Hacettepe University

Computer Engineering Department

BBM204 Software Lab II – 2022 Spring

Assignment 1: Algorithm Complexity Analysis

11.03.2022

Hasan Furkan Günay

21827464

Introduction

Analysis of algorithms is the area of computer science that provides tools to analyze the efficiency of different methods of solutions. Efficiency of an algorithm depends on these parameters; i) how much time, ii) memory space, iii) disk space it requires. Analysis of algorithms is mainly used to predict performance and compare algorithms that are developed for the same task. Also it provides guarantees for performance and helps to understand theoretical basis.

Objective

In this assignment, we were asked to implement various sort algorithms, explain how they work, and make inferences about runtimes. With the help of experimental analysis, we were expected to have proved the the complexity and plot the reasults.

Experiments on the Given Random Data

Algorithms	512	1024	2048	4096	8192	16384	32768	65536	131072	251281
Insertion	0.00	0.10	0.60	0.80	1.20	5.30	19.70	69.20	300.60	1219.40
Merge	0.00	0.00	1.60	3.10	7.80	18.80	59.30	203.10	773.20	3030.60
Pigeonhole	168.60	177.60	180.40	181.90	182.60	184.90	191.30	193.40	194.60	197.80
Counting	138.40	140.7	141.6	142.3	143.4	144.5	146.4	147.6	148.7	149.8

Experiments on the Sorted Data

Algorithms	512	1024	2048	4096	8192	16384	32768	65536	131072	251281
Insertion	0.00	0.00	0.10	0.10	0.10	0.10	0.10	0.10	0.20	0.50
Merge	0.60	1.50	1.50	1.60	6.30	17.20	56.20	203.10	788.80	2939.60
Pigeonhole	168.60	177.60	180.40	181.90	182.60	184.90	188.30	191.40	194.60	197.80
Counting	138.40	140.7	141.6	142.3	143.4	144.5	146.4	147.6	148.7	149.8

Experiments on the Reversely Sorted Data

Algorithms	512	1024	2048	4096	8192	16384	32768	65536	131072	251281
Insertion	0.10	0.20	0.40	1.20	2.10	8.90	34.30	139.50	568.20	2134.00
Merge	0.00	0.00	1.60	3.10	6.90	16.30	60.80	209.10	785.8	2958.90
Pigeonhole	168.60	177.60	180.40	181.90	184.60	185.90	191.30	193.40	194.60	197.80
Counting	138.40	140.7	141.6	142.3	143.4	144.5	146.4	147.6	148.7	149.8

Complexity Analysis and Result Plotting

Algorithms	Best Case	Average Case	Worst Case
Insertion Sort	O(n)	O(n^2)	O(n^2)
Merge Sort	O(n log n)	O(n log n)	O(n log n)
Pigeonhole Sort	O(n + range)	O(n + range)	O(n + range)
Counting Sort	O(n + k)	O(n + k)	O(n + k)

Algorithms	Axuiliary Space Complexity
Insertion Sort	O(1)
Merge Sort	O(n)
Pigeonhole Sort	O(n+k)
Counting Sort	O(n + k)

For Insertion Sort:

key ← A[j]

If the input array is already in sorted order, insertion sort compares O(n) elements and performs no swaps. Therefore, in the best case, insertion sort runs in O(n) time.

The worst case for insertion sort will occur when the input list is in decreasing order.

Insertion sort only takes O(1) auxiliary space complexity. It sorts the entire array just by using an extra variable. (the line in the above)

For Merge Sort:

left ← MERGESORT(left)
right ← MERGESORT(right)

To merge the subarrays, made by dividing the original array of n elements, a running time of O(n) will be required. Also when we divide the list to subarrays this gives us O(log n)

Merge Sort uses O(n) auxiliary space because of the recursive calls. (two lines in the above)

For Pigeonhole Sort:

output ← [ ]
holes: array of arrays ← [[ ], [ ], . . ., [ ]] (total of range elements)

It takes O(n + range) time to complete, with n denoting the number of items in the input array and 'range' defines the number of potential values in the array.

The pigeonhole sort algorithm requires two arrays. Because the first one must store the input elements, its size is n. The pigeonhole array, which has a size equal to range k, is the second. Overall, the complexity of space is O(n+k). (two lines in the above)

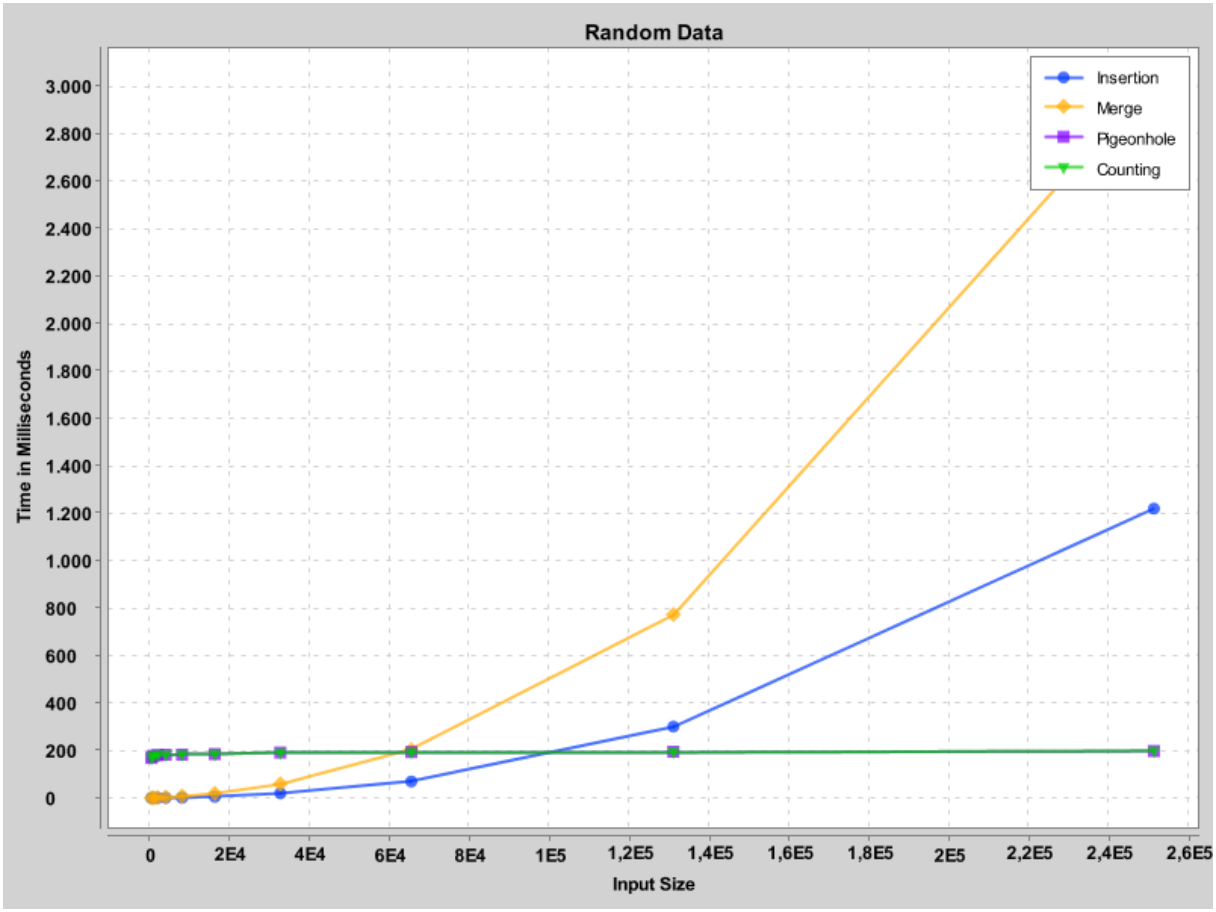
For Counting Sort:

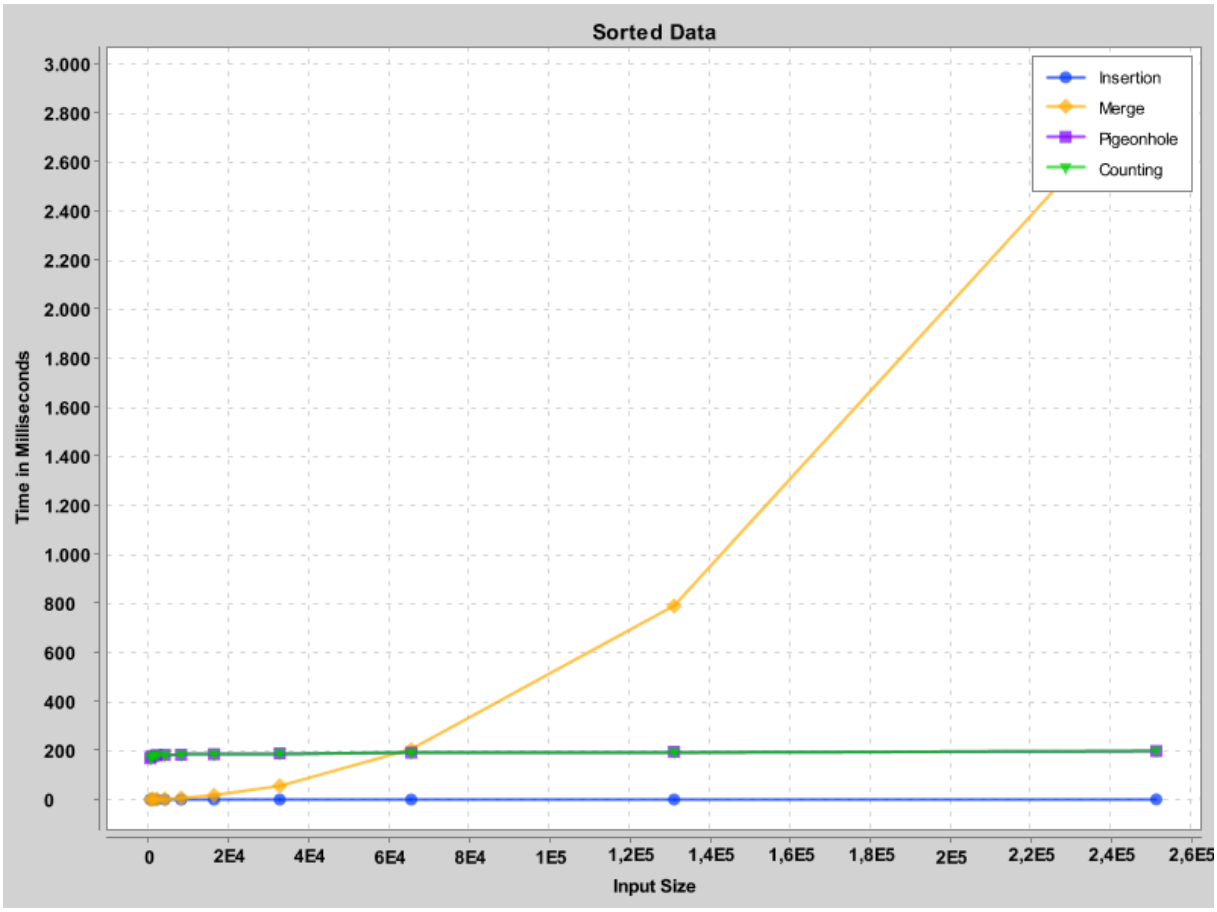
count ← array of k + 1 zeros

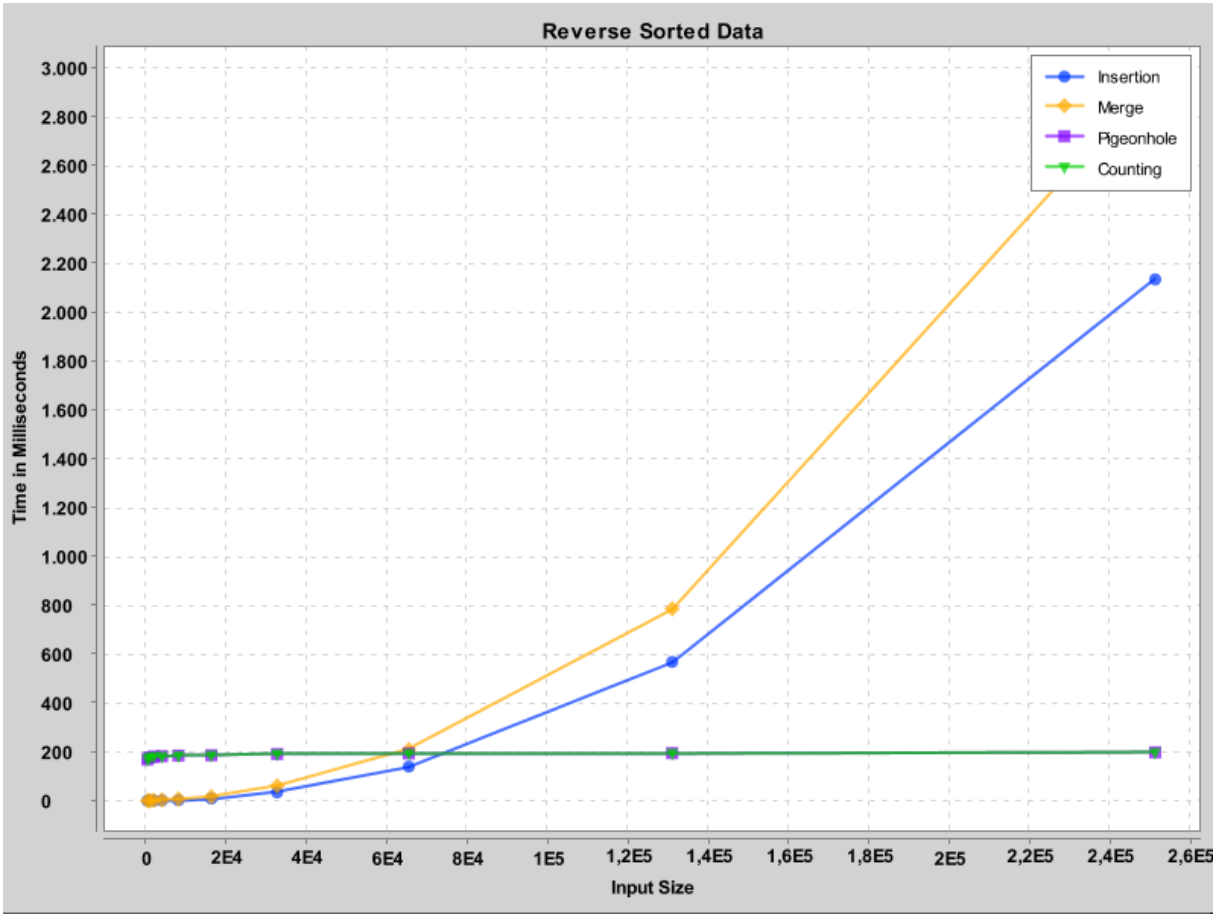
output ← array of the same length as A

The first loop goes through array, which has n elements. This step has a O(n) running time. The second loop iterates over k*k*, so this step has a running time of O(k). Therefore, the counting sort algorithm has a running time of O(n+k).

Also two lines in the above gives us space complexity O(n+k).


Random Data
Insertion
Merge
Pigeonhole
Counting
Time in Milliseconds
3.000
2.800
2.600
2.400
2.200
2.000
1.800
1.600
1.400
1.200
1.000
800
600
400
200
0
0
2E4
4E4
6E4
8E4
1E5
1,2E5
1,4E5
1,6E5
1,8E5
2E5
2,2E5
2,4E5
2,6E5
Input Size


Sorted Data
Insertion
Merge
Pigeonhole
Counting
Time in Milliseconds
3.000
2.800
2.600
2.400
2.200
2.000
1.800
1.600
1.400
1.200
1.000
800
600
400
200
0
0
2E4
4E4
6E4
8E4
1E5
1,2E5
1,4E5
1,6E5
1,8E5
2E5
2,2E5
2,4E5
2,6E5
Input Size



Result Analysis and Discussion

Q1: What are the best, average, and worst cases for the given algorithms in terms of the given input data to be sorted?

As you can see in the sorted data table and plot for the insertion sort best case is O(n). When we ran the algorithm on the sorted data, we got this result.

Insertion sorting is O(n^2) in the worst case and O(n^2) in the average case. When we ran the algorithm using reversed sorted data and random data, we got this result. The reverse sort data plot and the random data plot both show this result.

The pigeonhole sort's best, worst, and average cases are all the same and take O(n + range). In this algorithm we don’t care whether data is sorted or not. Because "range" determines complexity. This may be seen in the graphs above.

The pigeonhole and counting sort algorithms are very similar. The crucial factor in this approach, like in the pigeonhole algorithm, is k. The measure of complexity is determined by the size of k. You can see this observation in the graphs and tables above.

The merge sort's best, worst, and average cases are all the same, and O (n log n). It makes no difference whether the data is sorted, randomly or reverse sorted for this algorithm. Because when we merge sort, we split the array into two subarrays and continue the process. You can also see that the complexity has not changed in the plots.

Q2: Do the obtained results (running times of your algorithm implementations) match their theoretical asymptotic complexities?

To the theoretical results we are expected to insertion sort best case happens in the sorted data. On the other hand we are expect worst case and avg case in the reverse sorted data and random data. As you can see in the plots the expectations matched with my data.

We expect the complexity of the pigeonhole sort and count sort algorithms to be close to each other. As you can see in the plots the expectations matched with my data.

Except for sorted data, the join algorithm was supposed to have better complexity than the insertion sort and worse complexity than the pigeonhole and counting algorithm. But in my data, the complexity of the merge algorithm is worse than the complexity of the sorting algorithm. You can observe this situtation in the random data and reverse sorted data.