Sorting algorithms: comparing insertionsort, quicksort, merge sort and heap sort

Yakub Yakubov
December, 2022

1) Introduction

In this project, I will do the research on sorting and will try to find out the fastest algorithm among
insertion sort, merge sort, quciksort and heap sort. The problem is to create an array or possibly vector
of n integers which should be sorted by each and every algorithm of the given.

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

Quick sort is a sorting algorithm based on the divide and conquer approach. An array is divided into
sub-arrays by selecting a pivot element. The pivot element should be placed in such a way that all elements less than a pivot are on the left side and elements greater than pivot are on the right side of the pivot. The left and right sub-arrays are also divide using the approach provided and using the recursion till there is only one element in the array. At this stage the arrays are sorted, finally the arrays are combined to form an array. This sorting algorithm is highly efficient for big datasets as the average and best case complexity is O(n*log(n)).The space complexity is of O(log(n)). This sorting technique considered unstable since it does not maintain the key-value pairs initial order. The worst case time complexity is $O(n^2)$ it happens when the chosen pivot is the largest or the smallest. The best case occurs when the pivot is the average or and the size of the sub-arrays are equal or differs within 1 of each other. The quick sort is out of place

algorithm, as quick sort requires O(log *n*)stack space pointers to keep track of the sub arrays in its divide and conquer strategy. Consequently, quick sort needs additional space. So non-constant space takes the quick sort out of the group of in-place algorithms.

1.3)Merge sort

Merge sort algorithm as well as quick sort is based on the divide and conquer paradigm. It is one of the most efficient algorithms, it divides the list into two equal parts, calls itself recursively and the merge the two sorted halves. The sub-lists are divided up until there is no chance to divide again. Then combining the pair of one element list into two element lists sorting in the process. The sorted two elements pairs merged into four elements lists, and till the list is sorted. Time complexity or merge sort depends on the number of comparisons made within merging the sub-arrays in sorted manner. Therefore, the worth case scenario is when both left and right sub-arrays will contain the alternate numbers. For instance, if left sub-array consist of {7,9,11,13} and right of {8,10,12,14}, so every element should be compared at least once in order to be merged, this will create worth time complexity of merge sort O(n*log(n)). Best case – when no sorting required, array is sorted already and it is of the form O(n*log(n)).The space complexity is O(n) as extra variable required for swapping. The algorithm is stable.

1.4) Heap sort

Heap sort is one of the sorting algorithms used to arrange a list of elements in order. Heapsort uses one of the tree concepts called Heap Tree. Binary Heap Tree. Heap is a binary tree in which each node is either greater than or equal to its children(Max heap) or less than or equal to its children (Min heap).

Max heap is the largest element and is always at the top. The heap sort consist of two phases: In the first phase, the array to be sorted is converted into max heap. And in the second phase, the largest element is removed, and a new max heap is created. The array to be sorted must first be converted to the heap for this new data structure is created. Then max heap is created, visiting all parent nodes backward from the last one to the first. After heapifying we take the advantage that the biggest element is always on the top, swapping the root with the last element. Then heapifying is used again. Then swapping the top with the last element. Heap sort is an in place algorithm which says that no more memory allocation is required. Heap sort is not stable as the order of the equal values can be changed at the heap. Somewhat slower than quick sort except for the worst case. Heap sort is efficient in the time, memory and simplicity. The space complexity is O(1). The best time complexity is O(n*log(n)). No worst time complexity.

1.5) Hybrid sort.

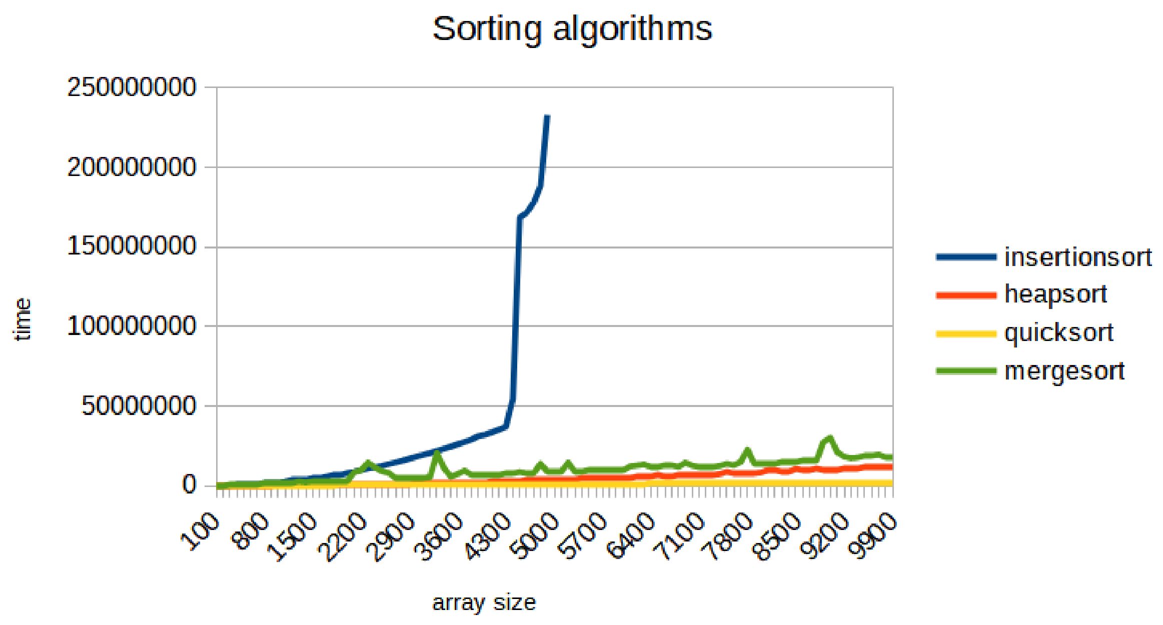
Hybrid sort is the sorting algorithm which was constructed from insertion sort and quick sort, as for the rather small arrays which are less than 75 elements insertion sort is the fastest and for the other arrays quick sort is the best option.

2) **Methodology**

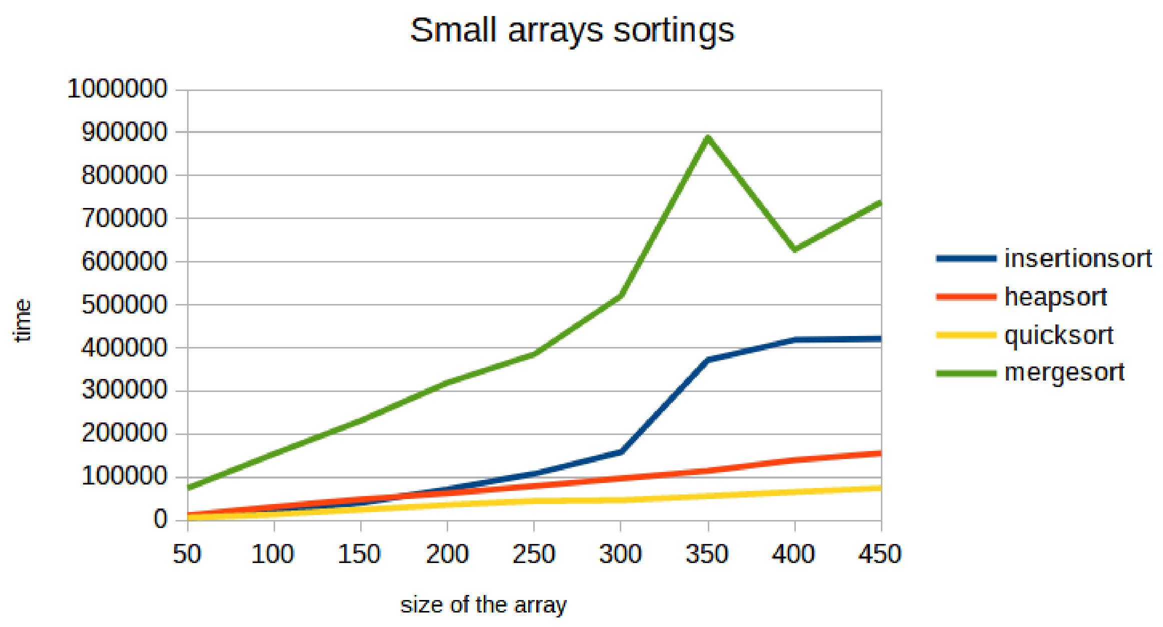
Programming language which was used is C++. The outputs were produced for 3 random vectors. The algorithms were used for vectors with the size from 100 up to 10000 with the step 100.The second step was in measuring the arrays with the size up to 500, the last testing was done on the arrays with the less than 100 elements. Each array size was tested 100 times. In the result the time of sorting indicated.

3) **Result**

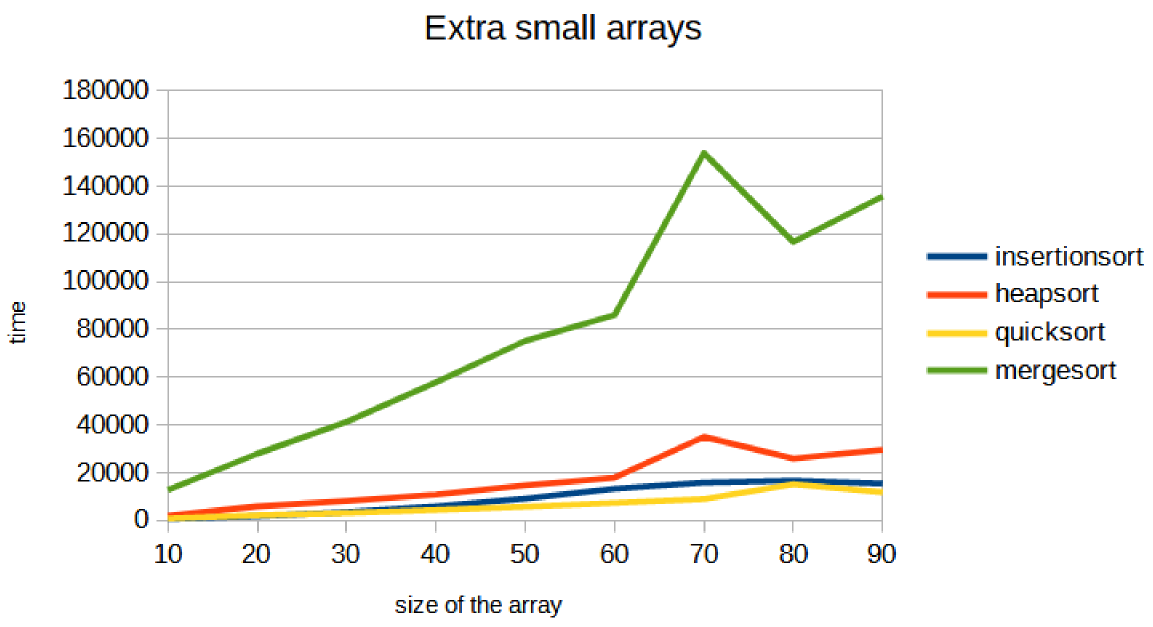
Graphs of result for the average time of every sorting algorithm is indicated at the picture below:



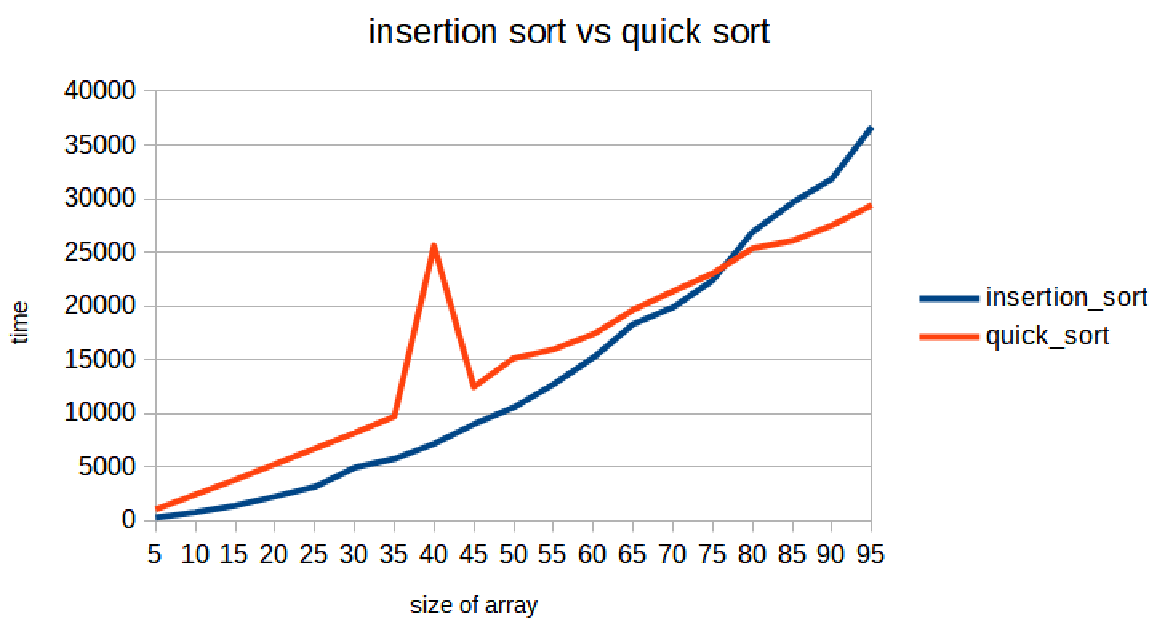
It is clearly visible from the graph that the insertion sort was checked for the values up to 5000, because for the greater values the sorting was completely difficult to measure due to the time consumption. With the array size of 5000, insertion sort took 9 times more than for the rest. It is also apparent that starting with the array size 2000 quick sort is the least time consuming. For the next determination, the sorting will be performed for the small arrays up to 500 with the step 50. As it can be seen at the graph given below, the insertion sort, quick sort and heap sort perform equally for the arrays with the size up to 100. Over this range, quick sort is the fastest, with heap sort being the second and insertion third fastest.



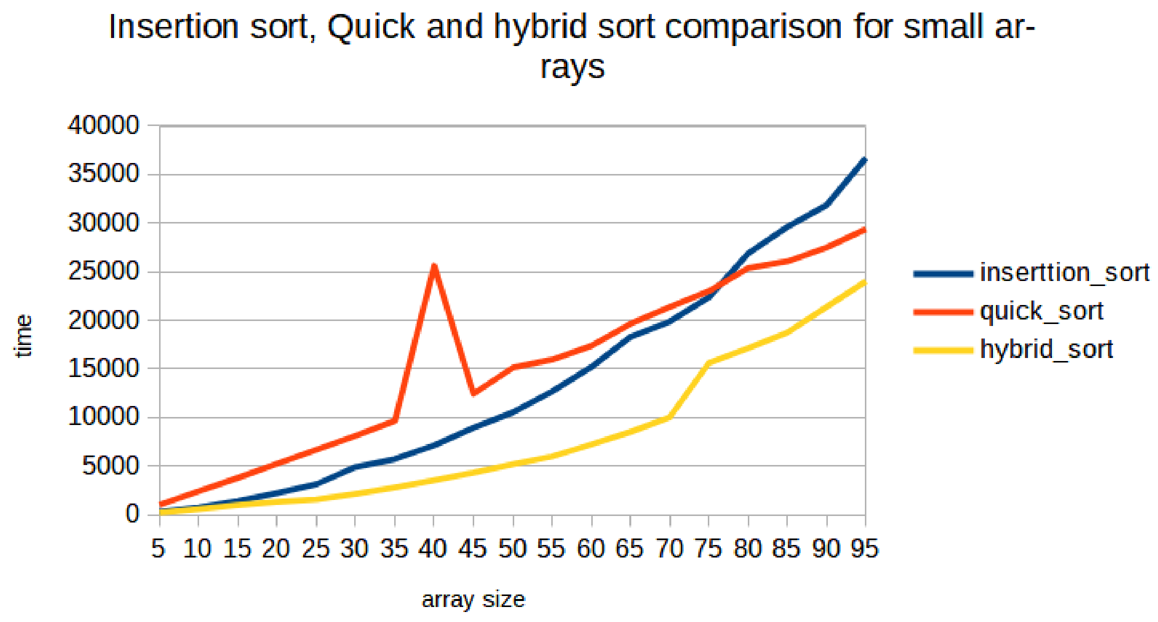
It is not really apparent from the graph which algorithm is the fastest for the arrays with the size of less than 100, because heap sort, insertion sort and quick sort perform almost equal. So the graphs of smaller arrays should be introduced.



From the graph above it is visible that for all size heap sort is the worst among the others. So the last step is to measure insertion sort and quick sort and compare them together and create the hybrid sort if necessary.



It is visible from the chart, that insertion sort is faster for the arrays up to 75 values. So the hybrid sort was introduced.



In conclusion, generally quick sort is the fastest one for the arrays with big size and insertion sort is in winning position for the arrays with less than 75 members, seeing that hybrid sort was constructed being the combination of quick sort and insertion sort.