Speed comparison between insertion, heap, merge and quick sort

Vladyslav Pallah

December 17, 2022

# Introduction

In this article, we analyze the speed and compare four sorting algorithms: insertion sort, heap sort, merge sort and quick sort. The way we will compare sorting methods is to track the time it takes for each sorting method to sort vectors whose size is from 100 to 10,000 numbers (from 0 to 10,000) randomly filled.

## Insertion sort (in-place)

Insertion sort is stable that 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 the moved to the array at 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*(*n*2). The insertion sort has a best case: the array sorted in an 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*(*n*2), same as average case.

The algorithm is stable and in-place.

## Heap sort (in-place)

Heap is not stable sort algorithm that can be seen as an improved selection sort, and consists of two parts.

First is the creation of a heap:

A heap is a correctly sorted array, in such a way that the largest number is on the first place (also called a parent) of the array the next 2 child elements which are smaller than a parent, then 2 child elements became parents and other numbers should be smaller than these parents and this order until the end.

And the second part is sorting the elements by value:

The implementation is that we take the largest element and insert it at the end of the array, and then we will heap again our array but the size will be decreased by 1 and continue this order till the size equal zero.

After these two steps, we have already sorted an array.

Generally slower than other O(n.log(n)) sorting algorithms such as Quick sort, Merge sort.

## Merge sort (out-of-place)

Merge sort is an effective stable sorting algorithm that provides stable sorting.

Merge sort — divides a large array into two smaller subarrays, and then recursively sorts the subarrays. Basically, the whole process includes two stages:

Divide the unsorted array into n subarrays, each of size 1 (an array of size 1 is considered sorted).

Repeatedly combine the subarrays to create new sorted subarrays until there is only 1 subarray left, which will be our sorted array.

The worst time complexity of merge sort is O(n.log(n)), where n is the input size.

## Quick sort (in-place)

Quicksort is an efficient not stable sorting algorithm that usually works about two to three times faster than Merge Sort with a good implementation.

Quick sort is an algorithm that first divides a large array into two smaller subarrays and then recursively sorts the subarrays.

Basically, the whole sorting process includes three stages:

Pivot selection: Select an element called pivot from the array ( usually it is the rightmost element of the section).

Separation: Reorder the array so that all elements with values less than the pivot are placed before the pivot element and all elements with values greater or equal than the pivot go after it. After this breakdown, the pivot element takes its final position.

Repeat: Recursively apply the steps described above to the subarray of elements with smaller values than the pivot's, and separately to the subarray of elements with larger values than the pivot's.

Quicksort on average gives O(n.log(n)) comparisons for sorting n Items. In the worst case, O(n2) comparisons are obtained, although this behavior is very rare.

## Hybrid sort (in-place and out-of-place)

Hybrid sort is not stable sorting algorithm which is made with the help of conclusions from graphs and contains 2 different modes of sorting on different sizes of vectors.

The bas the basis of the hybrid method is the quick sort, but when the segment of the vector is less than 100, we use the insert sort, so in this interval it is the fastest.

So we get the fastest method.

# Methodology

Algorithms were written using C++ and were tested on arrays of sizes 1 to 100 with step 5, and 100 to 10 000 with step 100, which are filled with numbers from 100 to 10,000 randomly.

Each vector was sorted 200 times and the result is an average time.

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

# Results

Graphs of results for the random-filled array case of all sorting algorithms are presented in Figures 1 and 2.

The slowest one is insert sort which takes multiple times more than others.

Merge sort in 3 times faster than insert sort however in 20 times slower than heap sort.

Heap sort and quick sort take the least time, but heap sort is 3 times slower than quick.

For small array sizes shown in Figure 2, the results are the same. However, the merge sort is a little bit slower than insert sort in a range from 0 to 3400 numbers.

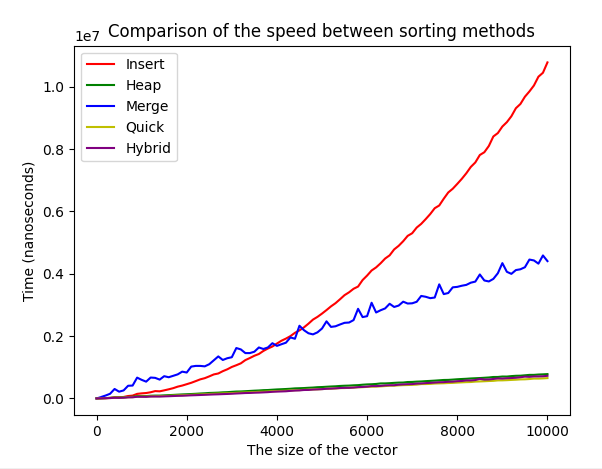


Figure 1: Average time each algorithm took to sort an array of randomly shuffled values from 0 to *n*− 1.

# Conclusions

The slowest sort method is insert sort.

The third place in terms of speed is the merge sort.

The second place is the heap sort.

According to the graphs, we have noticed that the quick sort is the fastest method throughout the testing period.

And the hybrid sort most effective.

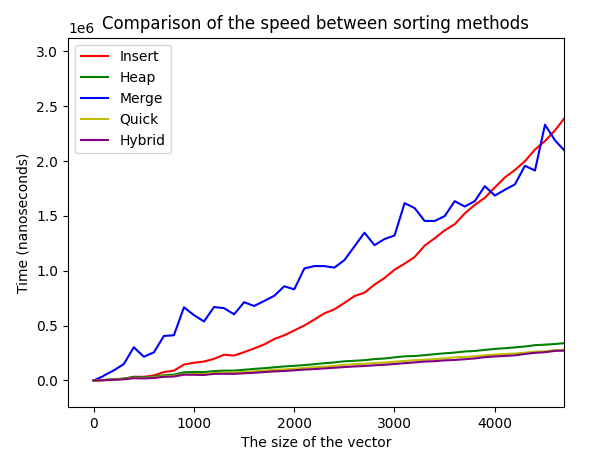


Figure 2: The graph shows results for vector with the size range (0 - 4000).

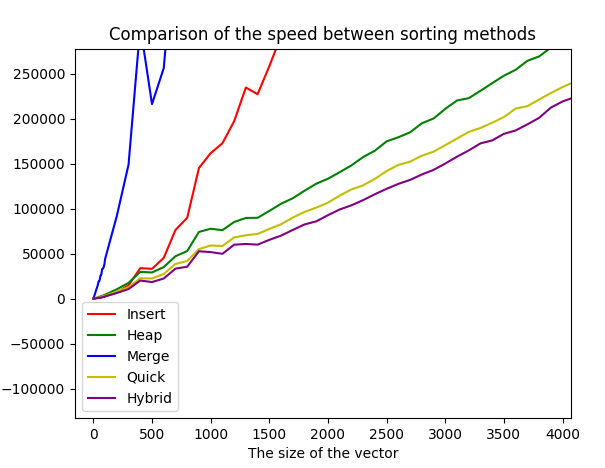
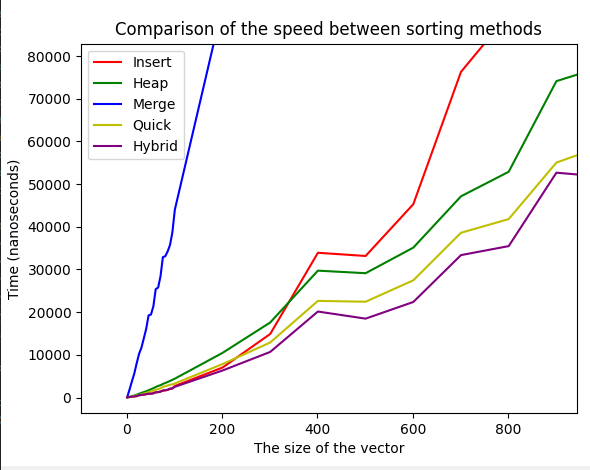


Figure 3: The graph shows results of Heap, Quick and Hybrid sort for vector with the size range (0 - 4000).

Figure 4: The graph shows results for vector with size range (0 - 800).

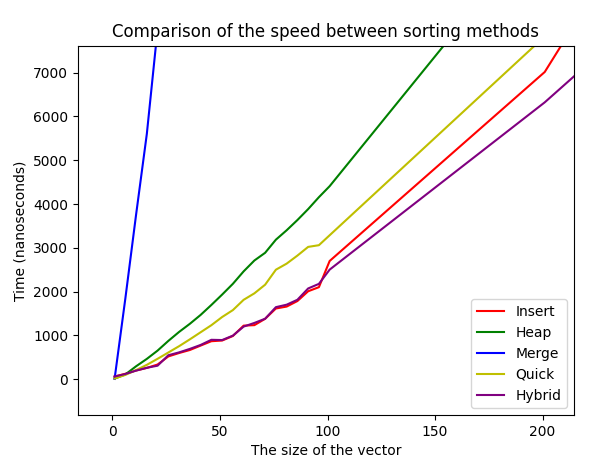


Figure 5: The graph shows results for vector with size range (0 - 200).