort(rand_arr, n);
            long insertionEnd =  System.nanoTime();
                                                                    //Find the time taken and records
the time for the algorithm
            System.out.println("Best_Insertion_Run-time "+(insertionEnd - insertionStart));
         }
    }

    int[] randomReverseSorted_array;//initialising the array
                            //create a loop of array length starting from 1000 to 20000, incrementing by
1000
    for (int value_worst = 1000; value_worst<=20000; value_worst+=1000) {
                            //for (int value_worst = 5; value_worst<=15; value_worst+=5) {
       System.out.println("Array Length: " +value_worst);
                                        //create a loop of data points from 1-5
       for (int h = 1; h<=5; h++) {
          System.out.println("Test: " +h);
          int[] rand_arr = getRandom(value_worst); //declaring array in worst case
                                                                    //Reverse the unsorted array to make it
sorted in reverse descending order
          randomReverseSorted_array = reversesort(rand_arr);
          int n = rand_arr.length;
                                                                    //using nanoTime to get the current time
          long insertionStart = System.nanoTime();
          insertionSort(randomReverseSorted_array, n);
          long insertionEnd =  System.nanoTime();
                                                                    //Find the time taken and records the time
for the algorithm
          System.out.println("Worst_Insertion_Run-time "+(insertionEnd - insertionStart));
       }
    }
   }

////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////COUNTING SORT/////////////////////////////////
////////////////////////////////////////////////////////////////////////////////

          //ALGORITHM FOR COUNTING SORT//
          public static void countingSort (int a[], int b[], int n, int k) {
                    int[] c = new int[k+1];
                    // System.out.println("Input: " +Arrays.toString(a));
                    // System.out.println("Counting Array: " +Arrays.toString(c));
                    // System.out.println('\n'+"Loop" +Arrays.toString(a));
                    for(int j = 0; j<(n); j++) {
                              c[a[j]] = c[a[j]]+1;
                    }
```

```
		for(int i=1; i<(k); i++) {
			c[i] = c[i]+c[i-1];
		}
		// System.out.println("Loop" +Arrays.toString(c));
		for(int j = (n-1);j<=0; j--) {
			b[c[a[j]]-1] = a[j];
			c[a[j]] = c[a[j]]-1;
		}
		// System.out.println("Loop" +Arrays.toString(c));
		// System.out.println("Loop" +Arrays.toString(c));
		// System.out.println("Loop" +Arrays.toString(b));
		// System.out.println("Output" +Arrays.toString(b));
	}

  public static int getMax(int[] arr_input){
                                    //set first value in array as the max
    int max = arr_input[0];
    for(int i = 1;i<arr_input.length;i++){
                                                    //if the value in the array is bigger, set new value as
max
                                                    if(arr_input[i]>max){
        max = arr_input[i];
      }
    }
                                    //return the max integer
    return max;
  }

		//Generate data for Counting Worse Case//
  public static void countingTest_worstcase() {
    int[] rand_arr ;
                          //create a loop of array length starting from 1000 to 20000, incrementing
by 1000
    for (int value_worst_C = 1000; value_worst_C<=20000; value_worst_C+=1000) {
                          // for (int value_worst_C = 3; value_worst_C<=9; value_worst_C+=3) {
      System.out.println("Array Length: " +value_worst_C);
                                                    //create a loop of data points from 1-5
      for (int h = 1; h<=5; h++) {
        System.out.println('\n'+"Test: " +h);
         rand_arr = getRandom(value_worst_C); //declaring array in worst counting case
                                                                //Empty array for storing random
integers
                                                                int[] empty_array = new
int[value_worst_C];
                                                                //Length of array is calculated
        int n = rand_arr.length;
        int k_value = (n*n/*+getMax(rand_arr)*//*Get max value*/);
```

```
                                                        //using nanoTime to get the current
time
            long countingStart = System.nanoTime();
            countingSort(rand_arr, empty_array, n, k_value); //Run mehtod for bad performance
            long countingEnd =  System.nanoTime();
                                                        //Find the time taken and records
the time for the algorithm
            System.out.println("Worst_Counting_Run-time "+(countingEnd - countingStart));
        }
     }
   }

                    //Generate data for Counting Best Case//
   public static void countingTest_bestcase() {
      int[] rand_arr ;
                                    //create a loop of array length starting from 1000 to 20000,
incrementing by 1000
      for (int value_best_C = 1000; value_best_C<=20000; value_best_C+=1000) {
                                    //for (int value_best_C = 5; value_best_C<=15; value_best_C+=5) {
         System.out.println("Array Length: " +value_best_C);
                                                //create a loop of data points from 1-5
         for (int h = 1; h<=5; h++) {
           System.out.println('\n'+"Test: " +h);
            rand_arr = getRandom(value_best_C); //declaring array in best counting case
                                                        //Empty array for storing random
integers
                                                        int[] empty_array = new
int[value_best_C];
                                                        //Length of array is calculated
            int n = rand_arr.length;
            int k_value = getMax(rand_arr); //getting the highest number in the array 'rand_arr'
                                                        //using nanoTime to get the current
time
            long countingStart = System.nanoTime();
            countingSort(rand_arr, empty_array, n, k_value); //run method for Good performance
            long countingEnd = System.nanoTime();
                                                        //Find the time taken and records
the time for the algorithm
            System.out.println("Best_Counting_Run-time "+(countingEnd - countingStart));
         }
      }
   }
}
```