Part- 1:

Analysis:

Insertion Sort:

a. Best Case:

The presorted input will become the best-case input for an insertion sort.

The two loops of insertion sort are as follows:

- the outer loop iterates (n-1) times from 2nd element to the last element
- For the above (n-1) loop iterations, the inner loop executes 1 time. It checks whether the (i-1) th element is greater than the i th element, if (i-1)>ith element then swapping occurs.

For the pre-sorted input, swapping doesn't take place as (i-1) th element is less than i th element. Therefore, taking less time to sort the pre-sorted input.

b. Average Case:

The randomly given input will become the average-case input for an insertion sort.

The integers are chosen randomly in the range (0,n-1) and are applied to the insertion sort, then on an average, half of the numbers are less than 'n' and half of the numbers are greater than 'n'. The resulting average-case running time turns out to be a quadratic function of the input size, just like the worst-case running time.

c. Worst Case:

The reverse sorted input will become the worst-case input for an insertion sort.

The two loops of insertion sort are as follows:

- the outer loop iterates (n-1) times from 2nd element to the last element
- For the above (n-1) loop iterations, the inner loop executes 1 time. It checks whether the (i-1) th element is greater than the i th element, if (i-1)>ith element then swapping occurs.

For the reverse-sorted input, swapping takes place as (i-1)th element is greater than ith element. Every element is swapped until a sorted order is obtained. Therefore, taking more time to sort the reverse sorted input.

Merge Sort:

Merge Sort uses the concept of Divide and Conquer. It first divides the input into equal halves and then combine them in a sorted manner.

a) **Best Case:**
   Time Complexity in best case is θ (n log n). When the largest element of a sub-array is smaller than the first element of the other sub-array, best case occurs. The total number

of comparisons made during the merge process will be reduce by half i.e, to N/2. Already sorted input forms the best case in merge sort.

b) Average Case:

The average time complexity of the merge sort is θ (n log n). It depends on input and it always divides the input into two halves and takes linear time to merge those two halves.

c) Worst Case:

Worst case scenario occurs when we have largest number of comparisons. In the worst case, every element in the left sub array and the right sub array of the divided array are compared at least once. The recurrence relation would be T (n) = 2 T (n/2) + n which can be solved using master's theorem to give T (n) = θ (n log n)

Counting Sort:

It is a stable sort i.e, multiple keys with same value are placed in sorted array in same order that they appear in input array. The basic idea of counting sort is to determine, for each input elements x, the number of elements less than x. Assuming that, each of the elements is an integer in the range 1 to k, for some integer k, when k=O(n), the counting sort runs O(n) times. Since, counting sort uses the actual values of the elements to index the array, there will be no comparisons made. Therefore, the complexity for best case, average case and worst case in Counting sort will be O(n).

2. Experimental Setup:

a) Machine used for all 3 sorting algorithms:

- Processor: Intel(R) Core™ i-7500U CPU @ 2.70GHz 2.90 GHz
- RAM: 16.00GB
- System type: 64 – bit Operating System, x64 – based processor.

b)

Timing mechanisms used:

The running of all the three algorithms (Insertion, Merge and Counting sorts) is calculated in terms of milliseconds.

c)

Insertion sort has been performed (repeated) 10 times on 11 different input sizes.

Merge and Counting sort has been performed (repeated) 10 times on 10 different input sizes.

d)

Reported times:

The experiment is repeated on 10-11 different inputs and the best case, average case and worst cases are noted. The run time has been calculated in terms of milliseconds. More details are there in the attached excel sheet in the project zip folder.

Insertion Sort:

Best Case: Run Time:

N	T average(millisec)
500	0.6
1000	1.4
2000	1.7
3000	1.7
4000	4.1
5000	4.5
6000	5.6
7000	4.4
8000	4.1
9000	4.5
10000	5.1

Average Case: Run Time:

N	T average(millisec)
500	7.3
1000	13.6
2000	18.8
3000	21.4
4000	20.9
5000	22.5
6000	26.2
7000	33.3
8000	33.8
9000	38.1
10000	51.1

Worst Case: Run Time:

N	T average(millisec)
500	8.7
1000	16.7
2000	16.8
3000	28.5
4000	26
5000	38.8
6000	43.3
7000	61.5
8000	68.3
9000	77.9
10000	90.7

Merge Sort:

Best Case: Run Time:

N	T average(millisec)
10000	4.8
20000	5.7
30000	9.9
40000	10.7
50000	10.8
60000	10.9
70000	11.2
80000	14.7
90000	15.7
100000	17.3

Average Case: Run Time:

N	T average(millisec)
10000	6.1
20000	8.9

30000	13.7
40000	11.4
50000	14.5
60000	15
70000	18
80000	21.3
90000	22.5
100000	21.9

Worst Case: Run Time:

N	T average(millisec)
10000	5.2
20000	11.1
30000	11.2
40000	13.7
50000	13.8
60000	13.6
70000	15.7
80000	17.1
90000	17.3
100000	23.5

Counting Sort:

Best Case: Run Time:

N	T average(millisec)
10000	40.7
20000	36.4
30000	32.8
40000	34.4
50000	36.7
60000	34.4

70000	36
80000	43.7
90000	44.9
100000	45.2

Average Case: Run Time:

N	T average(millisec)
10000	34.4
20000	34.7
30000	38.9
40000	43.8
50000	48.3
60000	39
70000	48
80000	51.5
90000	48.5
100000	46.5

Worst Case: Run Time:

N	T average(millisec)
10000	36
20000	31.7
30000	39
40000	42.3
50000	43.6
60000	41.8
70000	43.3
80000	43.4
90000	40.4
100000	45.3

e) Inputs are randomly generated for the given size. If we want 1000 numbers in input, RandomInputgenerator.java will generate 1000 numbers accordingly. We have used inputs ranging from 500-100000 numbers in single input file.

In all the three algorithms (Insertion, Merge and Counting sort),

- The average case has been calculated through the randomly generated numbers using RandomInputgenerator.java.
- The best case has been calculated by sorting the randomly generated numbers using Presort.java and that presorted output is given as input for every algorithm.
- The worst case has been calculated by reverse sorting the randomly generated numbers using reverseSort.java and that reverse sorted output is given as input for each algorithm.

f) The input size is same for all the three sorting algorithms, the numbers in the input file changes.

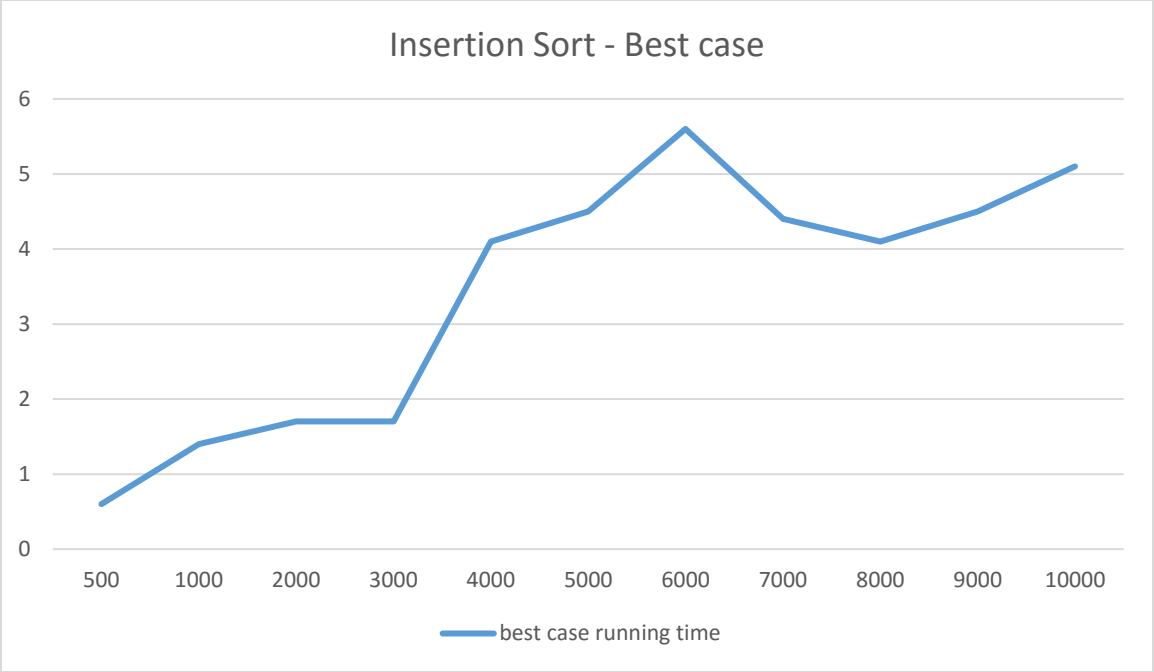
3.

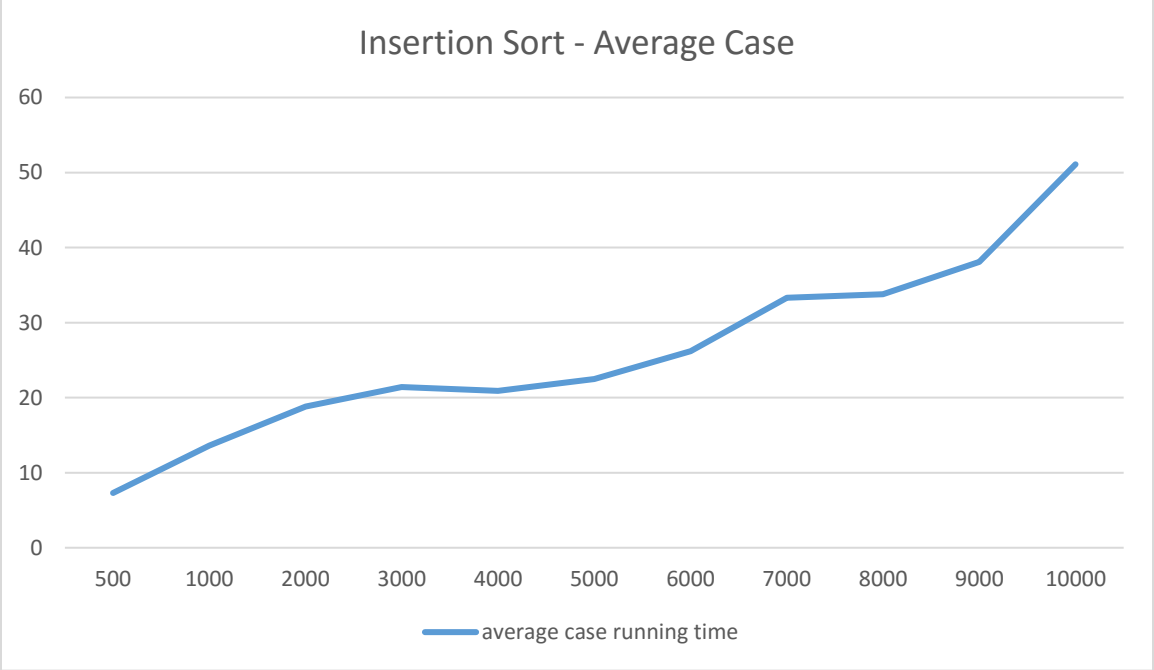
For all the graphs that follows:

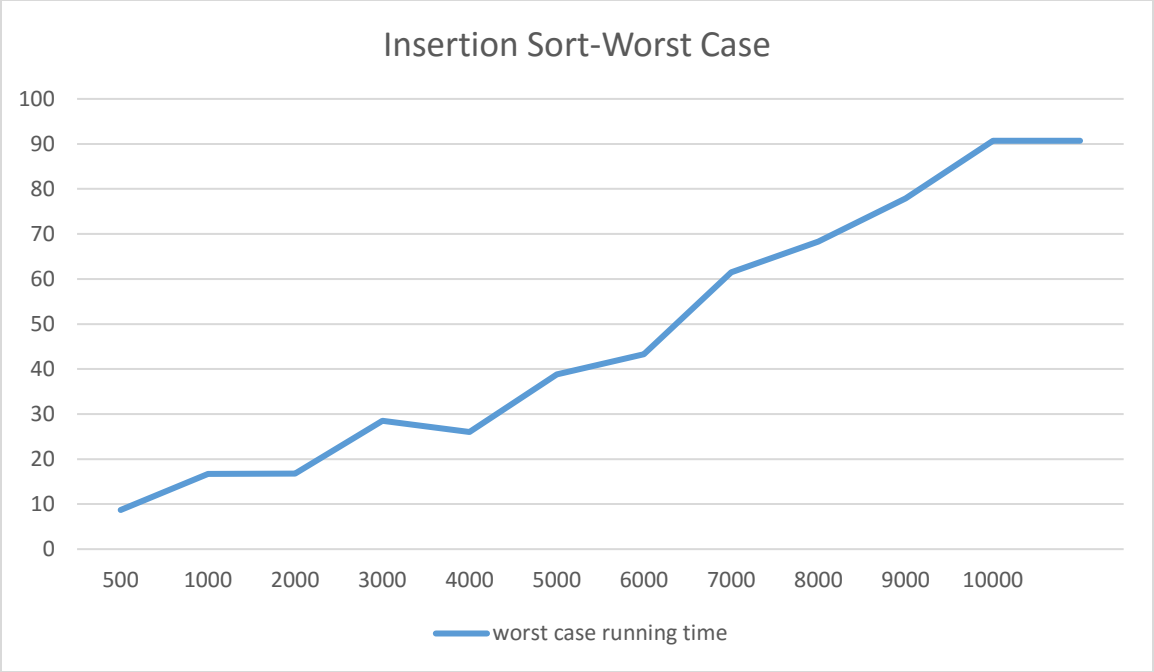
- The X-axis represents the input size
- The Y-axis represents the average running time (in millisecs) in each case

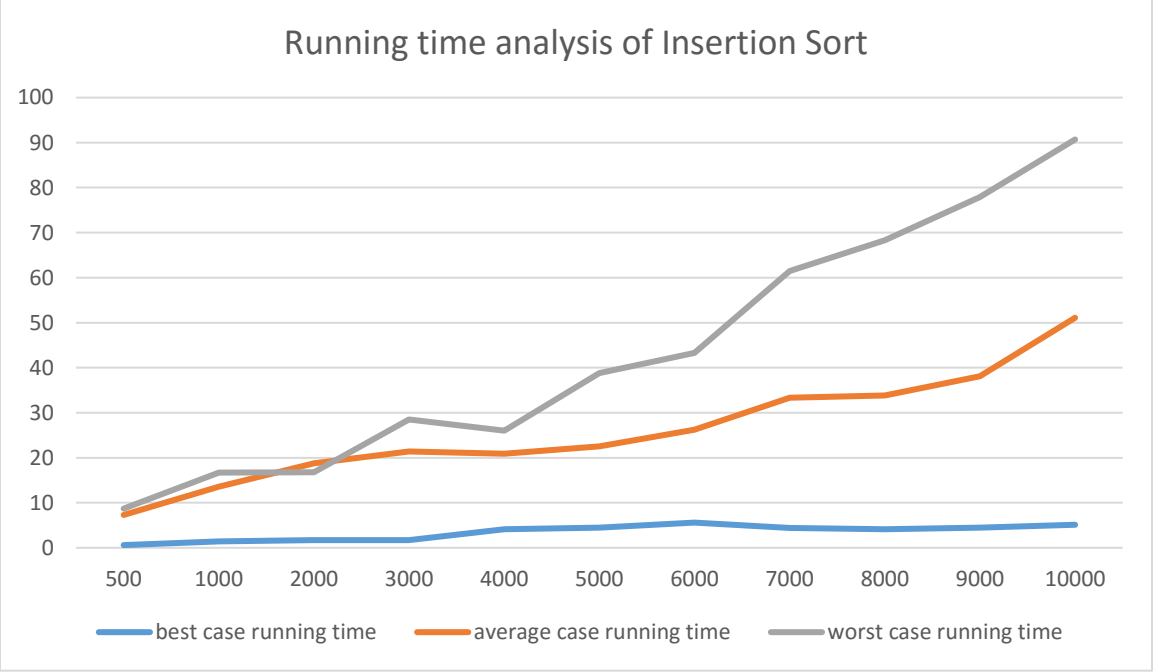
Insertion Sort:

N	best case running time	average case running time	worst case running time
500	0.6	7.3	8.7
1000	1.4	13.6	16.7
2000	1.7	18.8	16.8
3000	1.7	21.4	28.5
4000	4.1	20.9	26
5000	4.5	22.5	38.8
6000	5.6	26.2	43.3
7000	4.4	33.3	61.5
8000	4.1	33.8	68.3
9000	4.5	38.1	77.9
10000	5.1	51.1	90.7


Insertion Sort - Best case
6
5
4
3
2
1
0
500
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
best case running time

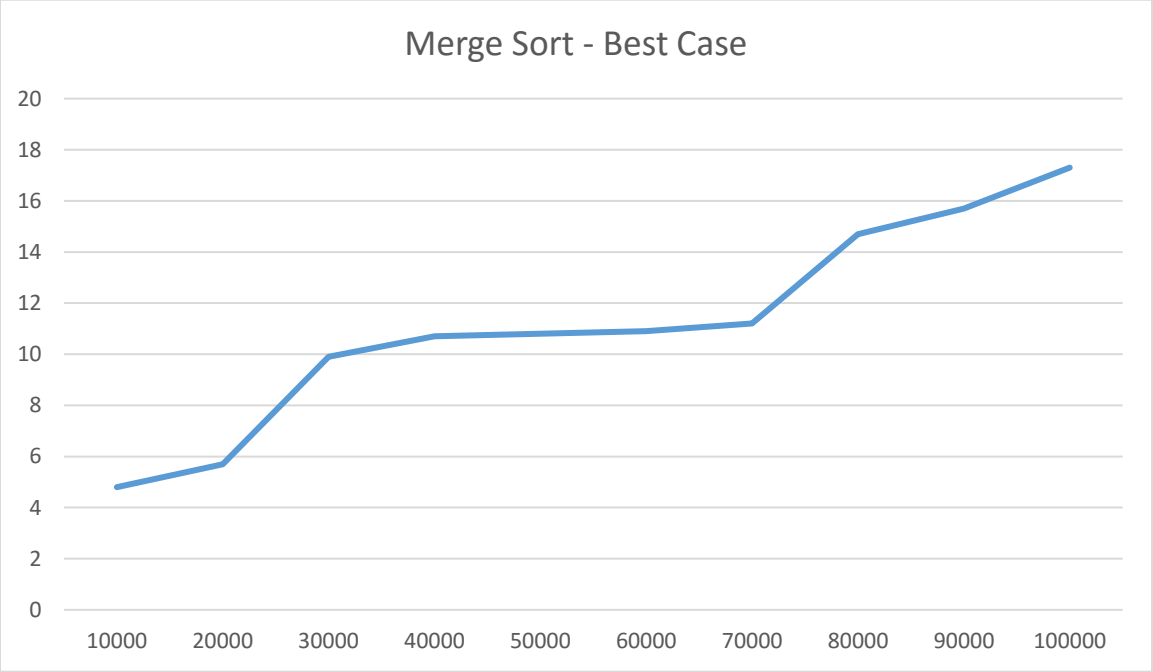

Insertion Sort - Average Case
60
50
40
30
20
10
0
500
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
average case running time

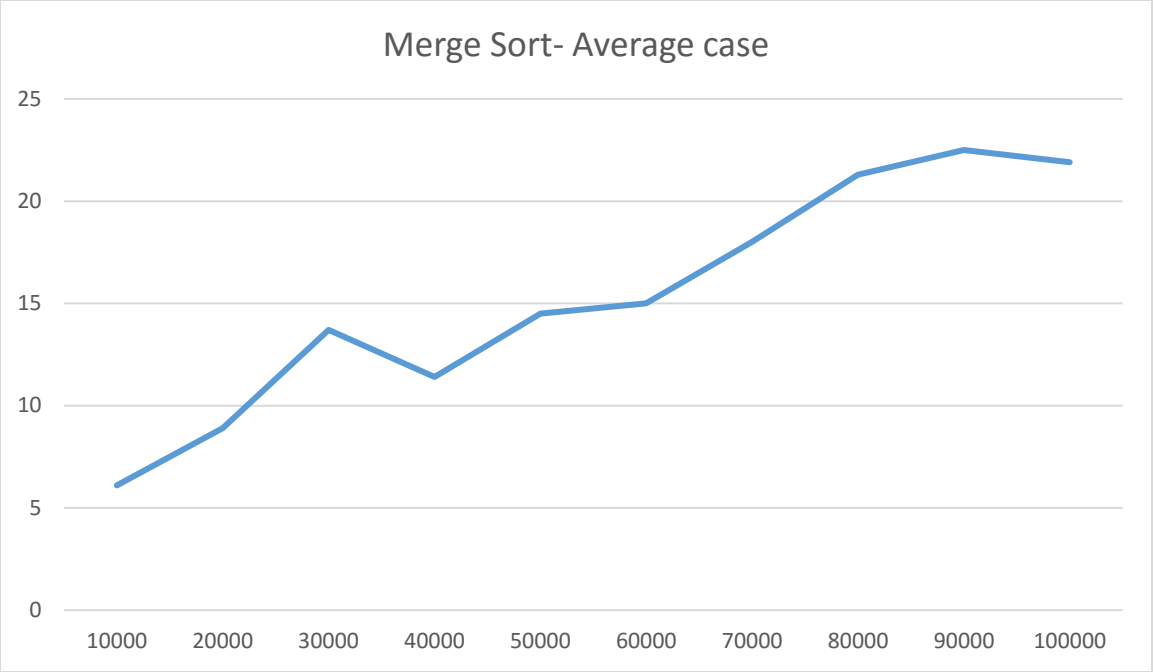

Insertion Sort-Worst Case
100
90
80
70
60
50
40
30
20
10
0
500
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
worst case running time

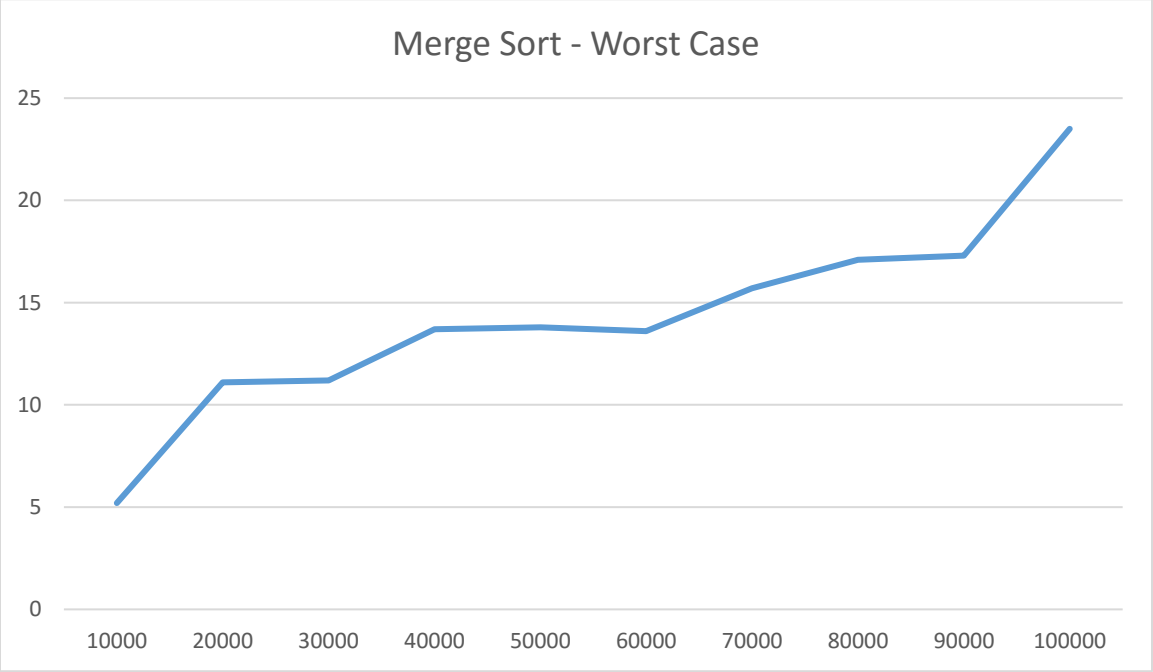

Running time analysis of Insertion Sort
100
90
80
70
60
50
40
30
20
10
0
500
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
best case running time
average case running time
worst case running time

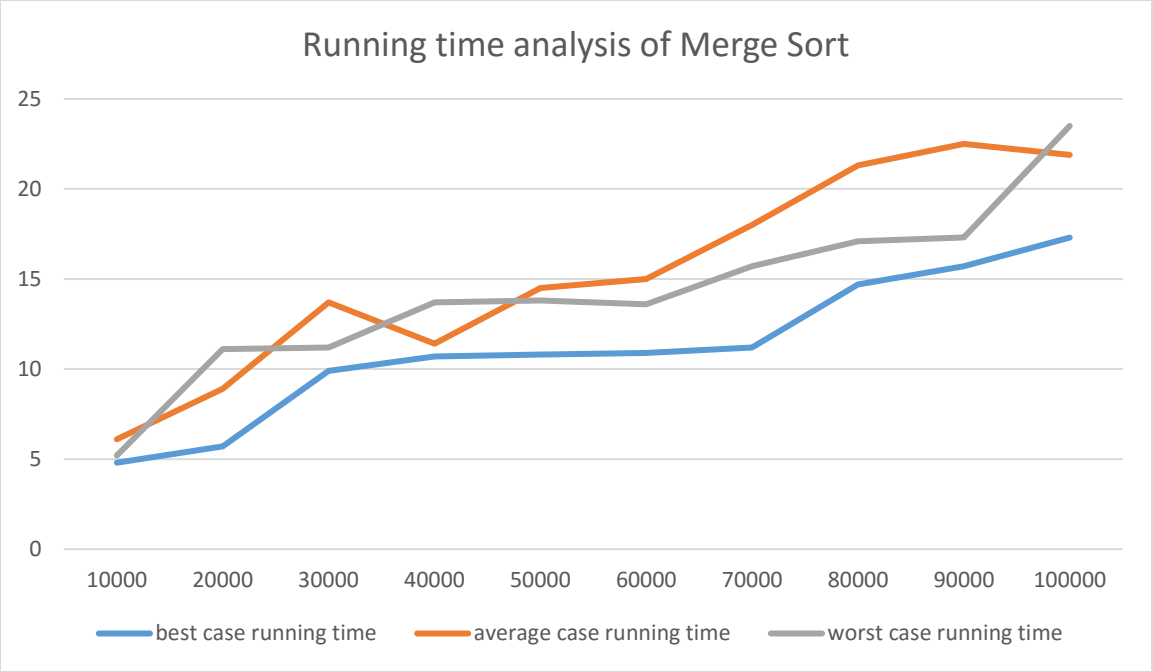
Merge Sort:

N	best case running time	average case running time	worst case running time
10000	4.8	6.1	5.2
20000	5.7	8.9	11.1
30000	9.9	13.7	11.2
40000	10.7	11.4	13.7
50000	10.8	14.5	13.8
60000	10.9	15	13.6
70000	11.2	18	15.7
80000	14.7	21.3	17.1
90000	15.7	22.5	17.3
100000	17.3	21.9	23.5




Merge Sort- Average case
25
20
15
10
5
0
10000
20000
30000
40000
50000
60000
70000
80000
90000
100000

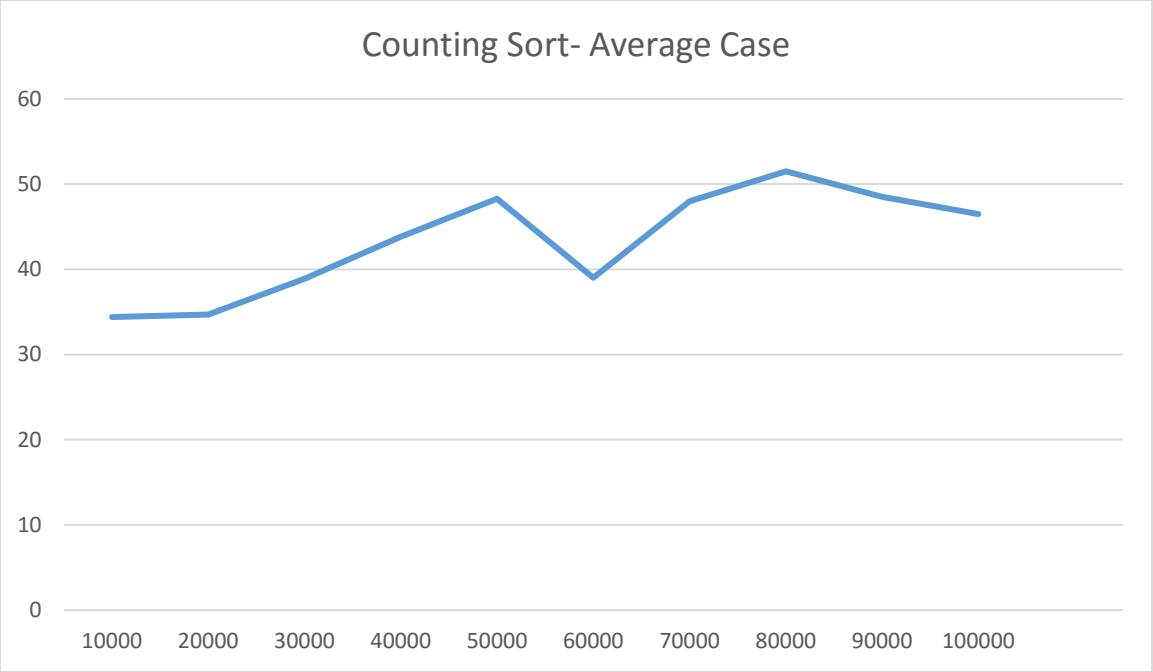

Merge Sort - Worst Case
25
20
15
10
5
0
10000
20000
30000
40000
50000
60000
70000
80000
90000
100000

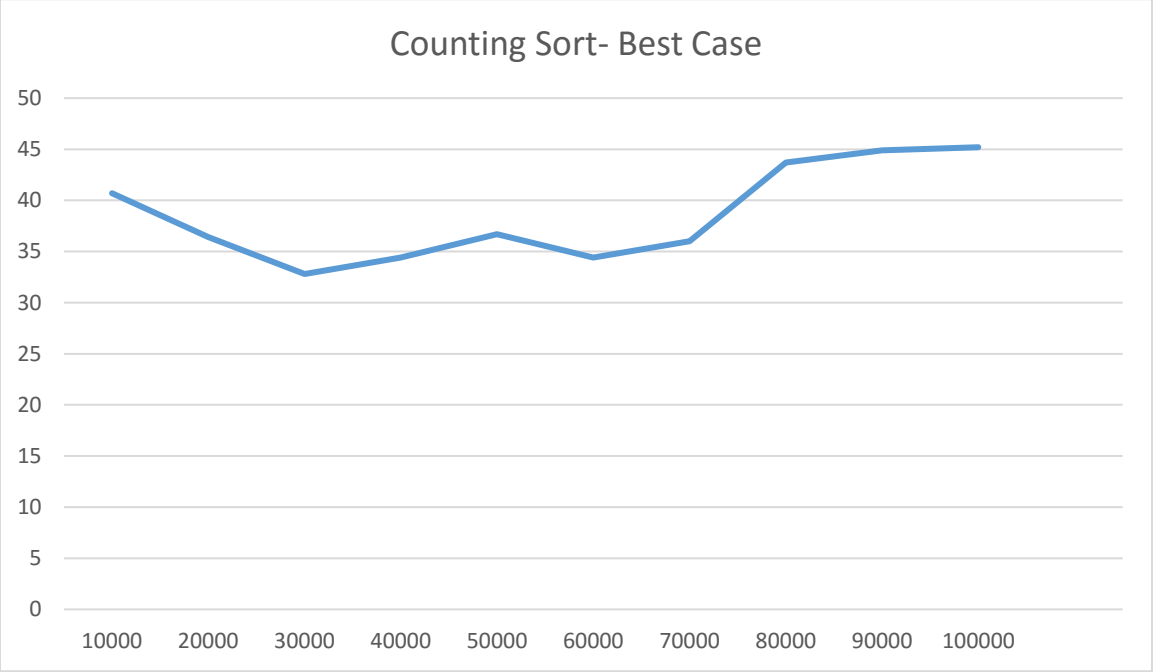


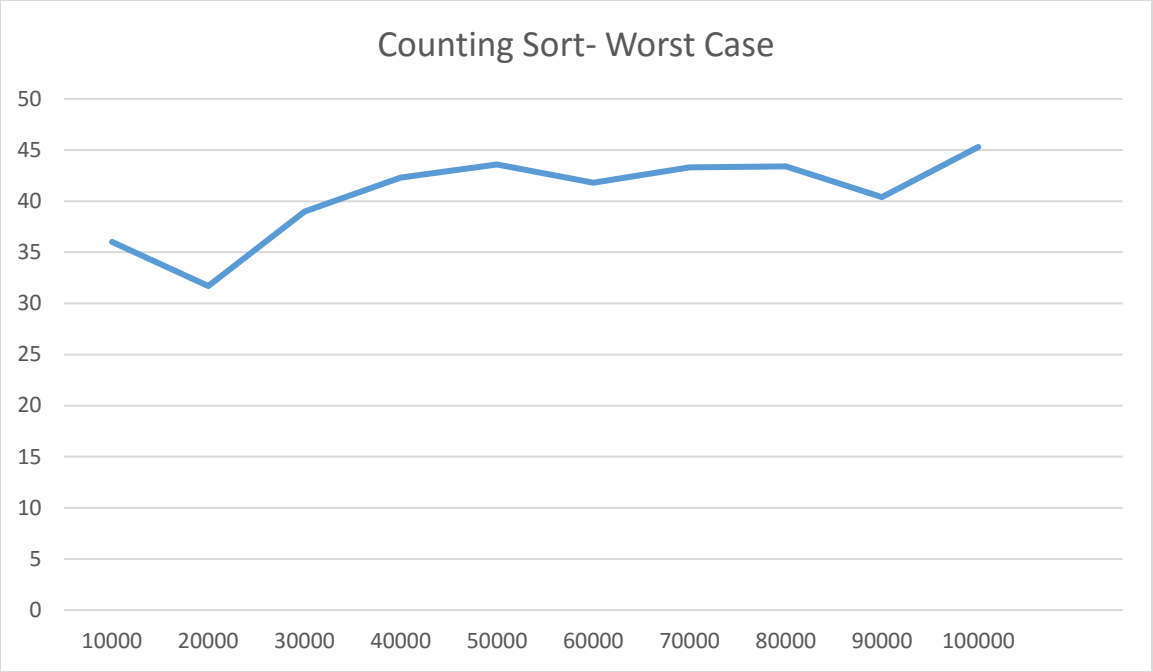
For merge sort, best, average and worst cases have the same time complexity. As per the experimental results, some inexplicable results favored the worst case than the average case.

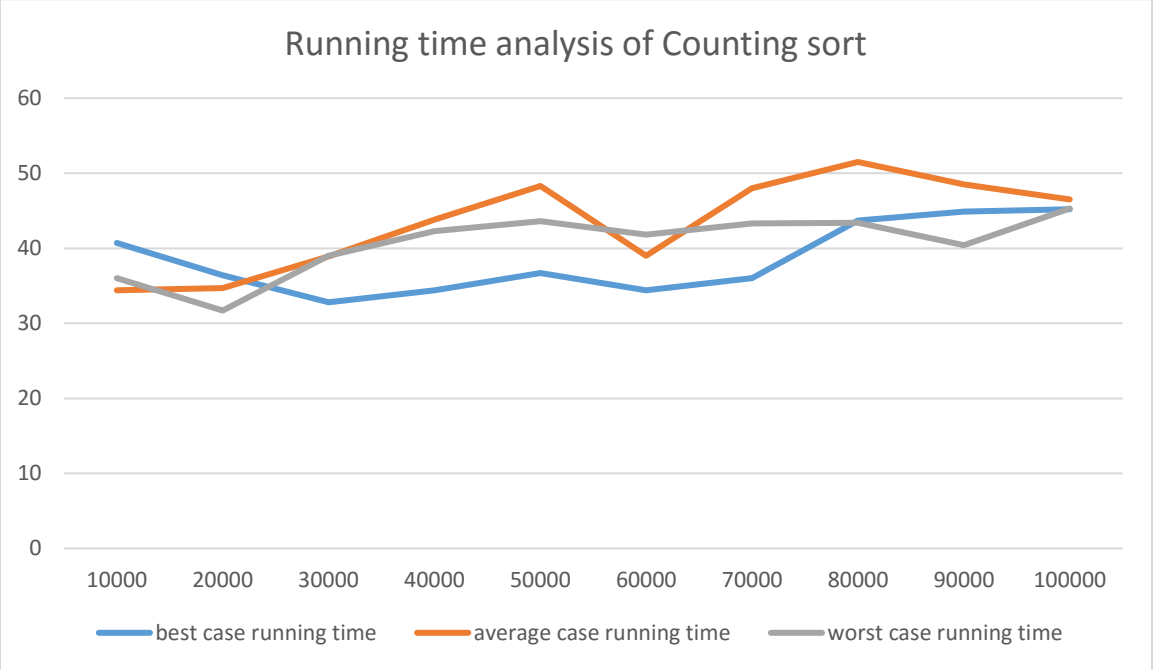
Counting Sort:

N	best case running time	average case running time	worst case running time
10000	40.7	34.4	36
20000	36.4	34.7	31.7
30000	32.8	38.9	39
40000	34.4	43.8	42.3
50000	36.7	48.3	43.6
60000	34.4	39	41.8
70000	36	48	43.3
80000	43.7	51.5	43.4
90000	44.9	48.5	40.4
100000	45.2	46.5	45.3


Counting Sort- Average Case
60
50
40
30
20
10
0
10000
20000
30000
40000
50000
60000
70000
80000
90000
100000


Counting Sort- Best Case
50
45
40
35
30
25
20
15
10
5
0
10000
20000
30000
40000
50000
60000
70000
80000
90000
100000





For merge sort, best, average and worst cases have the same time complexity. As per the experimental results, some inexplicable results favored the worst case than the average case.

From the above running times, we can infer that Merge sort seems to perform well. For larger input values, the complexity of merge sort is better compared to other sorts as logn in O(nlogn) vanishes.

4)

a) As we can see from the experimental running time in the above question (depiction through tables and graphs), it is of the same order as predicted by the asymptotic analysis.

For **Insertion** sort,

The running time of best case < average case < worst case

Therefore, the efficiency of best case > average case > worst case

Theoretically, the time complexity for insertion sort is as follows,

Best case – O(n)

Average case - O(n^2)

Worst case - O(n^2)

For **Merge** sort,

The running time of best case < average case < worst case

Therefore, the efficiency of best case > average case > worst case

Theoretically, the time complexity for merge sort is as follows,

Best case – O(nlog(n))

Average case - O(nlog(n))

Worst case – O(nlog(n))

For **Counting** sort,

The running time of best case < average case < worst case

Therefore, the efficiency of best case > average case > worst case

Theoretically, the time complexity for counting sort is as follows,

Best case – O(n+k) → O(n)

Average case - O(n+k) → O(n)

Worst case - O(n+k) → O(n)

b) Determining constants and n0:

For all the graphs that follows:

- The X-axis represents the input size
- The Y-axis represents the (time/complexity) in each case

Insertion Sort:

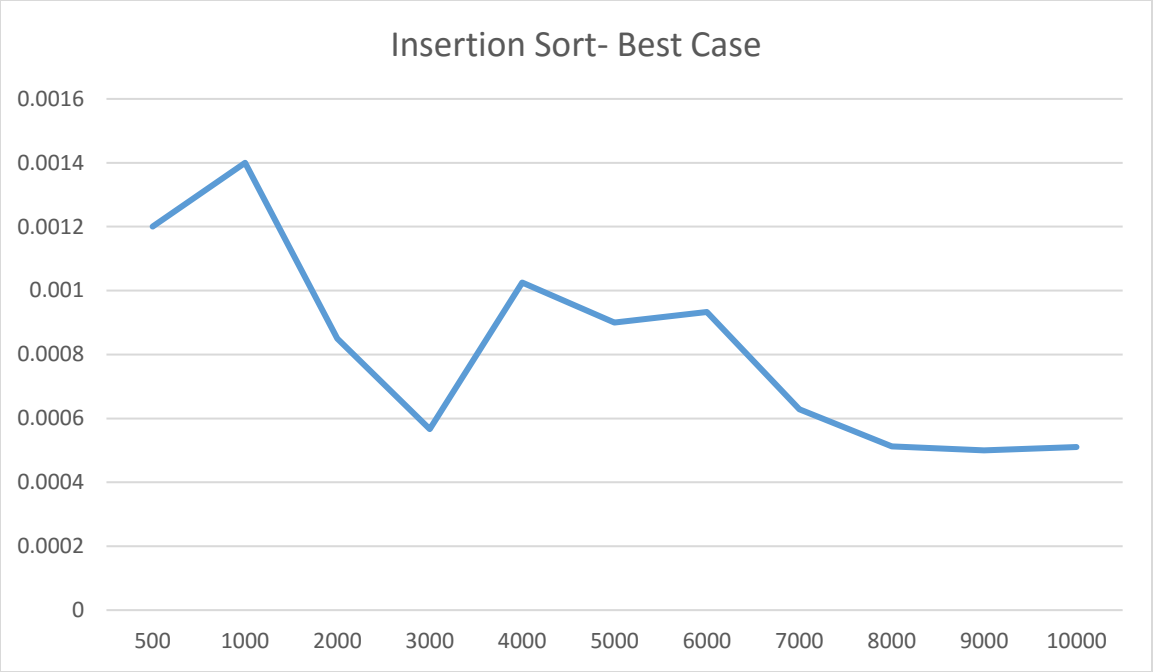
Best case:

The time complexity of insertion sort is O(n) in best case

- X- axis denotes input size n.
- Y axis denotes time/complexity i.e, (best case running time)/n

To calculate c, we'll have to divide time with complexity

N	C=time/n	best case running time
500	0.0012	0.6
1000	0.0014	1.4
2000	0.00085	1.7
3000	0.000566667	1.7
4000	0.001025	4.1
5000	0.0009	4.5
6000	0.000933333	5.6
7000	0.000628571	4.4
8000	0.0005125	4.1
9000	0.0005	4.5
10000	0.00051	5.1



n0 value in the best case asymptotic analysis of insertion sort is 8000.

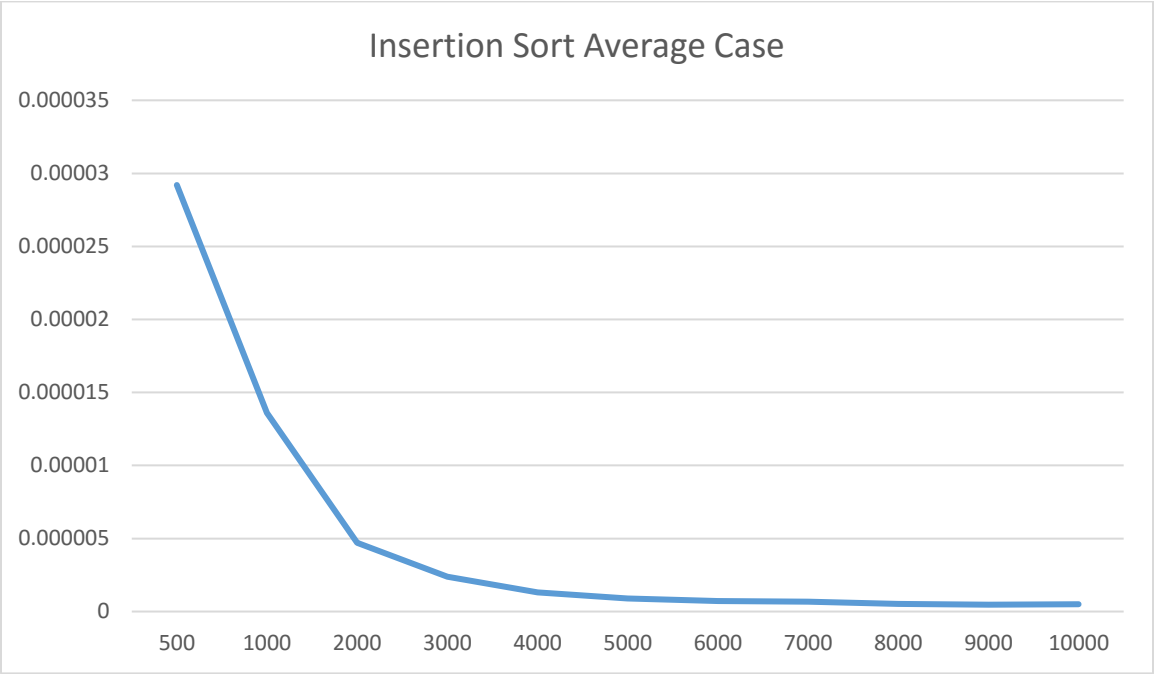
Average case:

The time complexity of insertion sort is O(n^2) in average case

- X- axis denotes input size n.
- Y axis denotes time/complexity i.e, (average case running time)/n^2

To calculate c, we'll have to divide time with complexity

N	average case running time	C=time/n^2
500	7.3	0.0000292
1000	13.6	0.0000136
2000	18.8	0.0000047
3000	21.4	2.38E-06
4000	20.9	1.31E-06
5000	22.5	0.0000009
6000	26.2	7.28E-07
7000	33.3	6.80E-07
8000	33.8	5.28E-07
9000	38.1	4.70E-07
10000	51.1	0.000000511



n0 value in the average case asymptotic analysis of insertion sort is 3000.

Worst case:

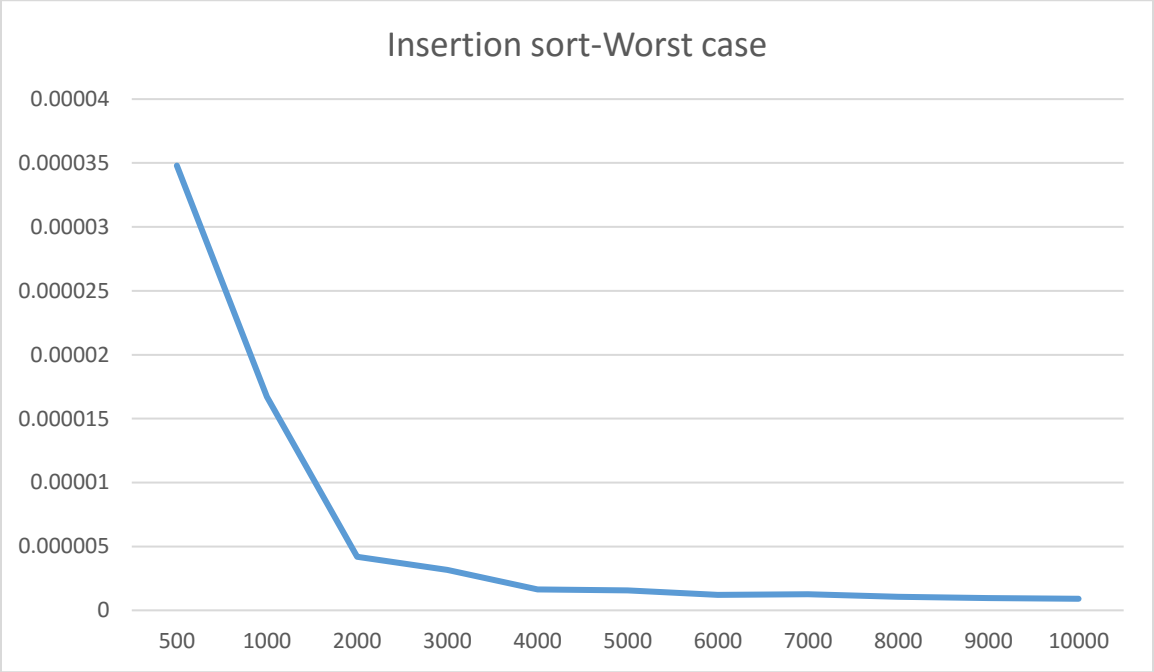
The time complexity of insertion sort is O(n^2) in worst case

- X- axis denotes input size n.
- Y axis denotes time/complexity i.e, (worst case running time)/n^2

To calculate c, we'll have to divide time with complexity

N	worst case running time	C=time/n^2
500	8.7	0.0000348
1000	16.7	0.0000167
2000	16.8	0.0000042
3000	28.5	3.17E-06
4000	26	0.000001625
5000	38.8	0.000001552
6000	43.3	1.20E-06
7000	61.5	1.26E-06
8000	68.3	1.07E-06

9000	77.9	9.62E-07
10000	90.7	0.000000907



n0 value in the worst case asymptotic analysis of insertion sort is 4000.

Merge Sort:

Best case:

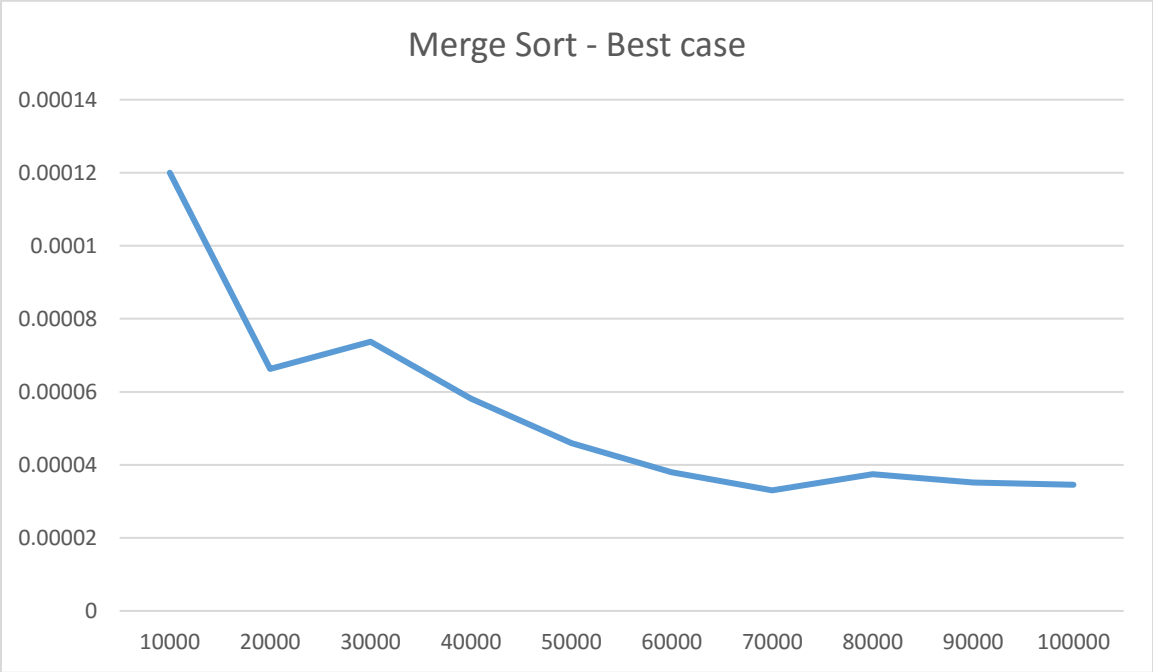
The time complexity of merge sort is O(nlogn) in best case

- X- axis denotes input size n.
- Y axis denotes time/complexity i.e, (best case running time)/nlogn

To calculate c, we'll have to divide time with complexity

N	best case running time	C=time/nlogn
10000	4.8	0.00012
20000	5.7	6.62632E-05
30000	9.9	7.37081E-05
40000	10.7	5.81261E-05

50000	10.8	4.59675E-05
60000	10.9	3.80203E-05
70000	11.2	3.30231E-05
80000	14.7	3.74764E-05
90000	15.7	3.52111E-05
100000	17.3	0.0000346



n0 value in the best case asymptotic analysis of merge sort is 8000.

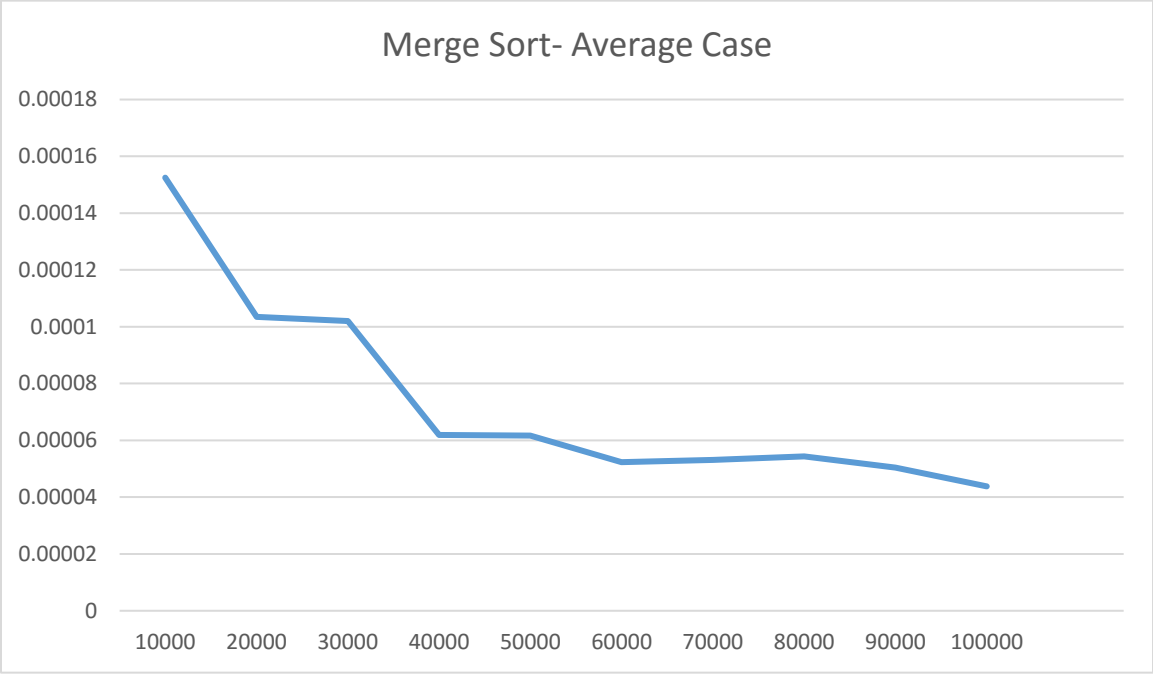
Average case:

The time complexity of merge sort is O(nlogn) in average case

- X- axis denotes input size n.
- Y axis denotes time/complexity i.e, (average case running time)/nlogn

To calculate c, we'll have to divide time with complexity

N	average case running time	C=time/nlogn
10000	6.1	0.0001525
20000	8.9	0.000103464
30000	13.7	0.000102
40000	11.4	6.19288E-05
50000	14.5	6.17157E-05
60000	15	5.23215E-05
70000	18	5.30728E-05
80000	21.3	5.43025E-05
90000	22.5	5.04618E-05
100000	21.9	0.0000438



n0 value in the average case asymptotic analysis of merge sort is 6000.

Worst case:

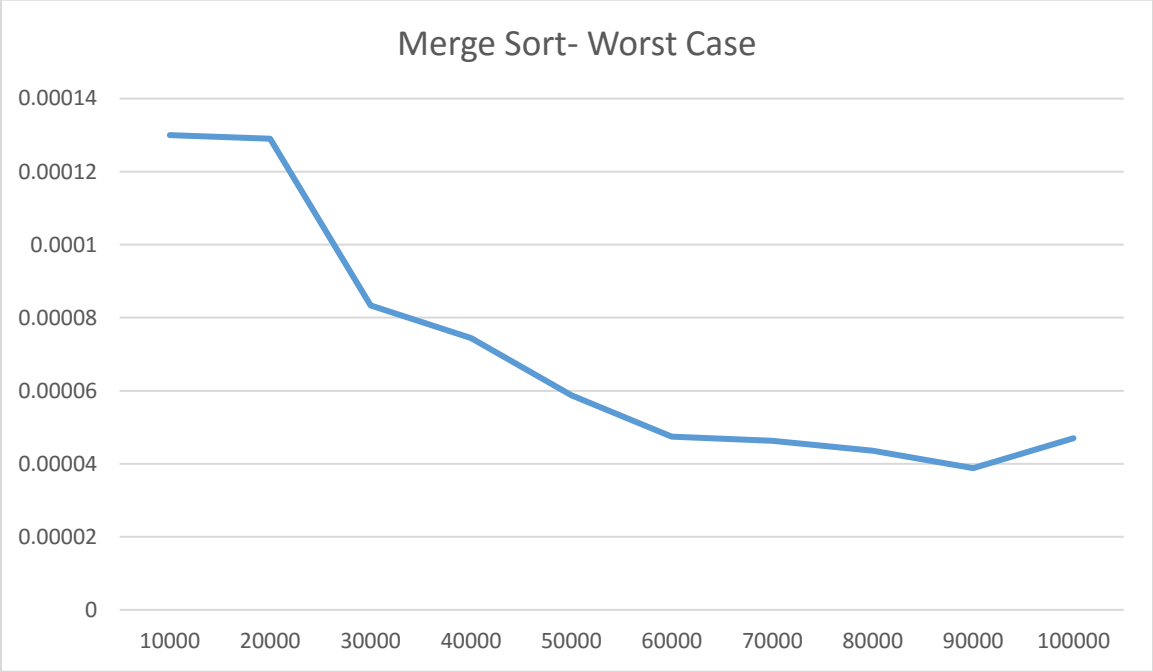
The time complexity of merge sort is O(nlogn) in worst case

- X- axis denotes input size n.
- Y axis denotes time/complexity i.e, (worst case running time)/nlogn

To calculate c, we'll have to divide time with complexity

N	worst case running time	C=time/nlogn

10000	5.2	0.00013
20000	11.1	0.000129
30000	11.2	8.34E-05
40000	13.7	7.44E-05
50000	13.8	5.87E-05
60000	13.6	4.74E-05
70000	15.7	4.63E-05
80000	17.1	4.36E-05
90000	17.3	3.88E-05
100000	23.5	0.000047



n0 value in the worst case asymptotic analysis of merge sort is 10000.

Counting Sort:

Best case:

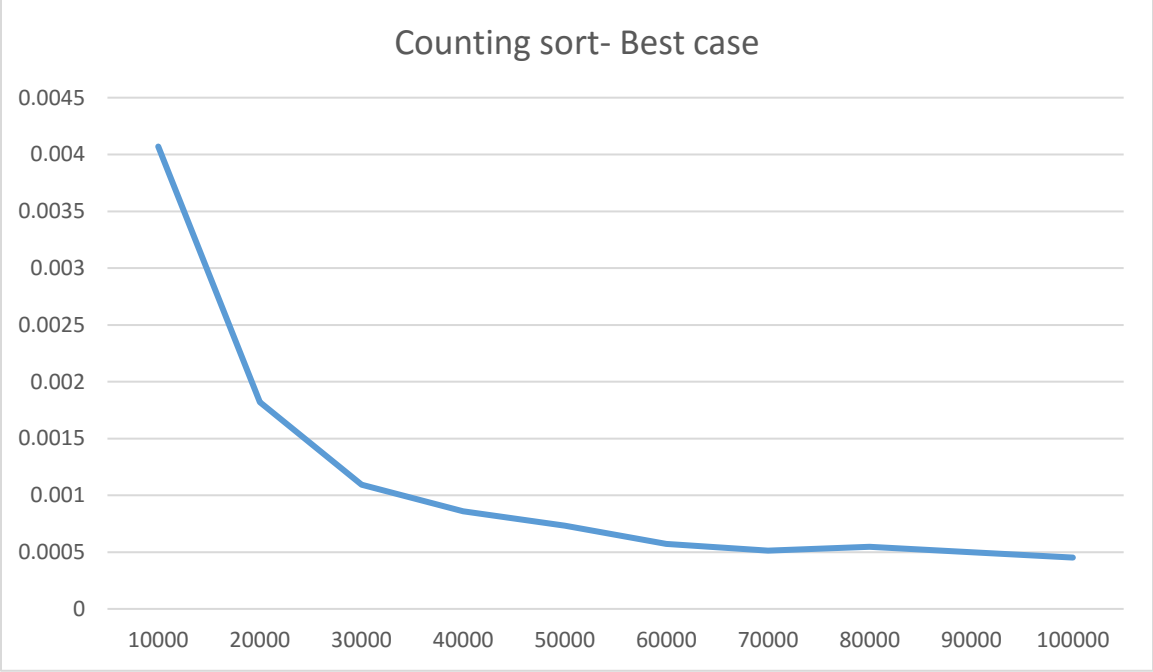
The time complexity of counting sort is O(n) in best case

- X- axis denotes input size n.
- Y axis denotes time/complexity i.e, (best case running time)/n

To calculate c, we'll have to divide time with complexity

N	best case running time	time/n
10000	40.7	0.00407

20000	36.4	0.00182
30000	32.8	0.001093333
40000	34.4	0.00086
50000	36.7	0.000734
60000	34.4	0.000573333
70000	36	0.000514286
80000	43.7	0.00054625
90000	44.9	0.000498889
100000	45.2	0.000452



n0 value in the best case asymptotic analysis of counting sort is 8000.

Average case:

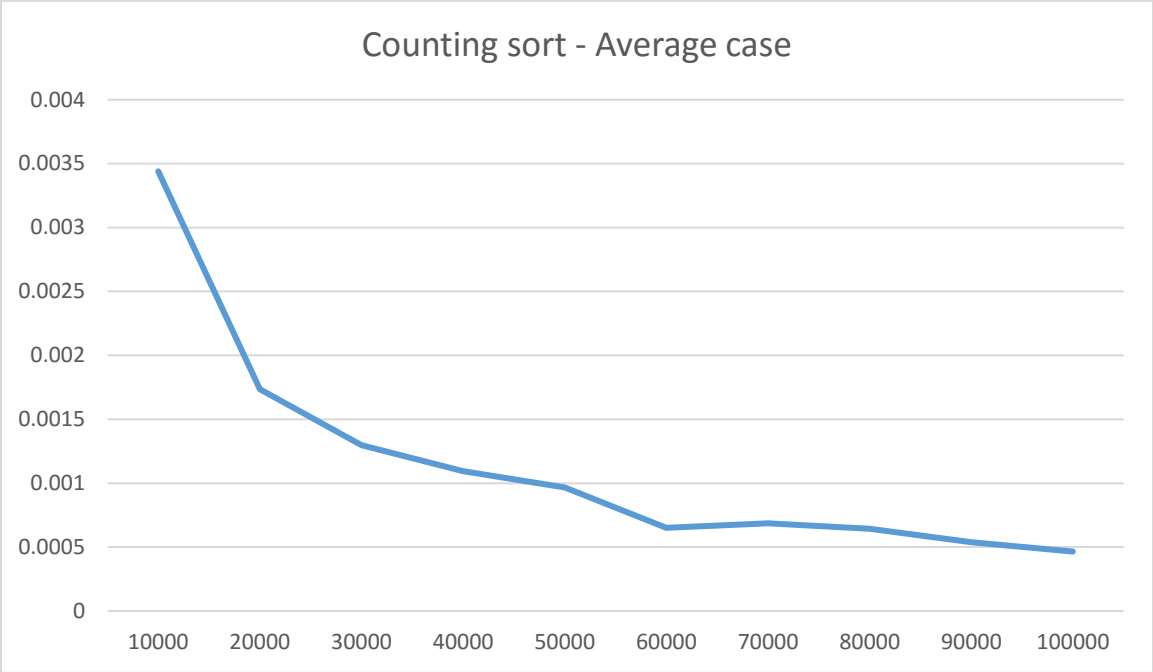
The time complexity of counting sort is O(n) in average case

- X- axis denotes input size n.
- Y axis denotes time/complexity i.e, (average case running time)/n

To calculate c, we'll have to divide time with complexity

N	average case running time	time/n
10000	34.4	0.00344
20000	34.7	0.001735

30000	38.9	0.001297
40000	43.8	0.001095
50000	48.3	0.000966
60000	39	0.00065
70000	48	0.000686
80000	51.5	0.000644
90000	48.5	0.000539
100000	46.5	0.000465



n0 value in the average case asymptotic analysis of counting sort is 7000.

Worst case:

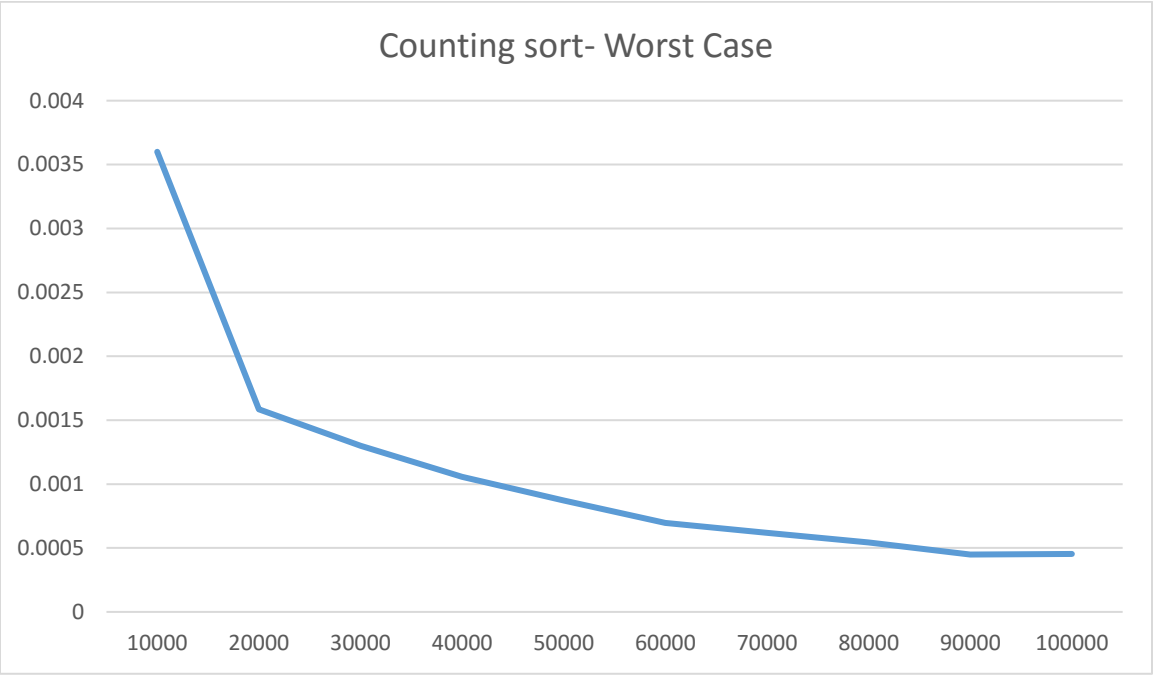
The time complexity of counting sort is O(n) in worst case

- X- axis denotes input size n.
- Y axis denotes time/complexity i.e, (worst case running time)/n

To calculate c, we'll have to divide time with complexity

N	worst case running time	time/n
10000	36	0.0036

20000	31.7	0.001585
30000	39	0.0013
40000	42.3	0.001058
50000	43.6	0.000872
60000	41.8	0.000697
70000	43.3	0.000619
80000	43.4	0.000543
90000	40.4	0.000449
100000	45.3	0.000453



n0 value in the worst case asymptotic analysis of counting sort is 9000.