Comparison between insertion, merge, quick, and heap sort algorithms

Faisal Kassem

December 22, 2022

1. Introduction

In this paper, we investigate a classical problem: sorting. The problem is to arrange an array of n integers according to some total order which is computed in $O(1)$. We use $<$ as the total order.

In this paper, we compare four comparison-based sorting algorithms: insertion sort, quick sort, merge sort, and heap sort.

1.1. insertion sort

Insertion sort repeatedly inserts elements into a sorted sequence.

Inserting an element into a sorted sequence is done by moving all elements that are larger than the value being inserted to the index one larger than currently occupied, starting from the largest. The value to be inserted is moved to the array at the index after the first value that is smaller or equal to the inserted value.

First, the element located at index 0 forms the sorted part of the array. The algorithm then performs $n - 1$ insertions, starting from the second element in the array to the last element.

The average complexity of insertion sort is $O\left(n^2\right)$. The insertion sort has a best case: the array is sorted in ascending order. In such a case, the complexity of insertion sort is $O(n)$. The worst case of insertion sort is the array sorted in descending order. In such a case, the complexity of insertion sort is $O\left(n^2\right)$, the same as the average case.

The algorithm is stable and in place.

1.2. merge sort

Mergesort divides the unsorted list into n sub-lists, each containing one element (a list of one element is considered to be sorted). It repeatedly merges sub-lists to produce new sorted sub-lists until there is only one sub-list remaining.

Mergesort is a divide-and-conquer algorithm that was invented by John von Neumann in 1945. Most implementations produce a stable sort.

In sorting n objects, merge sort has the best, average, and worst-case performance of $O(n \log n)$

1.3. quick sort

Quicksort was developed by British computer scientist Tony Hoare in 1959. Quicksort is slightly faster than merge sort and heapsort for randomized data, particularly on larger distributions.

Quicksort is recursively partitioning the array and sorts according to the comparison with the selected value called "pivot". Quicksort is a divide-and-conquer algorithm, as it divides the array into sub-arrays and sorts them.

In Quicksort, "pivot" is selected first. Selection can be done in different ways, however, the simplest choice is to choose the first value in the array. Once selected, data is partitioned into two sub-arrays, whether less than or greater than the pivot. Then, sub-arrays are sorted recursively.

Partitioning of an n element array is $O(n)$. The best case of quicksort (it is when pivot "pivot" should divide the array in half in all recursive quicksort calls) is $O(n \log n)$. According to the Mathematical analysis, the average case of quicksort turns out to be $O(n \log n)$ as well.

Efficient implementations of Quicksort are not stable.

1.4. heap sort

Heapsort divides its input into sorted and unsorted parts, and then iteratively reduces the unsorted region by extracting the unsorted region and placing it into a sorted region. To implement heapsort, a heap structure is required, and in spite of many ways of heap implementation, an array can be used, and the elements in the array form a binary-tree-like structure.

To construct a heap, one of the ways is to start from the last parent element of the array (the last element which according to the binary-tree structure could have children), and fix the tree from down to up. Reversely, the second way is the continuous insertion of the elements and fixing the order of the elements to fit in a tree-like structure.

Heapsort is often referred to as improved selection sort, Unlike selection sort, heapsort doesn't make a linear-time scan, and rather maintains the unsorted region, which is much faster in terms of searching for the largest element in each step.

Heapsort was invented by J.W.J. Williams in 1964, Robert W. Floyd published an Improved version in the same year, that could sort an array in place.

Heapsort is an in-place treesort algorithm, but it is not a stable sort. Worst, Best, and Average cases of performance, according to the Mathematical analysis are all $O(n \log n)$

2. methodology

2.1. Description

The algorithms were implemented in $\mathrm{C}++$. The results were generated for Randomly shuffled arrays of values $0, 1, \ldots, n-1$.

The algorithms were tested on arrays of sizes from 100, 200, 300, $\ldots, 10000$. In both categories, each algorithm was given the same input data. Each array size was tested 100 times.

The result for each size is the average time it took for each algorithm to sort the input array.

3. Results

Graphs of results for the average case of all sorting algorithms are presented in Figures 1 and 2. Obviously represented, Insertion Sort compared to other sorting algorithms has performed incredibly worse and took approximately more than 50 times longer time than the average performance of all other sorting algorithms, which is not surprising since insertion sort takes quadratic time.

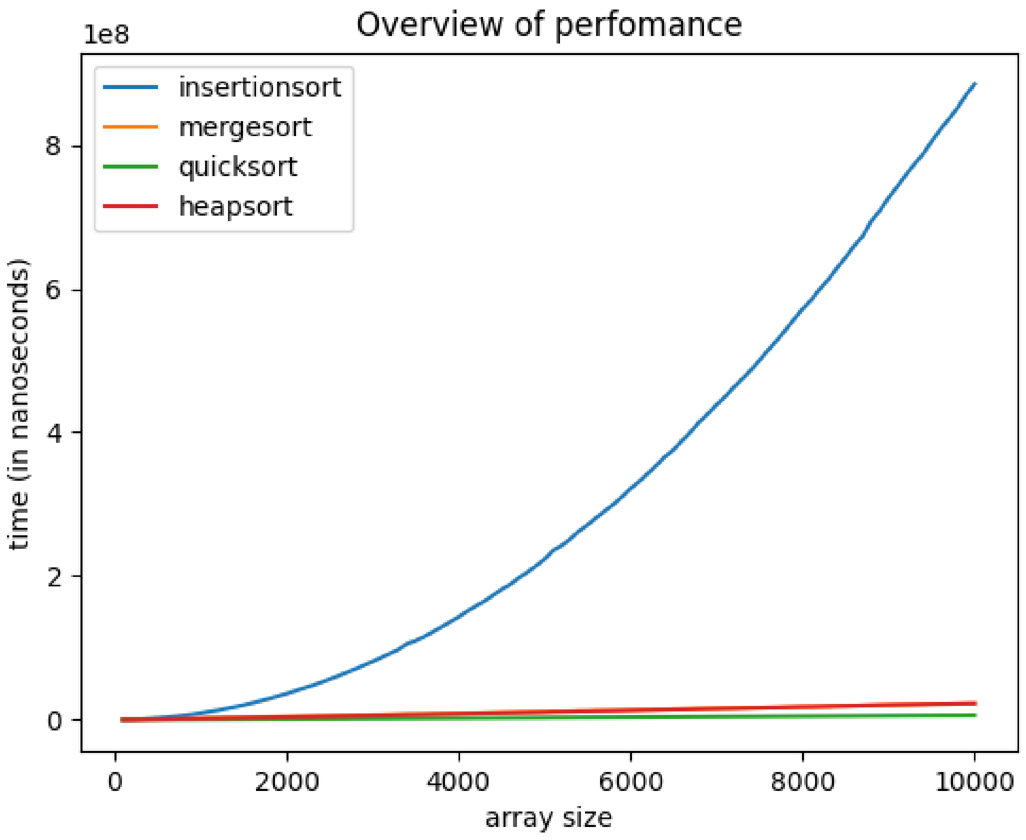

Figure 1. Average time each algorithm took to sort an array of randomly shuffled values from 0 to $n - 1$.

For small array sizes illustrated in Figure 2, the results are the same.

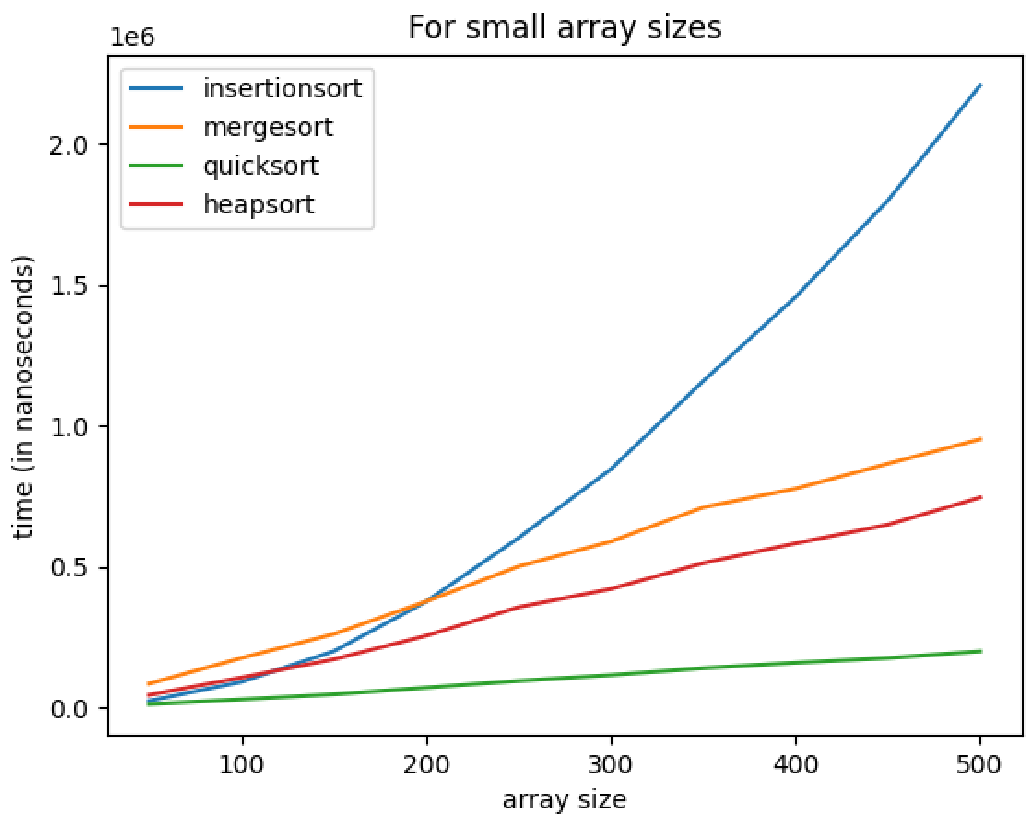

Figure 2. Average time each algorithm took to sort an array of randomly shuffled values from 0 to $n - 1$.

However, in Figure 2, Quicksort has the fastest level of performance, and heapsort rose slightly lower in terms of time when the array size is increased.

To have a better look, Figure 3 demonstrates the performance of all algorithms without insertion sort for array sizes up to 10000.

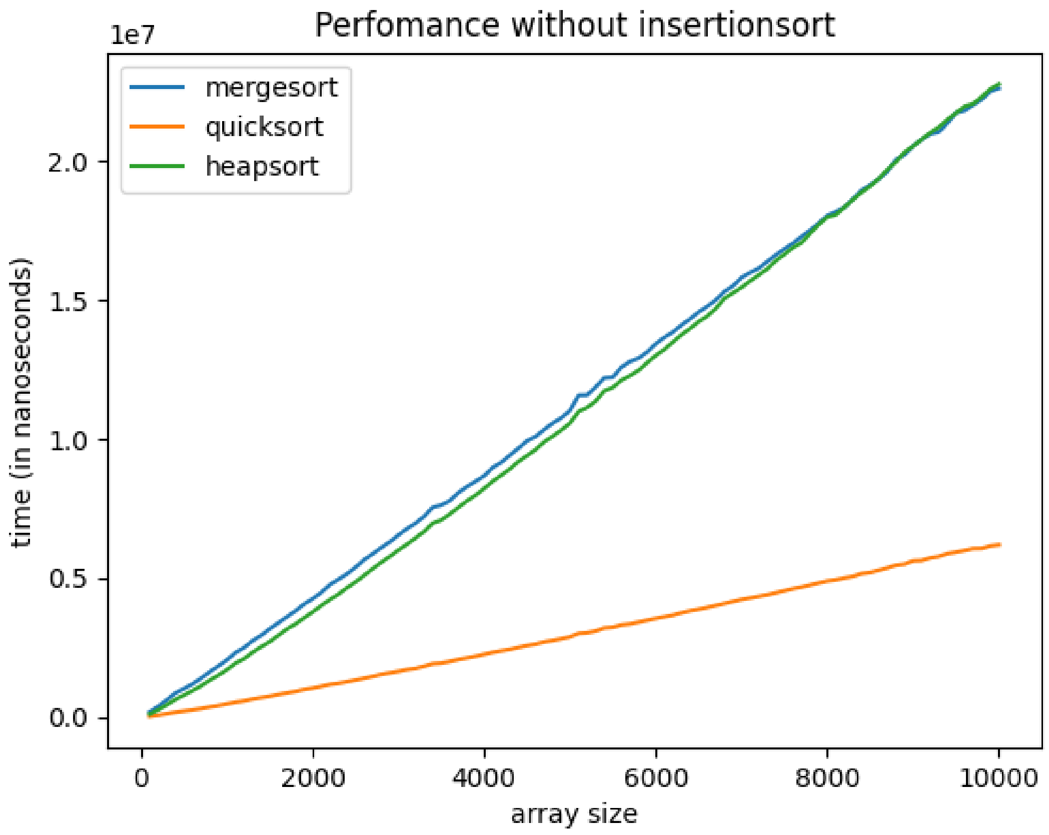

Figure 3. Average time each algorithm took to sort an array of randomly shuffled values from 0 to $n - 1$, without insertion sort taken into consideration

Clearly, according to the view of Figure 3, quicksort is faster about 3-4 times than merge and heapsort. Additionally, the important thing to note, from 0 to 8000 array size heapsort time performance is slightly lower compared to merge sort, however, from 8000 to 10000 the difference is imperceptibly distinguishable

A better observation for the above-mentioned case is shown in Figure 4.

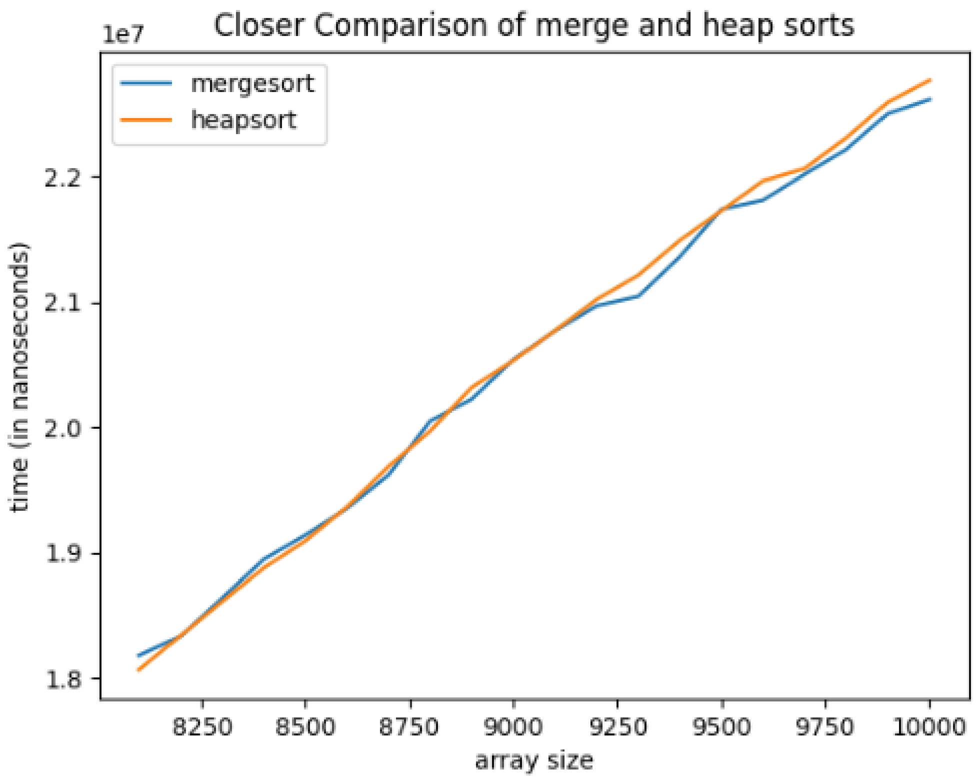

Figure 4. Average time merge and heap sort took to sort an array of randomly shuffled values from 0 to $n-1$.

4. Conclusions

Quick sort performed best in the average case especially on large size arrays, about 3-4 times faster than heapsort and mergesort. On the other hand, Heap sort performed worse than the Merge sort, but the difference is imperceptebly small, and reaching larger array sizes speed is almost differentiable, also those algorithms do not require optimization.

Hence, we conclude that quicksort is best of the considered sort in terms of speed of sorting, whereas the rest sorts may be cosnidered as optimal alternative solutions.

> It's like finding a needle in a huge pile of needles that lays in a massive haystack. The difference between theory and practice is that in theory there is no difference...