Comparison between insertion, quick, heap and merge sort and creation of hybrid sort based on research

Akhliddin Akhmadjonov

December 17, 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 $O(1)$. Our total order is going to be an ascending order. In this paper we compare four comparison-based sorting algorithms: insertion sort, quicksort, heapsort and mergesort. After that, based on the comparison, we write a hybrid sorting algorithm which combines the best parts of the all researched algorithms.

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 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.

1.2 Quick sort

Quicksort is a divide and conquer algorithm. It works by selecting a 'pivot' element from the array and partitioning the other elements into two sub-arrays, according to whether they are less than or greater than the pivot. The sub-arrays are then sorted recursively. This can be done in-place, requiring small additional amounts of memory to perform the sorting. When comes to complexity, the average case of quicksort is same as its best case $O(n \log n)$. In the worst case, it makes $O(n^2)$ comparisons. The algorithm is not stable but it is in-place.

1.3 Heap sort

Heapsort is a sorting algorithm that in its central idea is very similar to selection sort. Like selection sort, heapsort divides its input into a sorted and an unsorted region, and it iteratively shrinks the unsorted region by extracting the largest element from it and inserting it into the sorted region. Unlike selection sort, heapsort does not waste time with a linear-time scan of the unsorted region,. Rather heap sort maintains the unsorted region in a heap data structure to more quickly find the largest element in each step. Heapsort is an in-place algorithm, but it is not a stable sort. Average performance of this algorithm is $O(n \log n)$.

1.4 Merge sort

The Merge Sort is also one of efficient sorting algorithm that is based on the Divide and Conquer method. The divide-and-conquer algorithm breaks down a big problem into smaller, more manageable pieces that look similar to the initial problem. It then solves these sub-problems recursively and puts their solutions together to solve the original problem. $O(n \log n)$ is average performance of merge sort algorithm. Unlike heap sort, mergesort is stable but it is not in-place.

1.5 Hybrid sort

Hybrid sorting algorithm is combination of quicksort and insertion sort. Quicksort algorithm is efficient if the size of the input is very large. On the other hand, insertion sort is more efficient than quick sort in case of small arrays as the number of comparisons and swaps are less compared to quicksort. Therefore, in order to sort efficiently we create hybrid algorithm using both approaches.

2 Methodology

The algorithms were implemented in C++. The results were generated for randomly shuffled arrays of values 0, 1, . . . , n − 1. The algorithms were tested on arrays of sizes from 100, 200, 300, . . . , 10000. During research, 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

Figure 1 illustrates average case of four sorting algorithms, namely quicksort, mergesort, heapsort and insertion regarding to array size from 0 to 10000. It can be clearly seen that, insertion sort takes several times longer than other sorting algorithms while quicksort becomes most efficient method among others as input size grows.

Moving onto smaller arrays, Chart 2 presents the fact that, insertion sort gains the lead as the most productive sorting technique but as expected looses its dominance to quicksort at certain point that can be observed in Graph 3. In all three figures, heapsort and mergesort performed medium or worst time complexity in comparison with other two.

By considering mentioned information, we implemented hybrid sort that utilize advantages of quicksort and insertion sort with regard to input size for achieving best performance as possible. Figure 4 and 5 displays that hybrid sort is most efficient in all scenarios than other sorting algorithms.

4. Conclusion

Heapsort and mergesort remained as medium in respect of time efficiency. Insertion sort performed best on small arrays but in average case it performed several times worse than others. Quicksort was more productive sorting method in overall case, by loosing its lead to insertion sort only for smaller input data.

Implementation of hybrid sort introduce true time efficient sorting algorithm which performed best among all type of scenarios including big and small sized arrays. Therefore, we conclude that hybrid sort is the best of five sorts.

Figure 1. Array from 0 to 1000

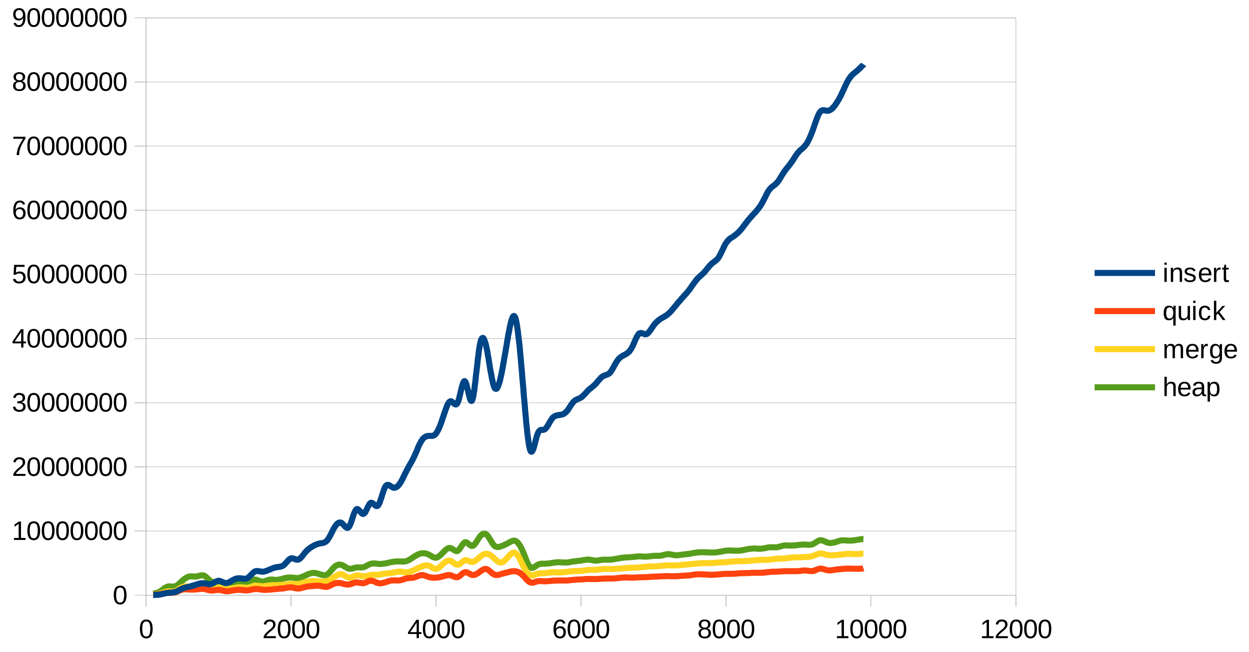


Figure 2. Small array from size to 100

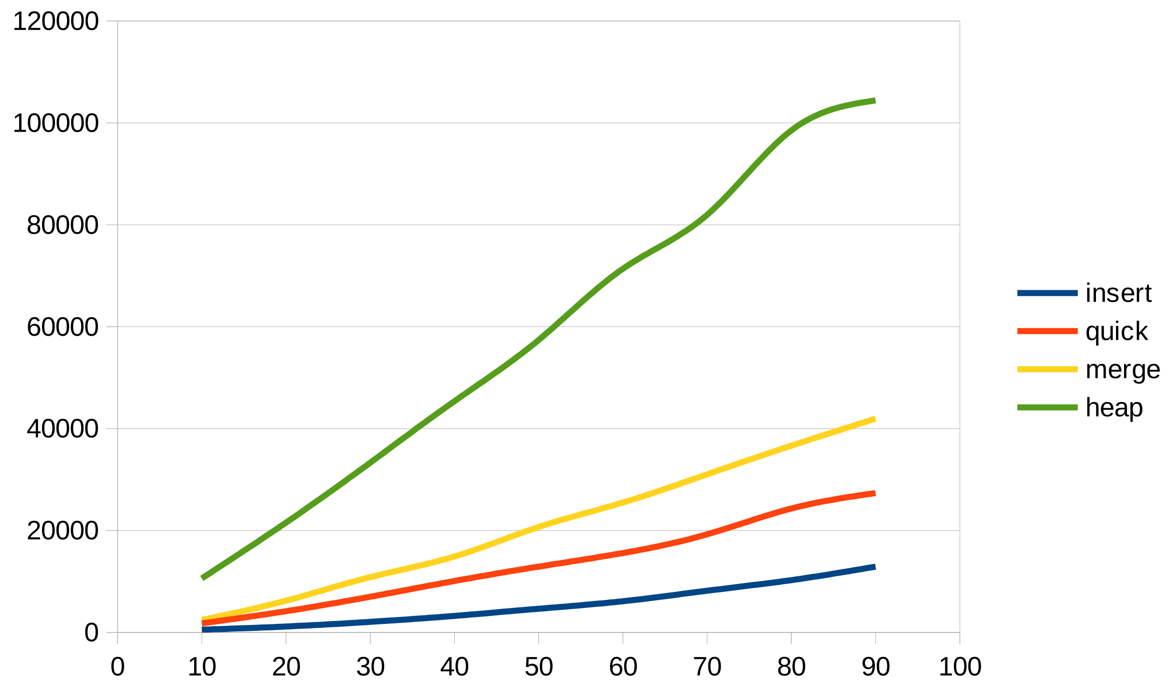


Figure 3. Shows switching lead in insertion and quicksort around 250-300

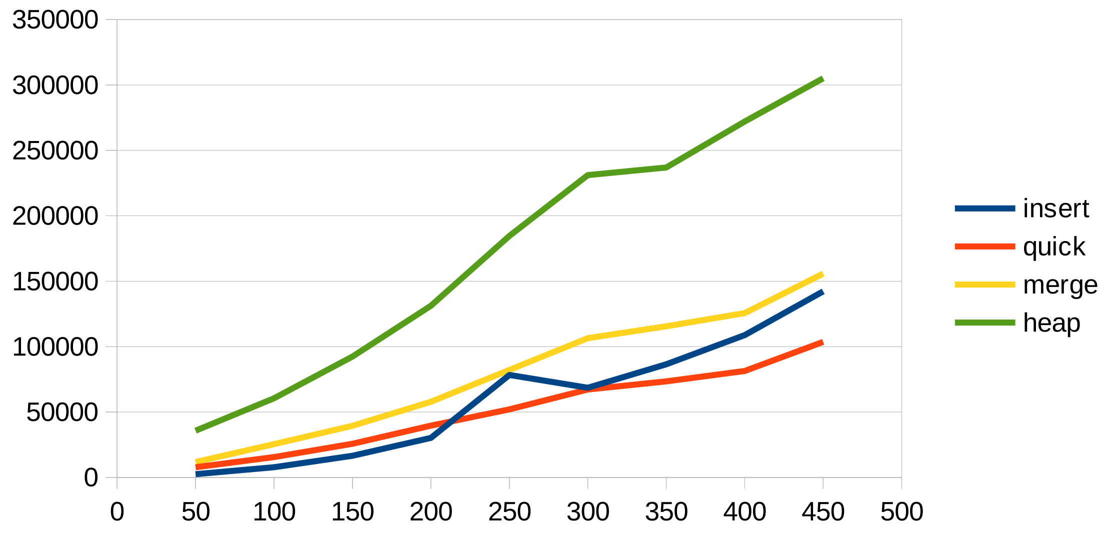


Figure 4. Hybrid sort efficiency in small arrays

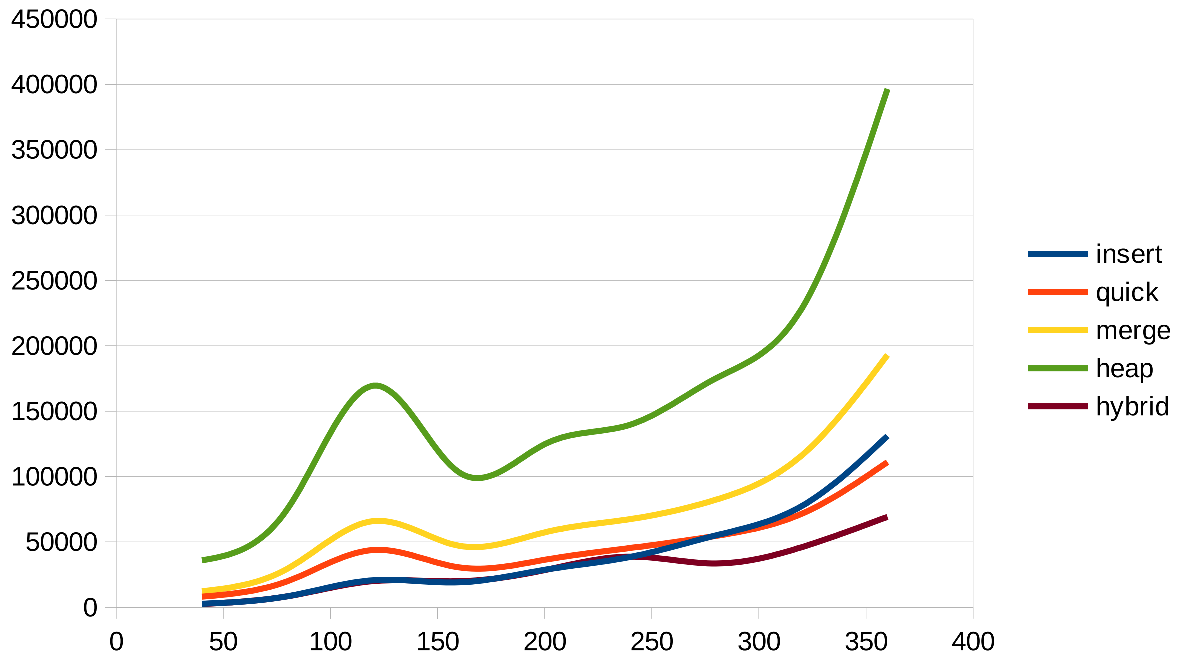
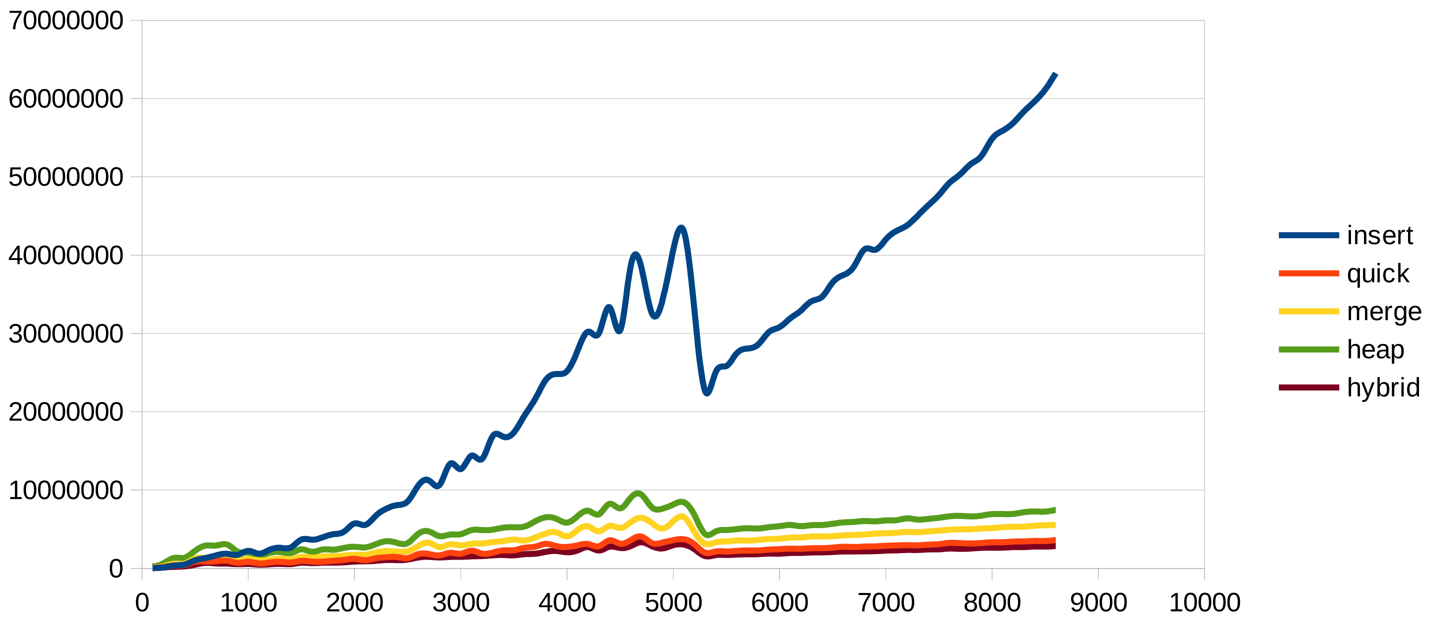


Figure 5. Hybrid sort efficiency in overall performance


insert
quick
merge
heap
hybrid