Sorting: Comparison between merge, heap and quick sort

Nijat Sadikhov

January 28, 2021

1 Introduction

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

1.1 Merge Sort

The Algorithm for Merge sorting is about continuously dividing a given array into 2 almost equal parts until the size of the divided array becomes 1. Afterwards, sorting all parts (left and right) and merging them altogether which basically means ordering merged array by selecting smaller values when comparing between left and right arrays.

The average complexity of merge sort is O(n * log(n)). It has no best or worst cases. And, in comparison with selection, insertion and bubble sorts, Merge sort might be considered faster. But it also requires additional memory space.

The algorithm is stable and not-in-place.

1.2 Heap Sort

This sorting method involves firstly building a heap data structure from a provided array. To create a heap, we should consider index 0 as our starting point and then insert values of following indexes. Simultaneously, checking a condition to sort between parent value and children values which is that parent value should be greater or equal to any of its children. After having a heap-based data structure, we are expected to swap the first element with the last one and reduce the size of an array. Of course, following the heap data structure rule, at the same time.

In all cases, the complexity of heap sort is O(n * log(n)).

The algorithm is non-stable and in-place.

1.3 Quick Sort

Quick sort uses partitioning to split he array into two parts: smaller and greater, with a pivot which is an element from array and can be optionally chosen. But in most cases, it is the first or the last element is being used. For each divided part, 2 elements must be found starting from leftmost and rightmost sides and following a rule of having left side element greater than the chosen pivot and the right one less than it. Furthermore, after those elements are found, they must be swapped in order to organize a properly sorted array.

The best and average complexity of merge sort is O(n * log(n)). The worst case occurs when the pivot is badly chosen and it performs in O(n^2) time.

The algorithm is non-stable and in-place.

2 Methodology

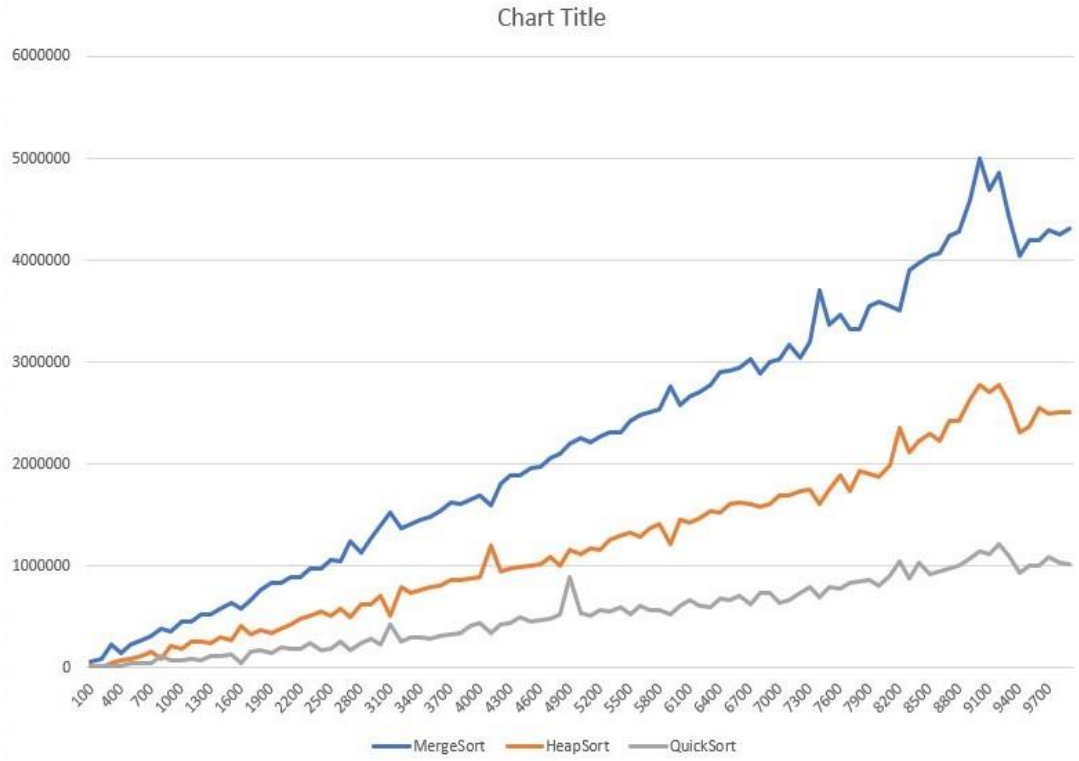
The algorithms were implemented in C++. The results were generated for two categories: randomly shuffled arrays of values 0, 1, . . . , n – 1 and sorted array.

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

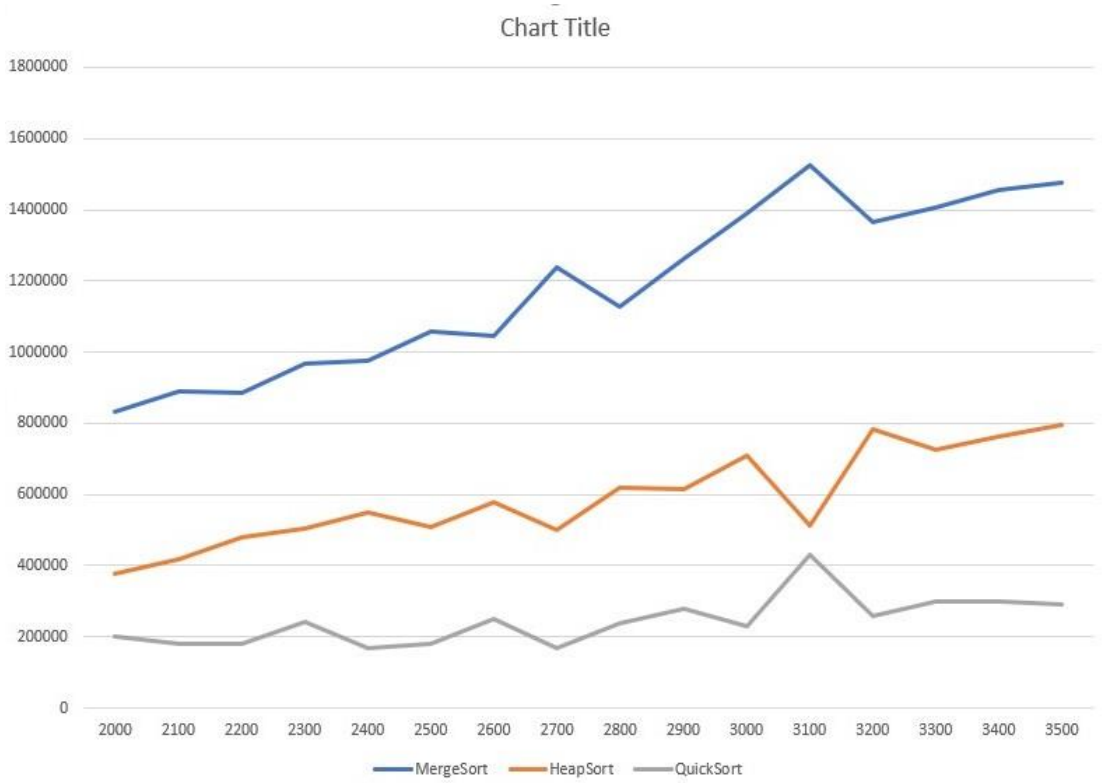
The result for each size is the average time it took for each algorithm to sort the input array.

3 Results

In average case, Merge sort takes more time than other sort algorithms. Heap sort takes half time required for merge sorting. The time in nanoseconds took for quick sort algorithm is also approximately as half time as heap’s.



Randomly Shuffled Array

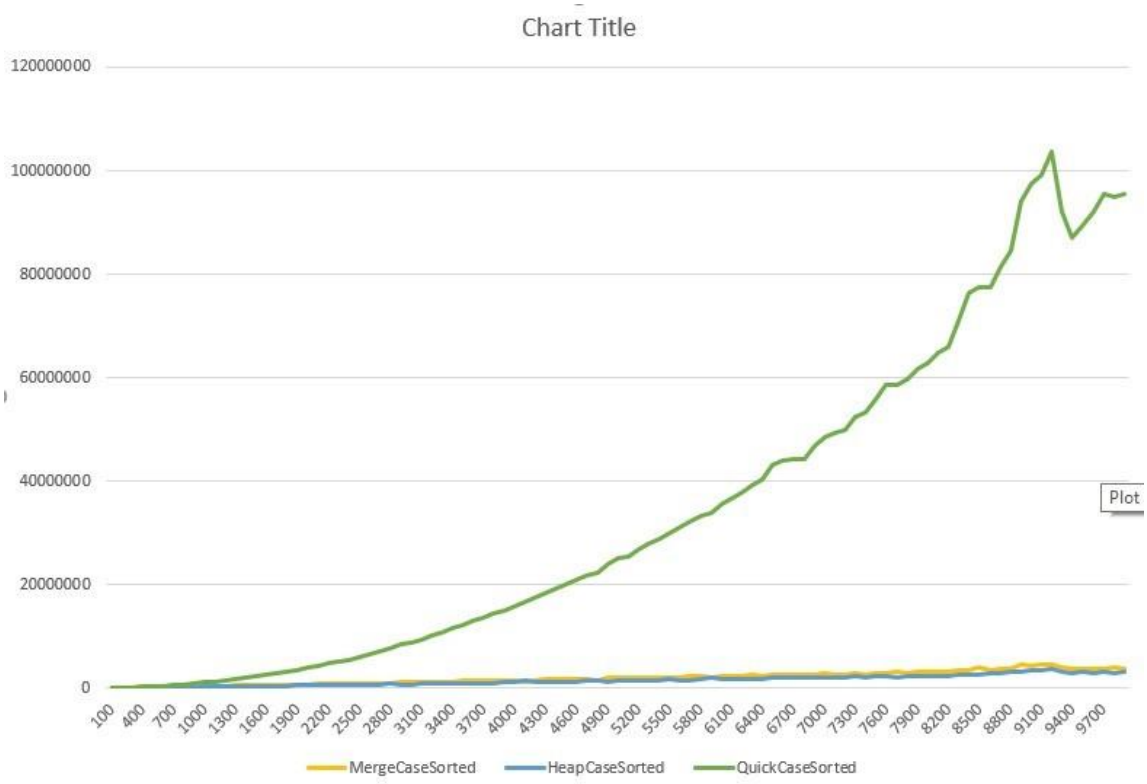


Randomly Shuffled for Small Array sizes

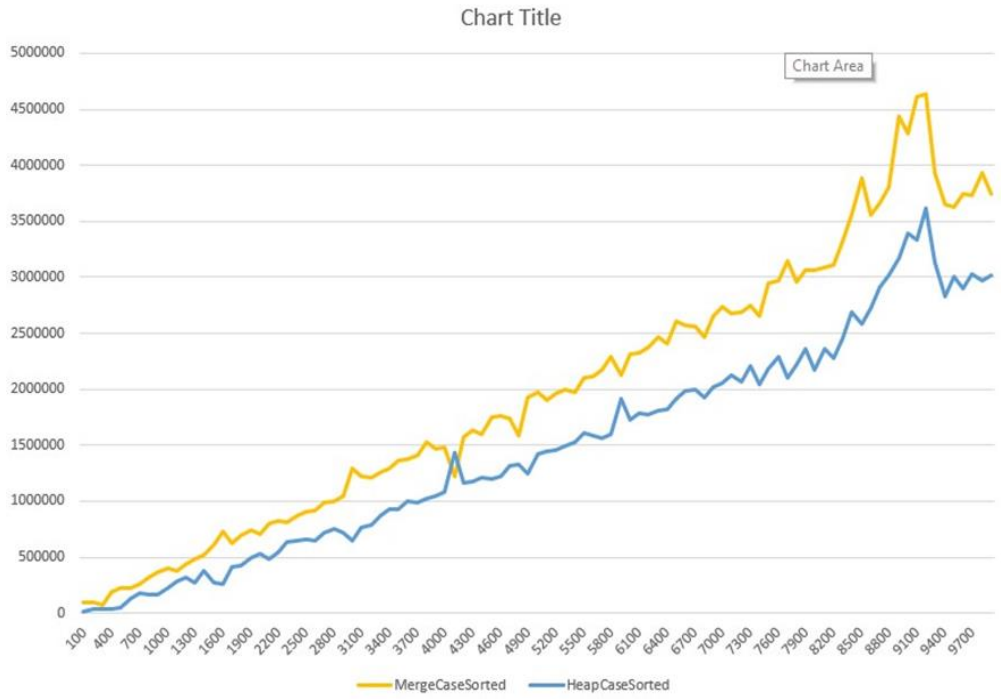
4 Conclusions

Quick sort performed best in the average case. When it comes to Heap sort, it performed worse than Quick sort only by a factor of about 2 (50 %). Lastly, Merge sort was the worst performed among others, around 50 % more time than Heap sort.

We can conclude that Quick sort is best of the three sorts in average case but in sorted array case, it is way much worse than others.



Average time each algorithm took to sort an array of sorted values



Average time each algorithm took to sort an array of sorted values, excluding quick sort.