Sorting Algorithms Comparison

2019-12172 이재원

April 02, 2024

1. Introduction

In this report various sorting algorithms (Insertion sort, Merge sort, Quick sort (randomized pivot), Quick sort (last element pivot), Heap sort, Radix sort, Stooge sort) are compared by their time complexity in **Table 1**.

The time complexity is shown experimentally by calculating the average sorting time in **c++** code changing the seeds of randomly generated data from 0~4.

Sorting Algorithm	Average Case Complexity	Worst Case Complexity
Insertion Sort	$O(n^2)$	$O(n^2)$
Merge Sort	$O(n \log n)$	$O(n \log n)$
Quick Sort (Random Pivot)	$O(n \log n)$	$O(n^2)$
Quick Sort (Last Pivot)	$O(n \log n)$	$O(n^2)$
Heap Sort	$O(n \log n)$	$O(n \log n)$
Radix Sort	$O(nk)$	$O(nk)$
Stooge Sort	$O(n^{2.709})$	$O(n^{2.709})$

Table 1. Time complexity of sorting algorithms. The 'k' in Radix Sort represents the number of digits in the largest number in the list.

2. Methods

The sorting algorithm for the **c++** code was compile with g++ using optimization level *-Ofast*.

The **c++** code generates **json** file, and it is converted into **pandas DataFrame** in **python**. After using ***pandas.DataFrame.grouby().mean(),*** the data frame are as shown as **Figure 1**.

algorithm	n	seed	sort_time
heap sort	100	2.0	0.0000
	1000	2.0	0.0000
	10000	2.0	0.0000
	100000	2.0	0.0082
	1000000	2.0	0.1188
	10000000	2.0	2.1238
	100000000	2.0	36.1700
insertion sort	100	2.0	0.0000
	1000	2.0	0.0000
	10000	2.0	0.0090
	100000	2.0	0.9100
	1000000	2.0	95.7792

Figure 1. A part of result after using Pandas Dataframe grouby() and mean().

Then using ***pandas.DataFrame.unstack()***, the average sorting time is shown as **Table 2.**

This DataFrame is plotted by ***pandas.DataFrame.plot()*** in section **3. Results.**

algorithm \ n	100	1000	10000	100000	1000000	10000000	100000000
heap sort	0.0	0.0000	0.0000	0.0082	0.1188	2.1238	36.1700
insertion sort	0.0	0.0000	0.0090	0.9100	95.7792	NaN	NaN
merge sort	0.0	0.0000	0.0000	0.0090	0.1048	1.2190	14.4270
quick sort last	0.0	0.0000	0.0000	0.0050	0.0660	0.8432	9.2770
quick sort rand	0.0	0.0000	0.0000	0.0060	0.0744	0.8946	10.3666
radix sort	0.0	0.0000	0.0000	0.0030	0.0410	1.0396	9.9438
stooge sort	0.0	0.1208	97.7914	NaN	NaN	NaN	NaN

Table 2. The result of average sorting algorithms. The NaN means that the sorting time was not measured because it was over 180 seconds.

3. Results

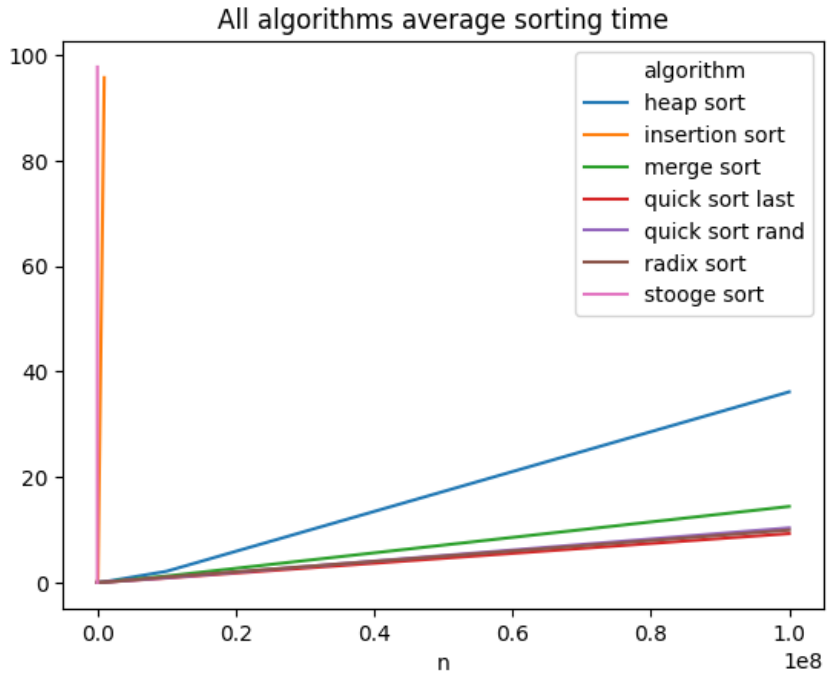

Figure 2. Average sorting time plot.

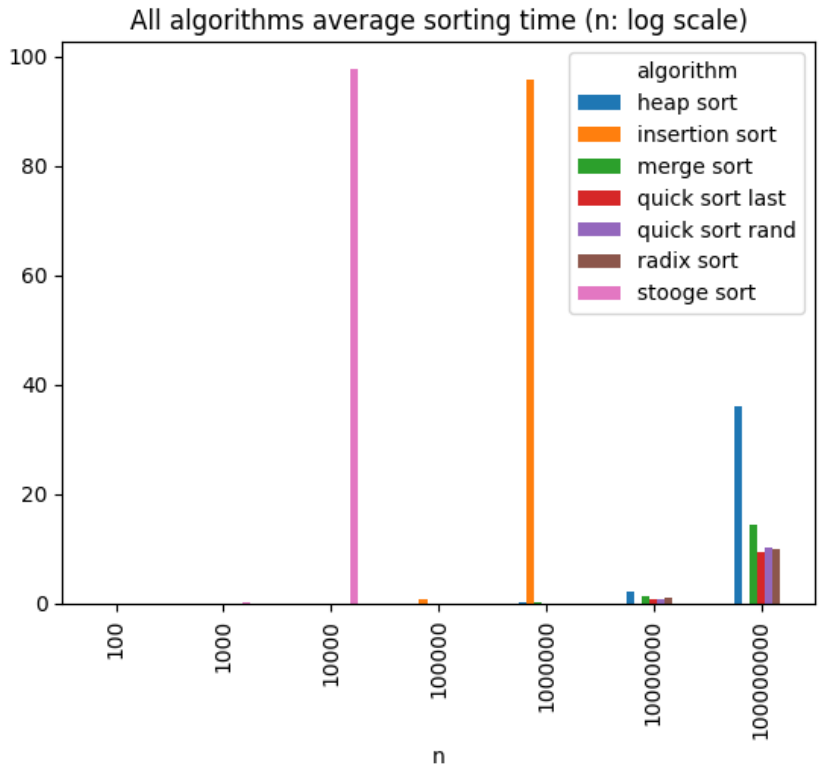

Figure 3. Average sorting time in bar plot, where axis x is log scale.

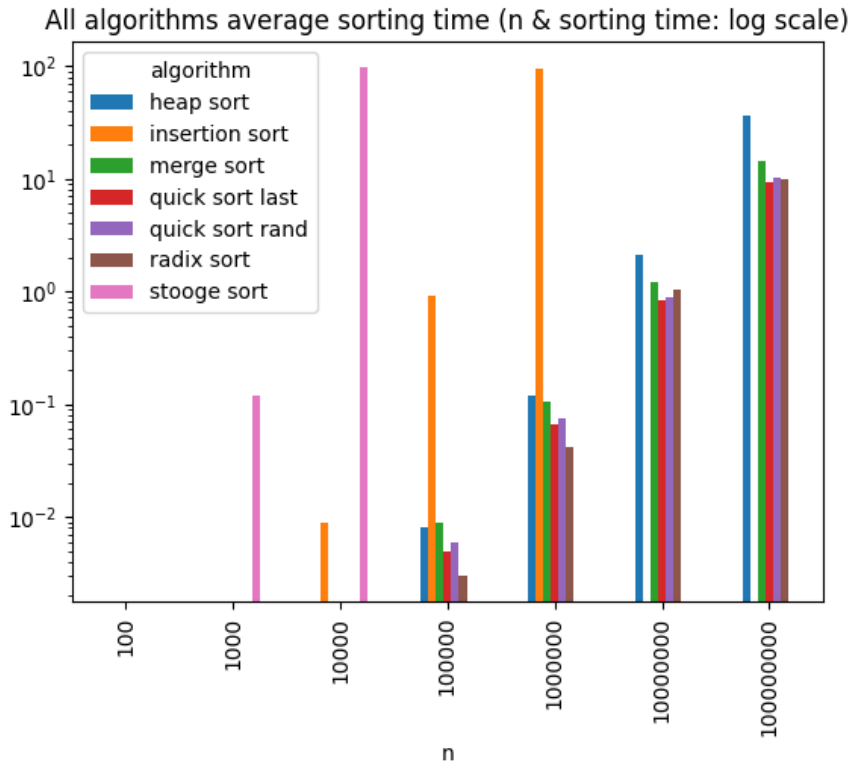

Figure 4. Average sorting time in bar plot, where axis x and y are both in log scale.

4. Discussion

In Figure 2, *insertion sort* and *stooge sort* have high time complexity compared with other sorting algorithms. In Figure 4, it can be seen that *stooge sort* has higher time complexity than *insertion sort*. This makes sense because *stooge sort* has time complexity of $O(n^{2.709})$, and *insertion sort* has time complexity of $O(n^2)$.

For other algorithms other than *insertion sort* and *stooge sort, heap sort* was the slowest sorting algorithm. However, *merge sort, quick sort, heap sort* all have same average time complexity. In Figure 2, it can be concluded that *heap sort* has a high constant factor compared to other $O(n \log n)$ algorithms.

Radix sort, which has time complexity of $O(nk)$ looks similar to $O(n \log n)$ algorithms in all results. This is because the 'k' in *Radix Sort* represents the number of digits in the largest number in the list. The data used in this experiment has range from 0 to *MAXINT_32*, which is 2147483647. Therefore, k is 10, and $\log n$ at $n = 10^8$ is not high enough compared to k = 10. And this makes it look the time complexity of $O(nk)$ and $O(n \log n)$ similar for $n \leq 10^8$.

5. Conclusion

In this report, various sorting algorithm times were measured and compared the average time. The results shows the time complexity of each algorithms.