Sorting Algorithms: comparison between

insertion sort, heapsort, quicksort and radixsort

1 Introduction

In this paper, we investigate four sorting algorithms. Insertion sort, heapsort, quicksort, and Radixsort. We will compare them on different data and decide which one is more suitable than the others. Also, we will consider a way to combine these algorithms into something more optimal.

1.1 Insertion sort

In this implementation of insertion sort, we consider the first element as sorted, and then by moving forward we compare if the current element is lower than the previous one, and if it is we put it before previous. The algorithm has an average complexity of O(n^2) , the worst case is O(n^2) and the best case is O(n). This algorithm is stable and in place.

1.2 Heapsort

To make a heapsort, firstly we should create a maxheap. To do it we use an array with the idea that the left child is located at position 2*k+ 1, where k is the current position of element in the array, and the right child is located at position 2*k+ 2. Heap is made by “fixing” all elements end to the start of the array. Fix function takes the array, its size and index, then if the elements have a child that is bigger than the parent, we swap it with a parent and again fix elements recursively. After we made a maxheap, sorting is quite simple, we swap the first element with the last one, decrement size, and fix until size != 0. So after this simple operation, we got a sorted array. Heapsort has the same complexity for best, average, and worst case – O(n*log n). Space complexity is O(1).

1.3 Quicksort

Quicksort is one of the fastest sorting algorithms and is commonly used in different algorithms, it consists of two main parts: partition and recursive call. To partition an array, we choose a random pivot, it’s one of the most optimal schemes and you almost never get an O(n^2) case. Firstly we swap the

random element with index 0. Then we use Hoare's algorithm to partition our array. In Hoare's partition scheme we take two indices, first is located right after the pivot(located on index 0), so we take value 1 as an index, and the other index has a value of size-1. The idea is that we iterate through elements and if an element on the left is bigger than the pivot and an element on the right is lower than the pivot we swap it. Then we swap the pivot with an element that is the last element that is lower than a pivot. And after all of that, we recursively run the quicksort partition from start to pivot exclusively, and from an element that is goes after pivot and to the end of the array.

Quicksort has the average and best complexity of O(n*log n), the worst case can be O(n^2), but it uses a random pivot, so chances are almost zero. Quicksort space complexity is O(log n).

1.4 Radixsort

This implementation of radixsort has two realizations, one that is capable of working with negative numbers and one that is not. Obviously, the last one is the fastest option. To sort an array with radixsort we use counting sort on ones, tens, hundreds, etc. To implement radixsort which is capable of sorting negative numbers we added some number that is equal to max_number + absolute(least_number) + 1 , and then after sorting we reduce all values again by this number. This operation O(n) time, and it doesn't change the complexity of our sorting algorithm, but measurements said that is slightly slowed down this algorithm. Radixsort has the same complexity for the worst/best/average case of O(n*k), where k can from O(1) to some very large number, which will make radixsort worse than the algorithm with complexity O(n^2). The space complexity of Radixsort if O(n+ k).

2 Measurments

Algorithms were implemented in c++ . The results were generated for random numbers (-size, size) for two categories array size for small sizes [1, 200], and array [100, 10000]. Average 10000 repeats were taken.

3 Results

Firstly let's have a look at Figure 1, we can see that insertion has the the fastest growth rate, and quicksort is the fastest algorithm here.
In Figure 2, we can see that on values <170 - insertion sort is the fastest one, and then after size of 170 quicksort is the fastest.

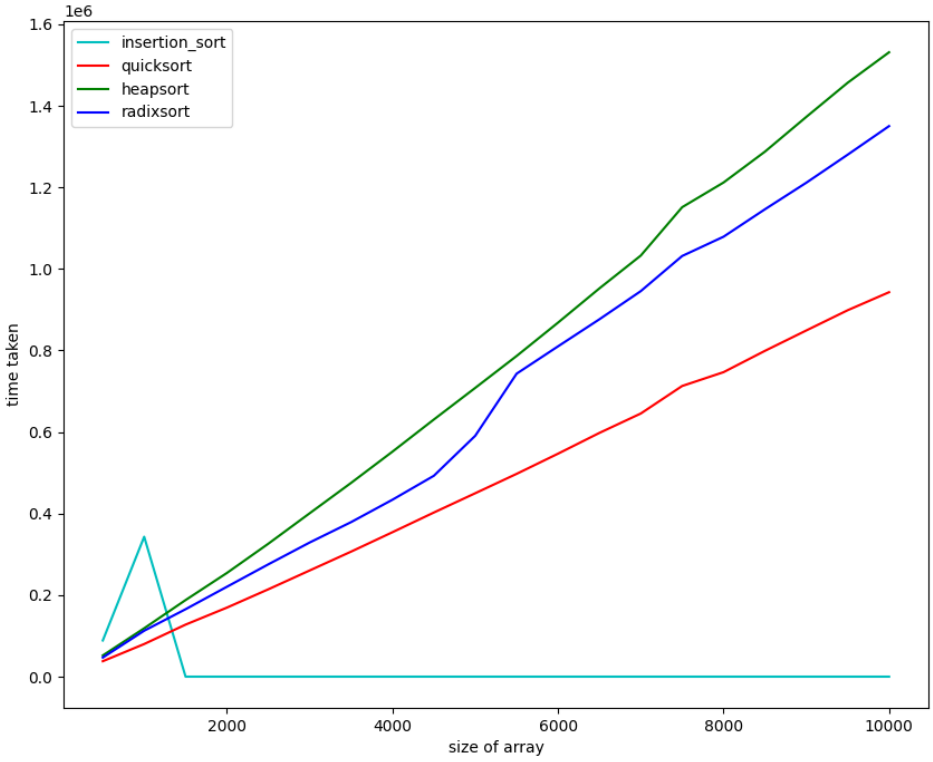

Figure 1, large size

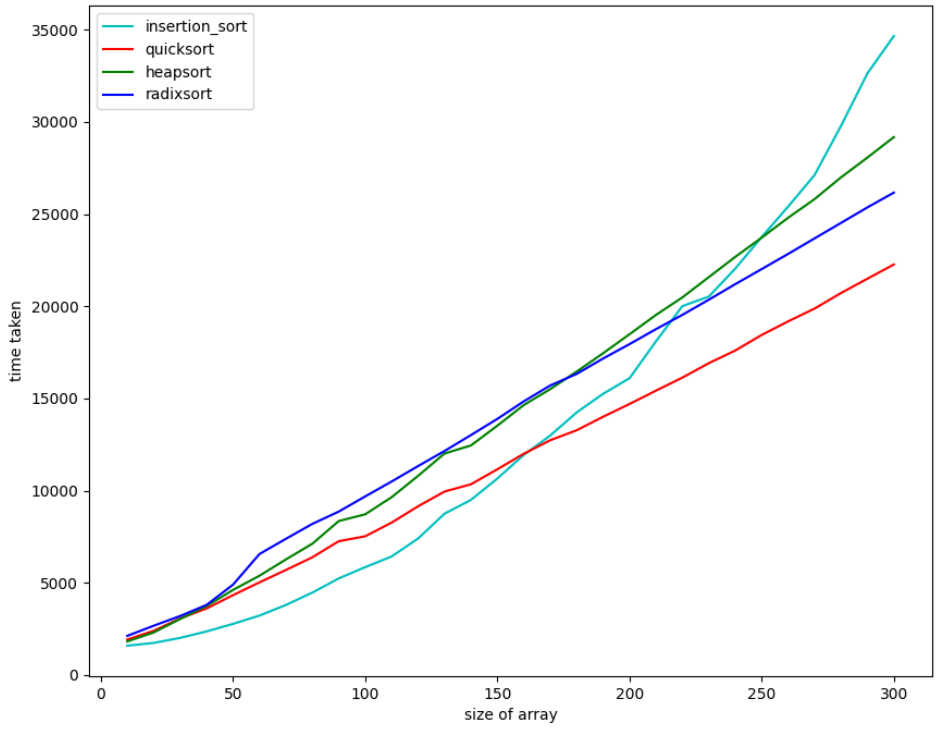

Figure 2, small size

3.1 Description of Hybridsort

I decided to combine quicksort and insertion sort for values < 170, because it shows more effectivity on it, also if quicksort has pivot that size less than 170, we use insertion sort (graphs on next page).

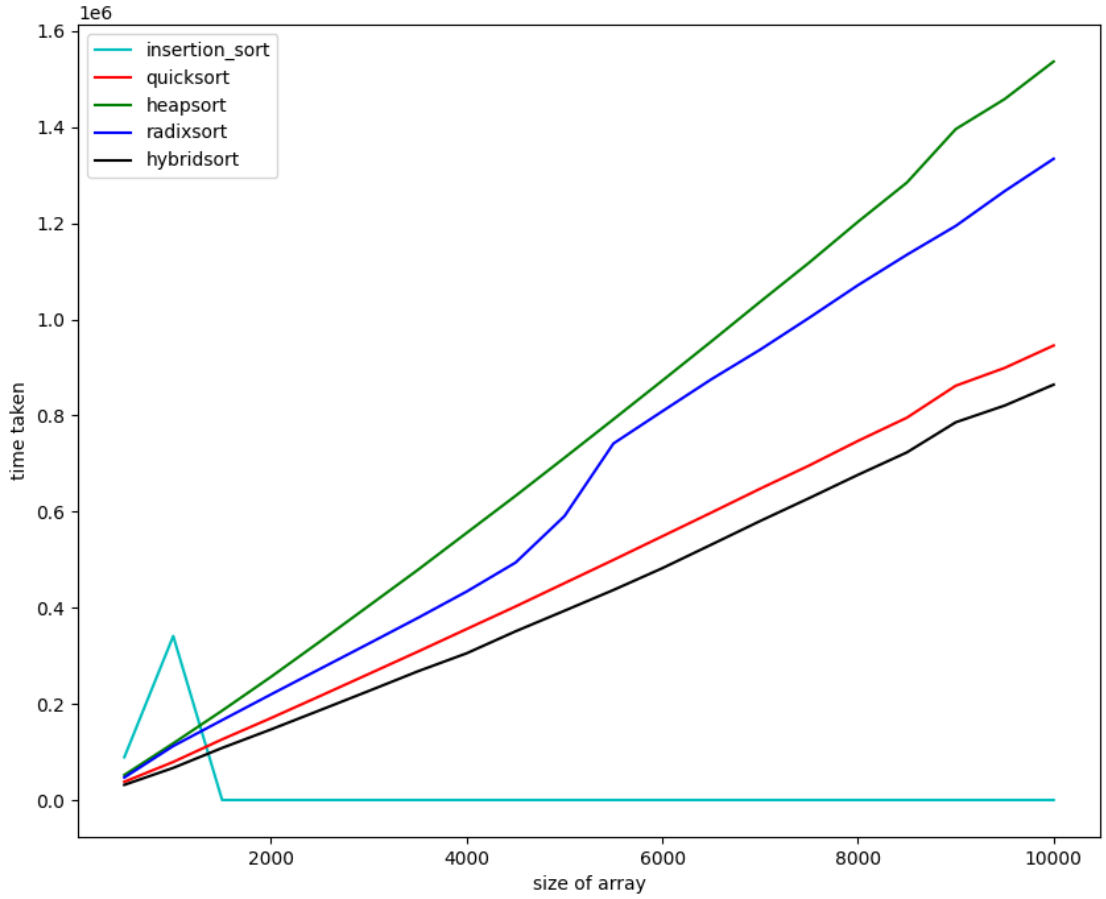

Figure 3, large sizes with hybrid sort

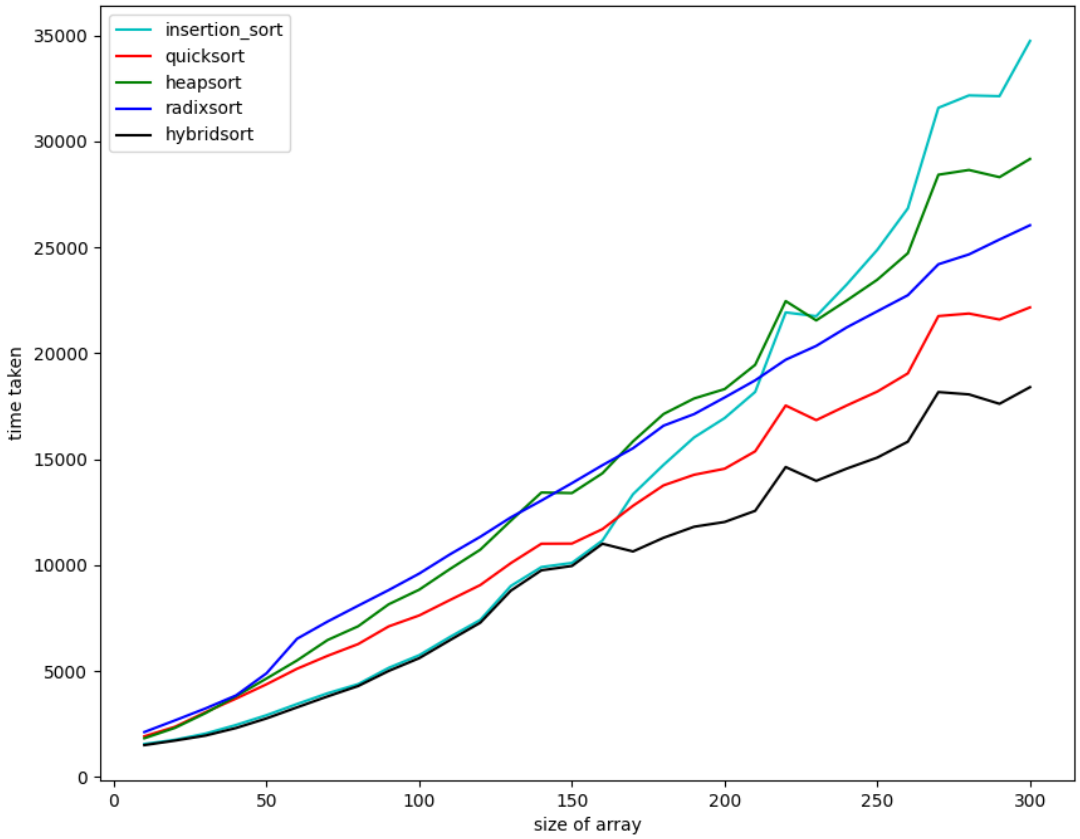

Figure 4, small numbers with hybrid sort

4 Conclusion

Hybridsort is the faster than other sorts or at least as fast as the best algorithm on every size.