Hacettepe University
Computer Engineering Department

BBM204 Algorithms Lab - Spring 2021

Assignment 1 - Complexity Analysis on Sorting Algorithms

Deadline : 24/03/2021

Buğrahan YÜCEL - b21946754

1 Introduction

In this experiment, the goal was to understand the real relation between theoretical assumptions and empirical observations. The real question was, we know sorting algorithms have best, average and worst cases and these attributes define their quality but how are they relevant in non-ideal environments? How can we observe the effects of time-complexity on systems we build?

For the experiment, 5 of the sorting algorithms are inspected. To do that, the sorting algorithms were implemented, some input generation and testing system is constructed. After testing, the time datas are saved in csv formatted files. Using python libraries (Pandas, Matplotlib) different datas from multiple testings were averaged and plotted.

2 What is Complexity ?

Complexity is an attribute of algorithms that shows growth rate of resources (time, space) depending on the input. It is generally shown using Big-O notation. If an algorithm has O(n^2) complexity, and we plot it on an input-time graph, we expect a curve similar to $y = x^2$ function. Or we expect roughly a line for an O(n) algorithm, or a concave curve for O(log(n)) etc.

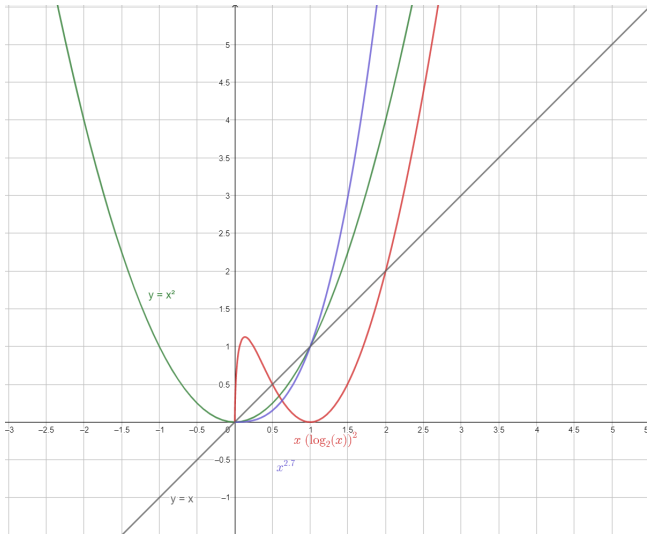

Figure 1: These curves represent the mathematical forms of complexity curves we will see

Time complexities of the sorting algorithms that are inspected in this experiment are given below.These results are obtained by a quick search on Wikipedia.

Case\Algorithm	Comb Sort	Gnome Sort	Shaker Sort	Stooge Sort	Bitonic Sort
Average Case	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^{2.7})$	$O(n\log^2 n)$
Best Case	$O(n\log n)$	$O(n)$	$O(n)$	$O(n^{2.7})$	$O(n\log^2 n)$
Worst Case	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^{2.7})$	$O(n\log^2 n)$

Figure 2: Known complexities of the algorithms

3 Experiment Datas

These datas are average runtimes of 10 test runs for average, best and worst cases. Best care input is a sorted array, worst case input is a reverse sorted array for all runs.

	combSort	gnomeSort	shakerSort	stoogeSort	bitonicSort
2	0.9	0.6	1.2	1.4	6.6
4	0.3	0.0	0.1	640.6	1.5
8	0.2	0.1	0.3	0.7	0.3
16	0.5	0.4	1.4	9.9	1.7
32	5.1	2.3	3.9	68.0	5.6
64	5.2	11.0	14.4	259.0	12.5
128	13.8	36.8	58.3	595.8	26.7
256	38.9	139.4	212.4	4990.4	51.5
512	61.4	424.6	644.7	40982.4	102.8
1024	130.7	1284.2	1577.7	159469.2	233.5
2048	302.3	4695.2	5107.0	1436316.6	498.2
4096	492.4	16804.3	21351.9	12898998.3	1126.4
8192	1057.7	73685.8	102402.7	116013712.4	2492.5

Figure 3: Average case (input size/microseconds)

We can observe that algorithm running times are proportional to the table above (Figure 2). (Only the comb sort's worst case weirdly runs better than its average case. This brings up the question that is a reverse sorted array really the worst case input for this algorithm? So i think presenting it as a worst case seems dubious and it should be taken notice of.)

	combSort	gnomeSort	shakerSort	stoogeSort	bitonicSort
2	0.5	5.6	0.9	0.4	0.9
4	0.0	1.9	0.1	0.0	1.1
8	0.0	1.3	0.0	0.0	0.0
16	0.0	1.5	0.0	3.6	0.9
32	0.0	0.3	0.2	24.7	4.0
64	0.4	0.8	0.7	197.4	4.4
128	1.4	1.1	1.5	355.8	24.4
256	2.8	2.5	2.1	3018.0	19.8
512	12.2	5.1	4.2	26412.1	45.8
1024	16.9	11.5	8.8	123024.0	103.4
2048	39.6	17.6	15.1	1112469.1	242.7
4096	88.5	70.1	43.6	9980211.0	529.9
8192	178.2	113.7	80.0	89827279.6	1143.8

Figure 4: Best case (input size/microseconds)

	combSort	gnomeSort	shakerSort	stoogeSort	bitonicSort
2	0.8	0.3	4.3	1.6	8.2
4	0.0	0.0	0.1	0.0	2.4
8	0.0	0.0	0.2	0.1	0.2
16	5.3	0.0	1.5	3.5	1.0
32	0.6	1.8	4.8	26.1	2.1
64	1.4	7.4	12.3	210.9	4.7
128	3.3	30.8	41.0	498.5	8.9
256	8.1	135.9	124.6	4306.7	23.1
512	25.2	481.8	454.0	38005.4	48.3
1024	39.4	1924.3	1780.2	147215.5	106.0
2048	85.4	7595.7	7121.0	1320799.2	232.9
4096	171.9	30463.8	28423.5	11795054.0	533.4
8192	355.8	119421.2	114954.5	106035590.9	1187.6

Figure 5: Worst case (input size/microseconds)

4 Plots of Datas

Below are the curves generated using data in the tables shown. For better understanding, the curves are added to the same graph one by one and plotted together. (See figures 6-14)

For the graphs below, we expect similar convex parabolic/hyperbolic curves for each algorithm. They might be hard to discriminate looking at the shape of them. But the sizes of the curves are expected to be much different especially for big input sizes.

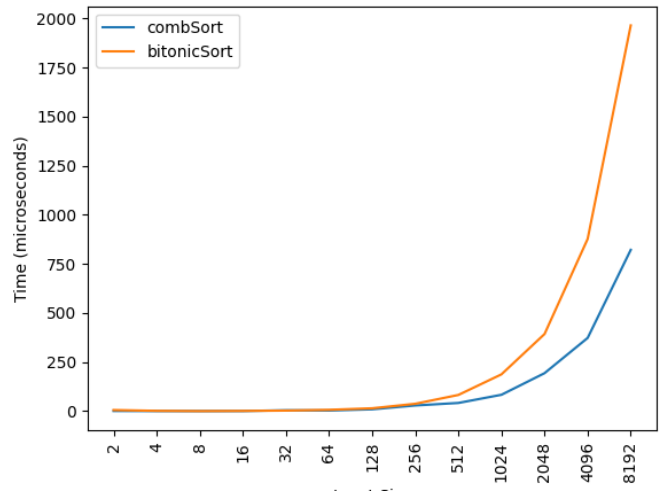

Figure 6: Average case for comb sort and bitonic sort

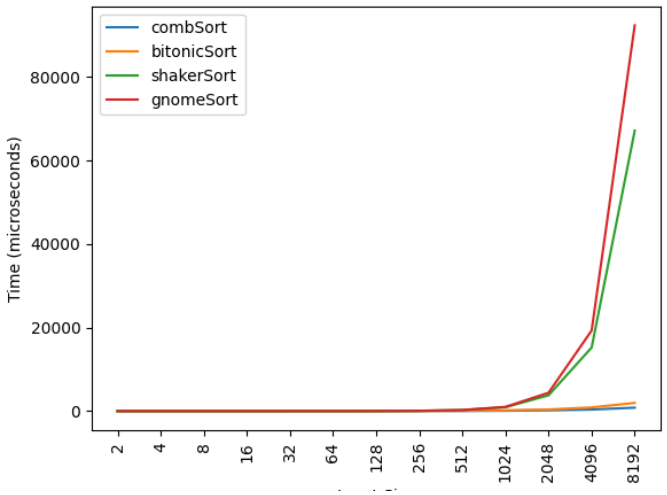

Figure 7: Average case for gnome sort and shaker sort. Notice that parabolic-looking curves look nearly flat compared to newly added ones. The thing that confused me here was, although comb sort, shaker sort and gnome sort having the same average complexity (n^2), bitonic sort was between them. ($nlog^2n$) That made me think that it may be caused by these parabolic functions having different sized coefficients. (like the difference of a^2 and $(5a)^2$)

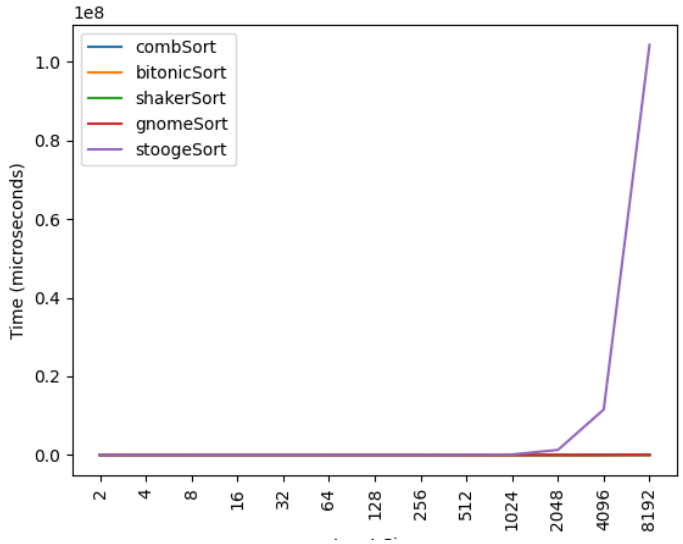

Figure 8: Average case for stooge sort. Its power $n^{2.7}$ is slightly bigger than n^2 but the difference is huge for big inputs

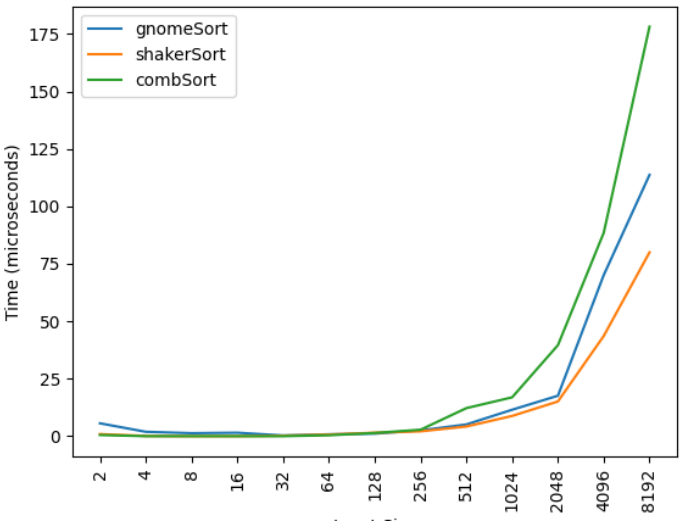

Figure 9: Best case for gnome sort, shaker sort and comb sort

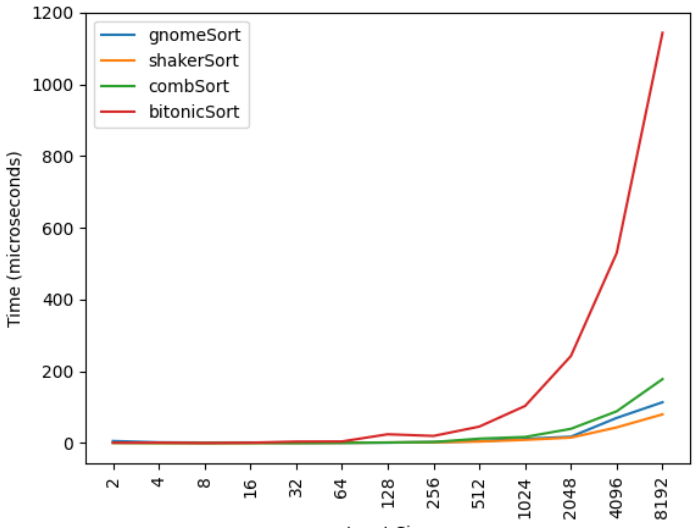

Figure 10: Best case for bitonic sort

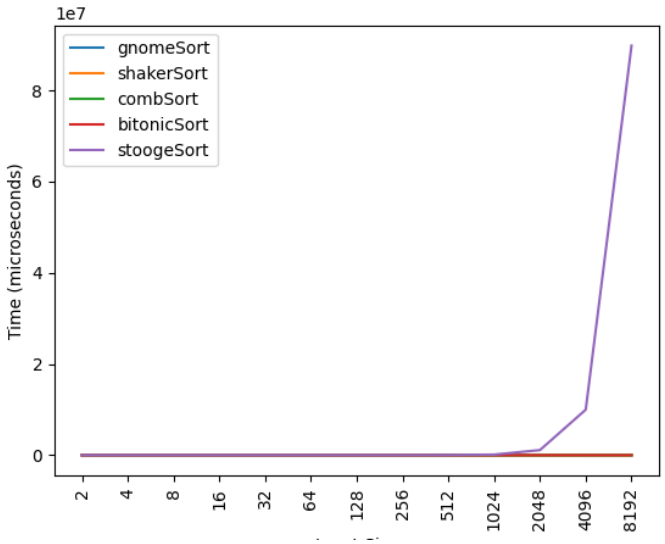

Figure 11: Best case for stooge sort

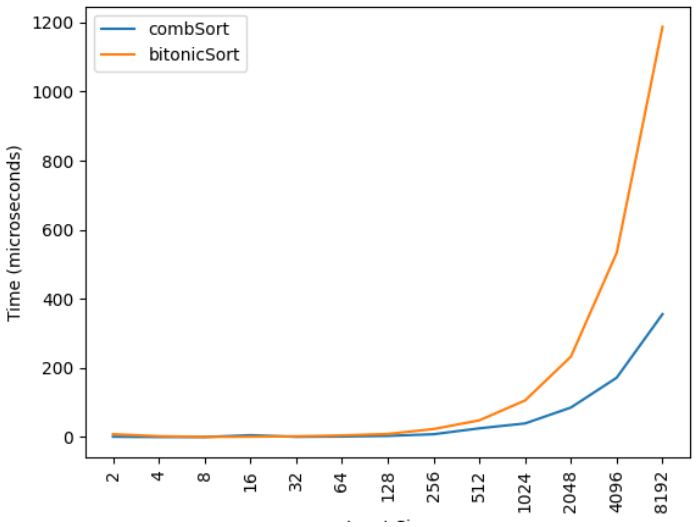

Figure 12: Worst case for bitonic sort and comb sort

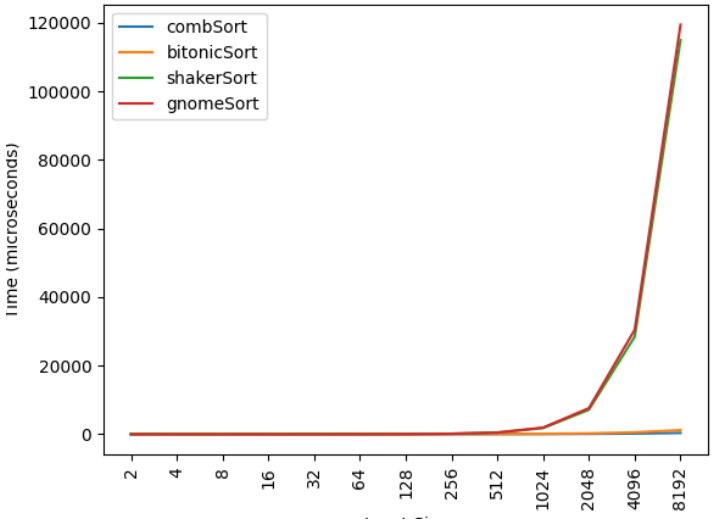

Figure 13: Worst case for shaker sort and gnome sort

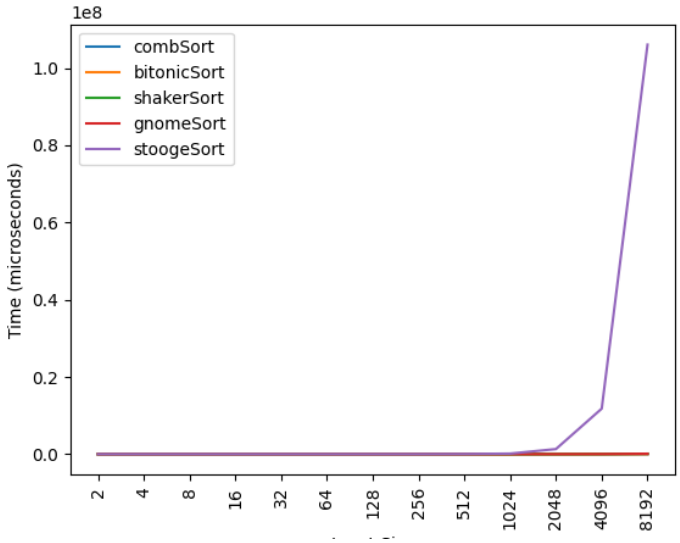

Figure 14: Worst case for stooge sort

5 Stability

Stability basically is whether the inputs with same keys preserve their relative order after the sorting or not. We can check manually if an algorithm is stable or not by looking at swap operations. If the algorithm is swapping non-adjacent elements, that means it is unstable. If it only swaps adjacent elements, then it is stable. Of course that has a reasoning. Suppose you have a sorting algorithm only swapping adjacent elements. When it faces two elements sharing same key, it won't swap them. That means there is no way that two elements with same key will ever change their relative order. On the other hand, if you swap far elements, there is no guarantee that you will save the relative order.

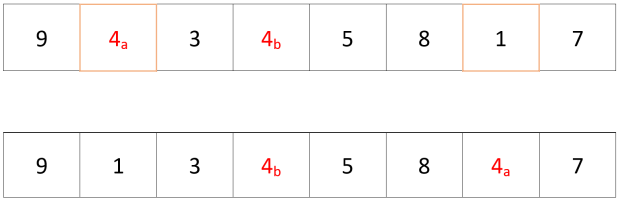

Figure 15: An operation like this breaks the relative order of 4a and 4b

To check stability, an ascending sequence of numbers (stability numbers) were given to each element when creating the input array before sorting. After the sorting, using a simple program, the elements with same keys are checked if their stability numbers remain their ascending order.

```
combSort Algorithm is Unstable
gnomeSort Algorithm is Stable
shakerSort Algorithm is Stable
stoogeSort Algorithm is Unstable
bitonicSort Algorithm is Unstable
```

Figure 16: Result of the stability check. It is consistent with the working principles. Gnome sort and shaker sort are stable because they always iterate one by one while swapping, comb sort is unstable because of the combing procedure (two far ends of the "comb" swapping), bitonic sort performs merging (see merge sort) and stooge sort swaps leftmost and rightmost elements