Project: Exploring Insertion, Merge, Quick, and Heap Sort Performance

Hassan Samine

1 Introduction

In this paper, we will explore the problem: "Arranging elements of an array in a certain order" using four different sorting algorithms: Insertion Sort, Merge Sort, Quick Sort and Heap sort. Additionally, we will compare these algorithms to determine which one of them preforms better.

1.1. Insertion Sort

The insertion sort is an algorithm that simply works the way we sort playing cards in our hands.

In insertion sort we start by assuming that the 0^{th} element is in the correct placement, and then we select the 1^{st} element and compare the two, if the selected element is bigger we don't change it's placement, and if it's smaller we put it on the left of the 0^{th} element, or in other words we swap their positions, and then we move on to selecting the next element in the array and compare it to the two elements before it, and according to that we put it in it's correct place, with the smaller element on the left and the bigger one on the right, we keep repeating this pattern for all the remaining elements until we have a sorted array.

1.2. Merge sort

The merge sort is an algorithm works with the divide and conquer principle, which means that if we face a big problem, we must divide it into many sub-problems and divide them aswell until we reach the simplest sub problem possible, and after solving this sub problem we start merging them back on again.

In merge sort we continuously divide the array into half (if the number of elements is not an even number we put one more element in the left array) until we are left

with individual elements, and then we examine them two by two, compare them and merge them into temporary sorted arrays, and then we compare the temporary arrays we have and then merge them into 4 elements sorted arrays, we keep repeating this until we sort our initial array.

1.3. Quick sort

The quick sort algorithm is also an algorithm based on divide and conquer principle, to perform this algorithm we choose a pivot element at index 0, and then we move all the elements that are less or equal to the pivot to its left side and the elements that are larger on the right side, and by doing this, now we have the pivot in the correct final position in the sorted array, and on the it's sides we have a left sub-array and a right sub-array that are less and bigger than the pivot respectively, we perform this pattern for both the sub-arrays until we have a sorted array.

1.4. Heap sort

The heap sort is an algorithm that simply find the largest element in the array and sort it at the end and keeps doing this repeatedly until we are left with a sorted array, but the way it does this is using the heap method, the heap method's purpose is to do this task faster than a normal comparison of all the elements to find the largest value, and to do this it sorts the array in a complete binary tree.

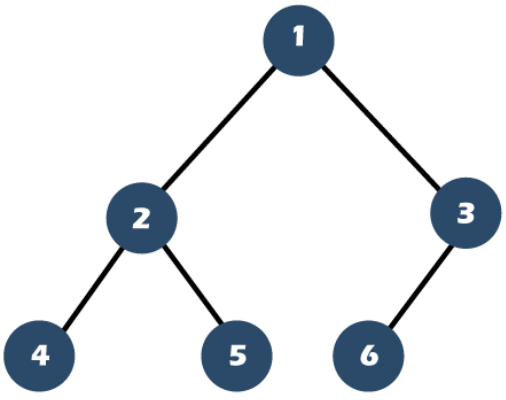

Figure 1: Complete binary tree

The 1st element of the array is stored in the 1st spot of the tree, the second element in the second spot…

After putting the array in the complete binary tree we start comparing the child nodes to their grand children starting by the node number 3 (array size / 2) we compare it to see if any of it's child nodes is bigger* than it (*depends if we are sorting from ascending or descending) and if it is we swap the two elements, and then we do the same until we have a max heap, where the biggest element is in number 1, the second biggest in number 2 and so on, this method is called to heapify the tree. After that we put the element in the node number 1 in the ending of the array and remove it from the tree and put the smallest element in the tree (element that is in the 6th node) in its place, and heapify the tree again. We keep doing this pattern until we have a sorted array.

2 Methodology

The algorithms were implemented in C++, and we tested with arrays generated using the <random> library that provides various random generators.

The results were generated using arrays of a size from 100 until 10 000 with a step of 100 between each size, with the elements values varying from 100 000 and -100 000. And to measure the performance we repeat the algorithm 100 times and average the results.

3 Results

In the figure 2 below we find the results for all sorting algorithms with the array sizes varying from 100 until 1000:

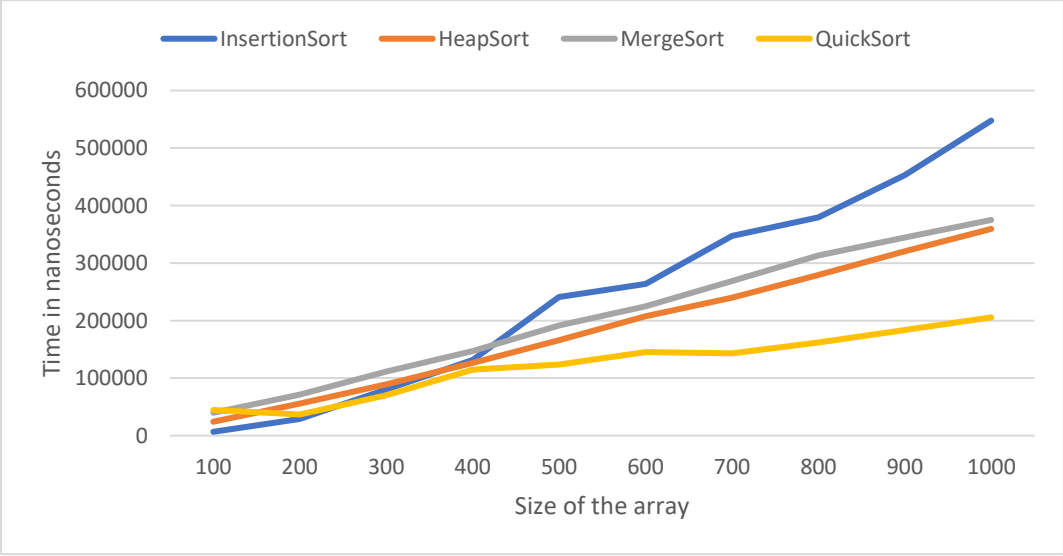

Figure 2: Average time each algorithm took to sort an array of a size varying from 100 to 1000

In the figure 3 below we find the results for all sorting algorithms with the array sizes varying from 100 until 10 000:

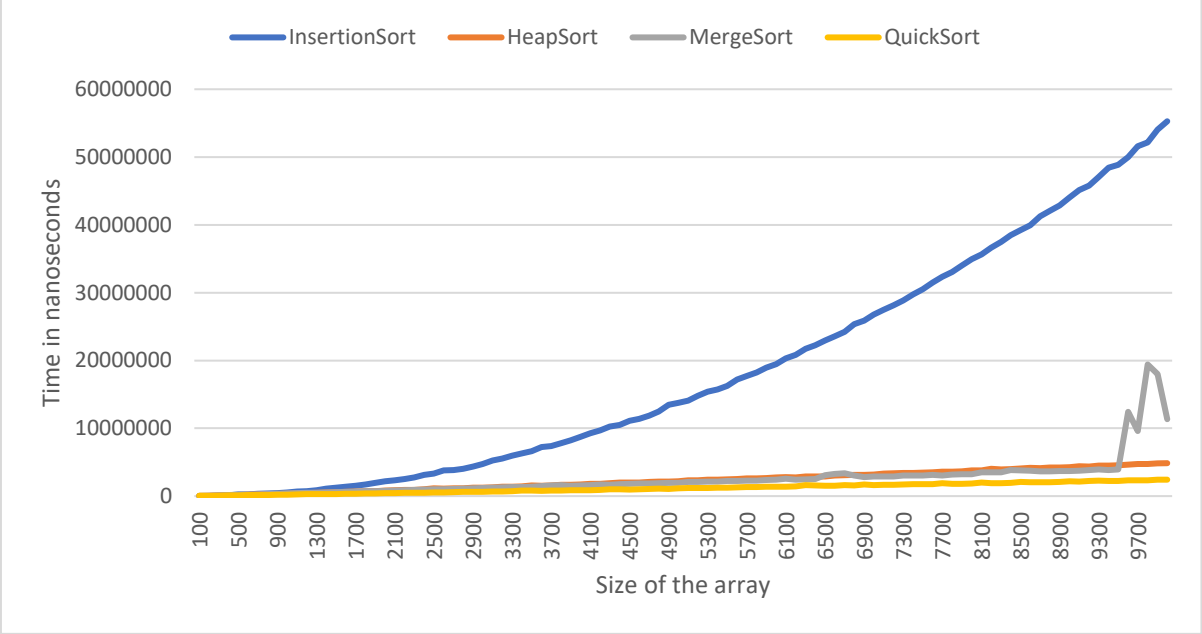

Figure 3: Average time each algorithm took to sort an array of a size varying from 100 to 10 000

4 Conclusion

We conclude that for small size arrays the difference between the 4 sorting algorithms is not too significant, but the quicksort is the fastest out of the bunch, this fact is more visible for the big size arrays. but both the heapsort and mergesort don’t take a lot more time than the quicksort, the only algorithm that we can see is much slower compared to the other algorithms is the insertion sort.

We conclude that out of the four algorithms the one that performed the best is the quick sort and the one that performed the worst is the insertions sort.