Sorting Algorithms Comparison

Alp Bölükbaşı

August 13, 2022

Abstract

The main objective of the work is conducting an experiment by drawing a comparison between 4 sorting algorithms which comprehendsively studied and implemented throughout the CS201 course.

1 Introduction

Sorting Algorithms are responsible from rearranging a given array or list elements according to a via recognizing each element to compare into a specific order, so that you can reorganize large number of elements into a increasing or decreasing value order (i.e. shortest to longest distance or highest-to-lowest value.)

2 Experimental Setup

2.1 Sample Sizes

The sample sizes that used in this experiment are:

N = 1000,

N = 10000,

N = 100000.

2.2 Test Environment

Each result was generated and re-ran a few times manually. The observed results that are stated above are the average measurements. All results have measured in my laptop with Intel i7 7th Gen and licensed Windows 10. Used C++20 for each implementation and only core .cpp libraries are used (iostream and fstream, for taking large amount of inputs.)

2.3 Algorithms Used in the Experiment

The algorithms that used in this experiment are:

AlgorithmSortAll

o It does a simple insertion sorting operation to the given input. It's the slowest algorithm among all other algorithms. In worst case (time complexity), it works in $O(N^2)$.

AlgorithmSortK

o It also does an insertion sorting operation by kth input and compares kth with next inputs. In worst case, it works in $O(N^2)$.

AlgorithmSortHeap

o It sorts by a Binary minheap tree-based structure and does a sorting operation by referencing the minimum element on the binary tree. Repeats the same process for each remaining element. In worst case, it works in $O(NlogN)$.

AlgorithmSortQuick

o It picks an element as "pivot" (usually pivot is median) and does many partitions around the pivot. This is the fastest known generic sorting algorithm. In worst case, it works in $O(N^2)$ with a very low probability. Algorithm time is usually $O(N)$.

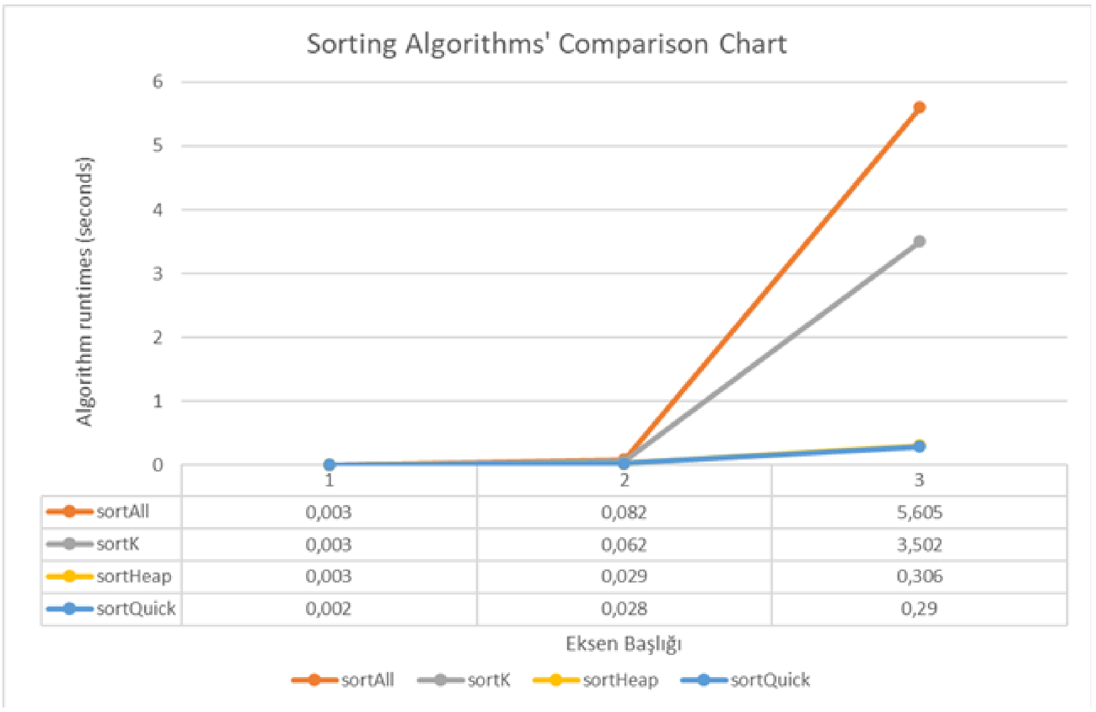
3 Results

Below, is the comparison of the each algorithms' runtime, demonstrated by the table.

Sample Size	sortAll	sortK	sortHeap	sortQuick
1000	0.015	0.014	0.013	0.005
10000	0.015	0.014	0.013	0.005
100000	6.059	3.876	0.407	0.368

Table 1: Sorting algorithm runtime comparison.

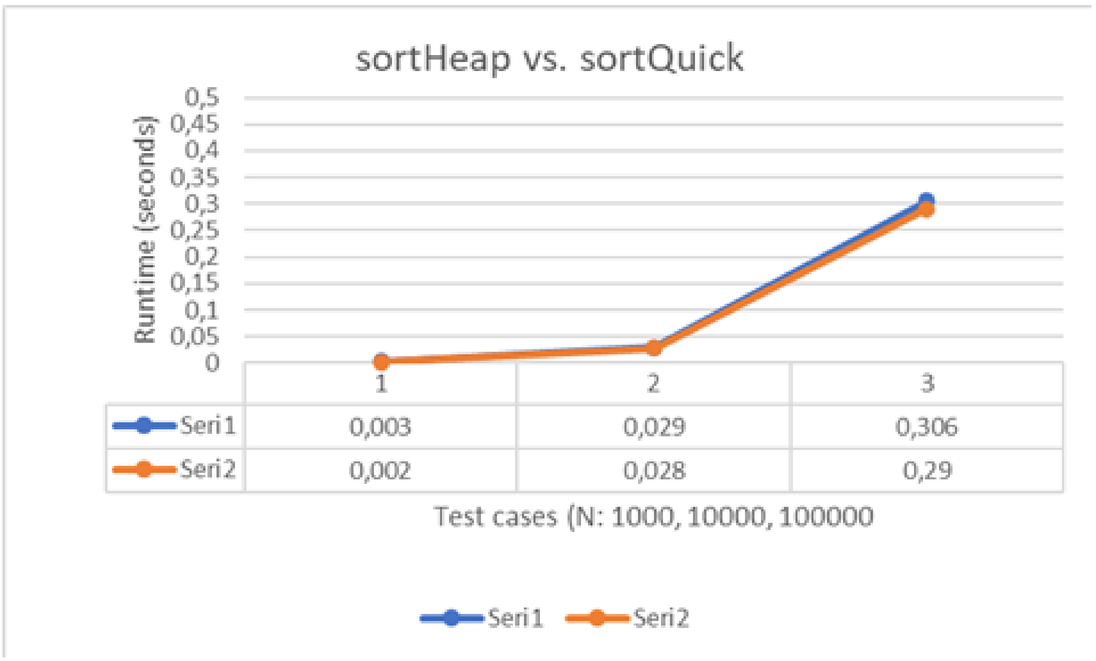
3.1 Charts



	1	2	3
sortAll	0,003	0,082	5,605
sortK	0,003	0,062	3,502
sortHeap	0,003	0,029	0,306
sortQuick	0,002	0,028	0,29

Figure 1: Comparison of each algorithms' runtimes.

It is crystal clear that, sortAll is way slower than sortK. sortHeap and sortQuick is way quicker than other two. To make a better observation, we must make a comparison in between sortQuick and sortHeap. (numbers on x-axis represents sample sizes: 1000, 10000, 100000)



	1	2	3
Seri1	0,003	0,029	0,306
Seri2	0,002	0,028	0,29

Figure 2: sortQuick is faster but their runtimes are close to each other.

4 Discussion

My interpretation before the experiment was almost correct and I was foreknown about the correct runtime order.

What I've mistaken about experiment was, even though these four algorithms' runtimes are different, I've encountered with almost same speeds when N ($= samplesize$) is small. I was surprised about that. As a result, we can observe and interpret that, insertion sort can be efficient in small sample sizes.

Even though both AlgorithmSortAll and AlgorithmSortK have same time complexity case rate: $O(N^2)$, AlgorithmSortK is way faster. Because AlgorithmSortK does insertion operations only around kth index. So that means, AlgorithmSortK does not encounter with unnecessary data and thus, not processes them.

When we look at the second chart, we can see that AlgorithmSortHeap and AlgorithmSortQuick have very similar runtimes. We must look at bigger N values in order to analyze the exact relation. Quicksorting is faster because it is more cache-efficient than Heapsort. In Quicksort, the inner loops have a smaller body. It almost doesn't do unnecessary element swaps (swaps are time consuming). But in Heapsort, even if your all data is ordered, you "absolutely" are going to swap elements in order to fill the array. Secondly, it's faster in terms of their O-Notations. Yes, Quicksort's worst case, $O(N^2)$ but it's a very low probability to come across with. In my implementation, the median element selected as "pivot" and thus, it works in $O(N)$. So, we did not encounter the worst case.

We can clearly say that there's a colossal difference between first two algorithms: SortAll, SortK and last two algorithms: SortHeap, SortQuick.