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 swap it, and check again until the previous element is lower or we get to the start of the array. 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 of 100 repeats for large array and 1000 repeats for a small one were taken. Also, arrays with numbers in range (-10, 10) and (0, 10) were measured, for large and small arrays. Also, sorted arrays were measured.

3 Results

Firstly let’s have a look at figure 1, we can see that insertion sort takes much more time than other sorting algorithms.

In figure 2, we can look more precisely into graphs, insertion sort is still the worst, and quicksort is slightly faster than others.

Figure 3 shows that on a small range of values, radixsort is way better than other algorithms.

Figure 4 shows that on range high [0, size], quicksort is still the best one. But let's have a look at the smaller graph.

In figure 5 we can see that on the range [0, 200], with random values from 0 to size, we have a part, where radixsort, has better speed than quicksort.

And finally figure 6 shows that on a sorted array we got really fast insertion sort, and quicksort which is slightly faster than other algorithms.

PS: excuse me for the format of the pictures, they are way too big for a text redactor and I have no idea how to fix it, so maybe better to have a look at the files in the folder.

Figure 1, all sorting algorithm together, random numbers [-size, size]

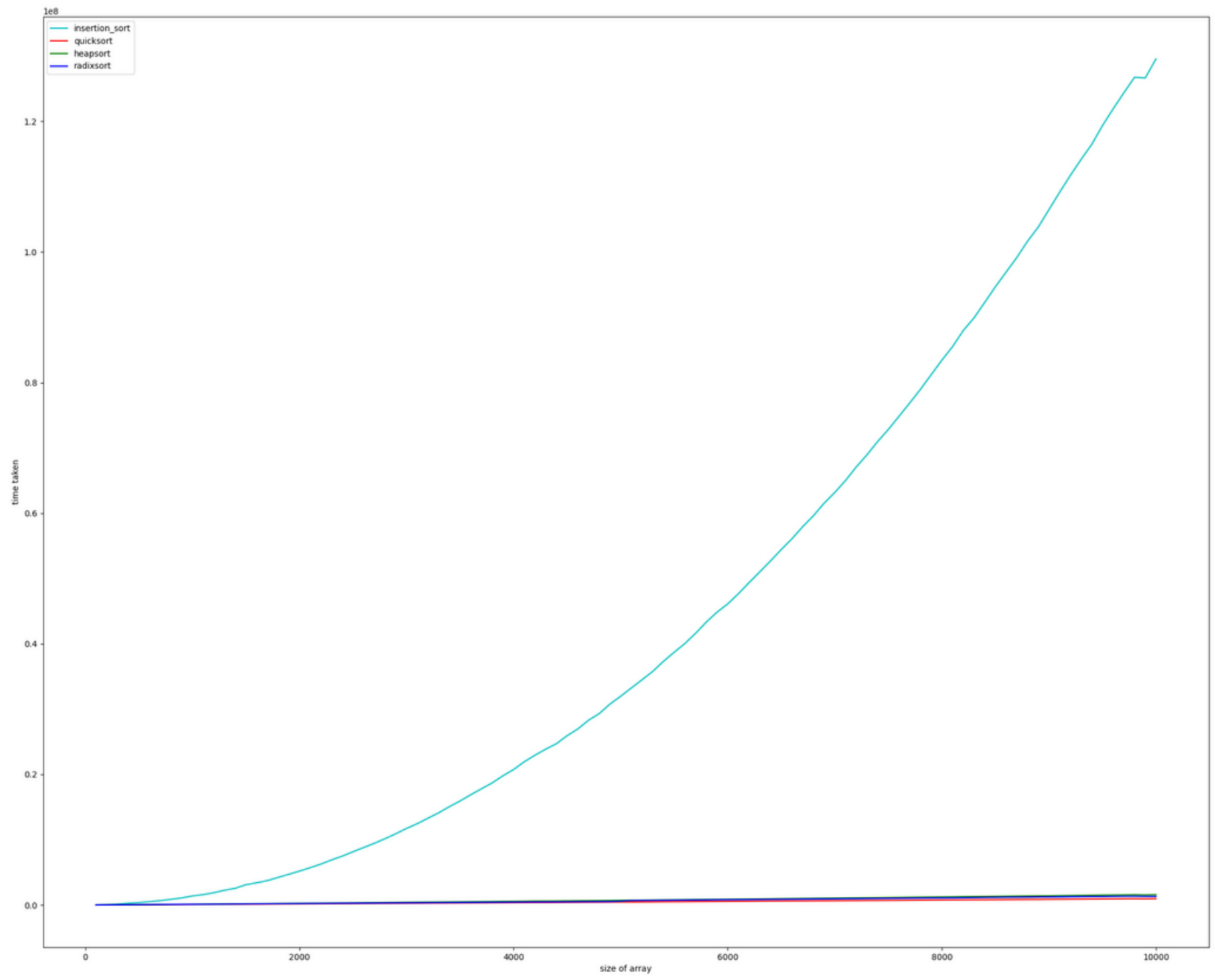


Figure 2, all sorting algorithms on array [0, 200], with random numbers [-size, size]

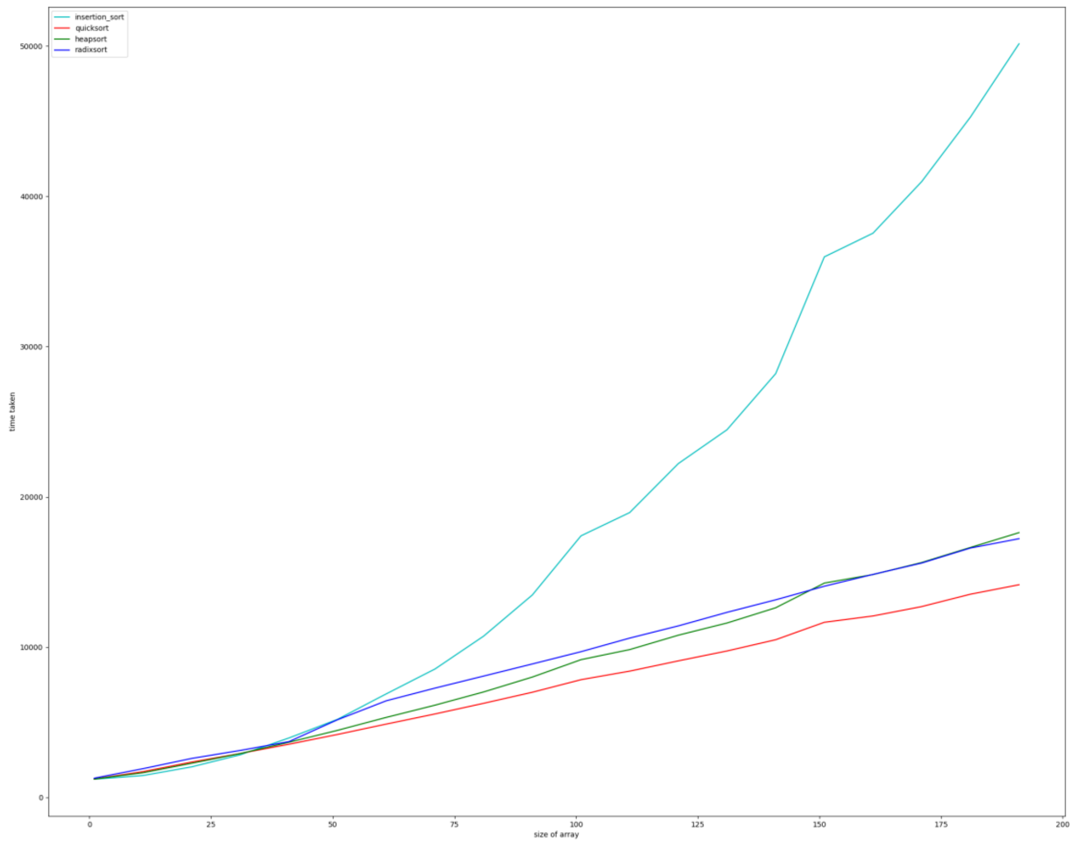


Figure 3, sorting algorithms on random values [-10, 10]

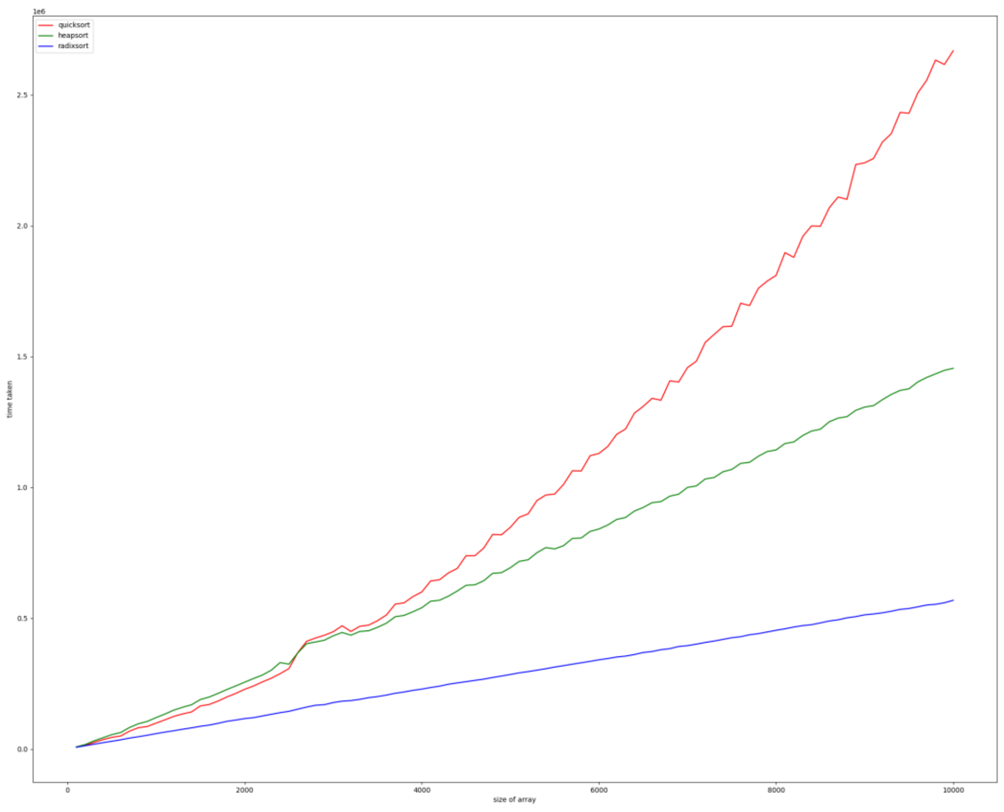


Figure 4, sorting algorithm together, random numbers [0, size]

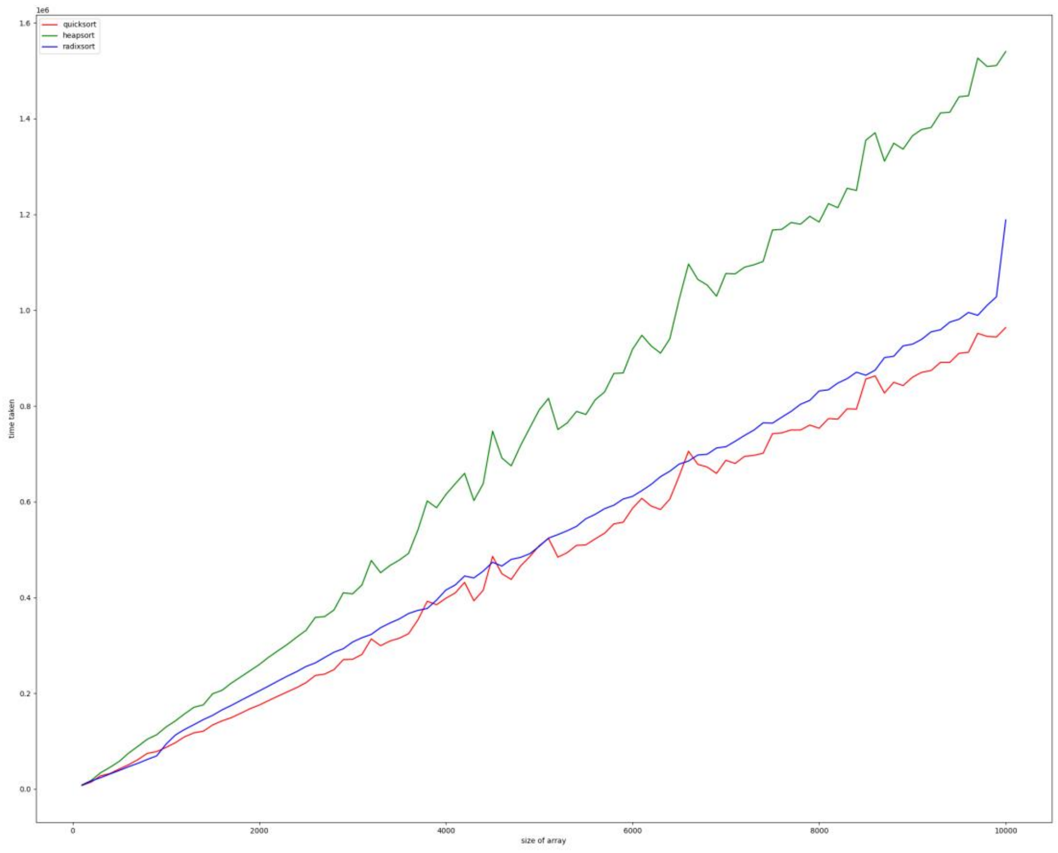


Figure 5, array on range [1, 200], with values [0, size]

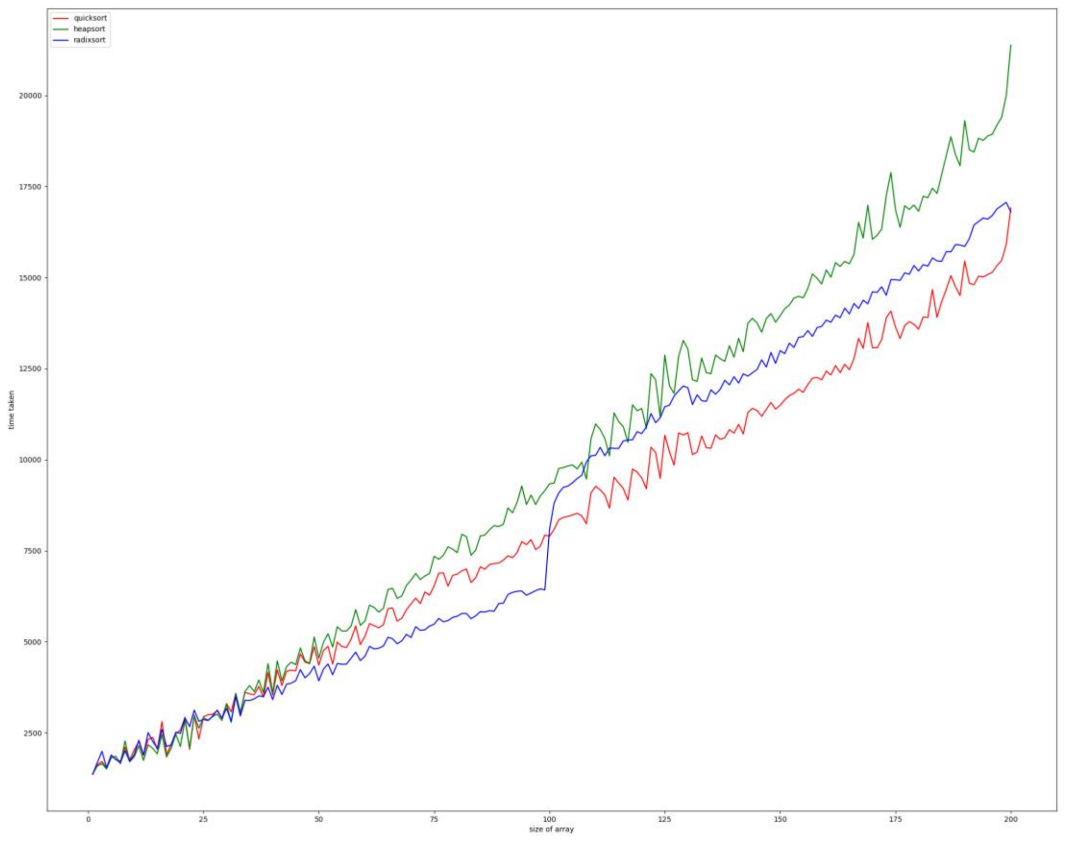
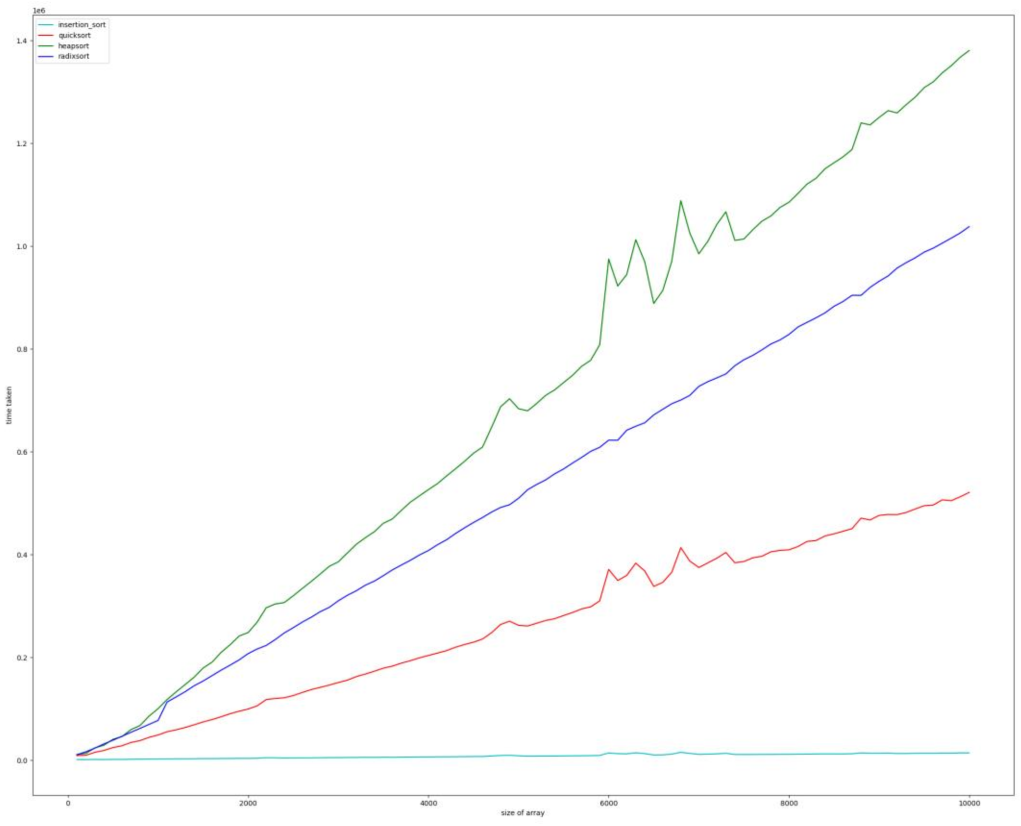


Figure 6, all sorts on sorted array



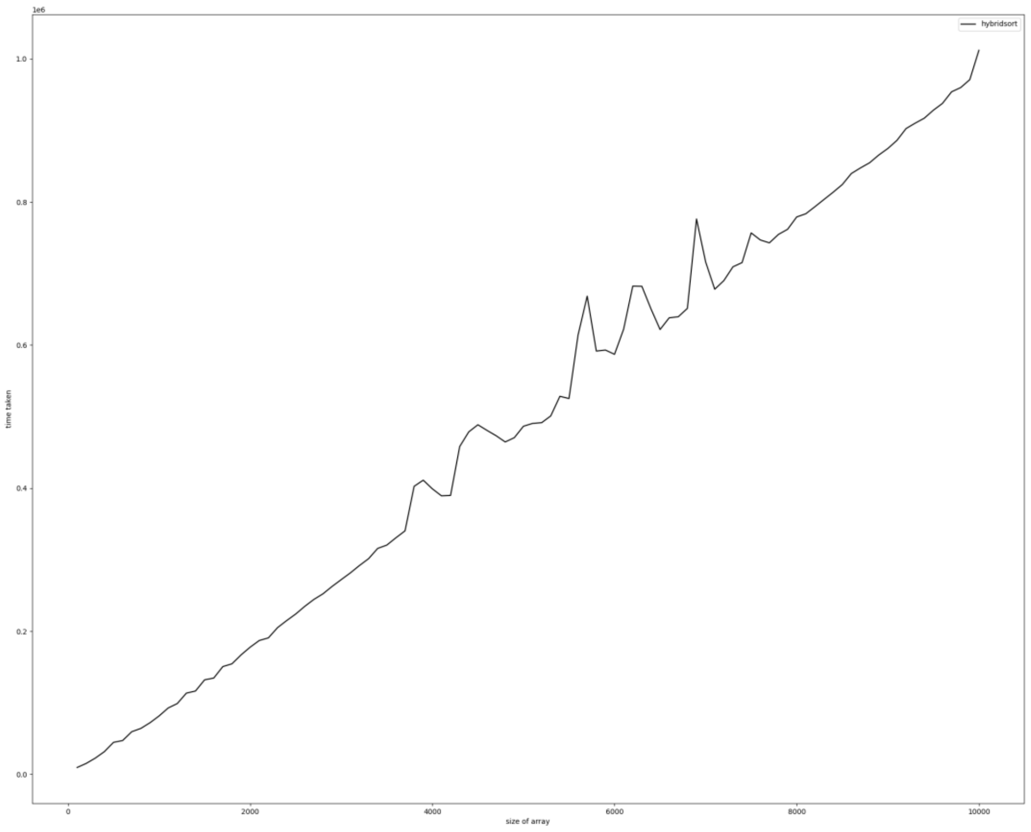
4 Conclusion

On an array of a random numbers with a big range quicksort is faster than other algorithms, on sorted array insertion sort is much faster than others. Radixsort is faster when we get a non-sorted array with a small range of numbers. To sum up: Quicksort is faster at the most amount of tests, however, there are some cases where radixsort or even insertion sort outrun quicksort. So to make a better sort I decided to implement a hybridsort.

4.1 Description of Hybridsort

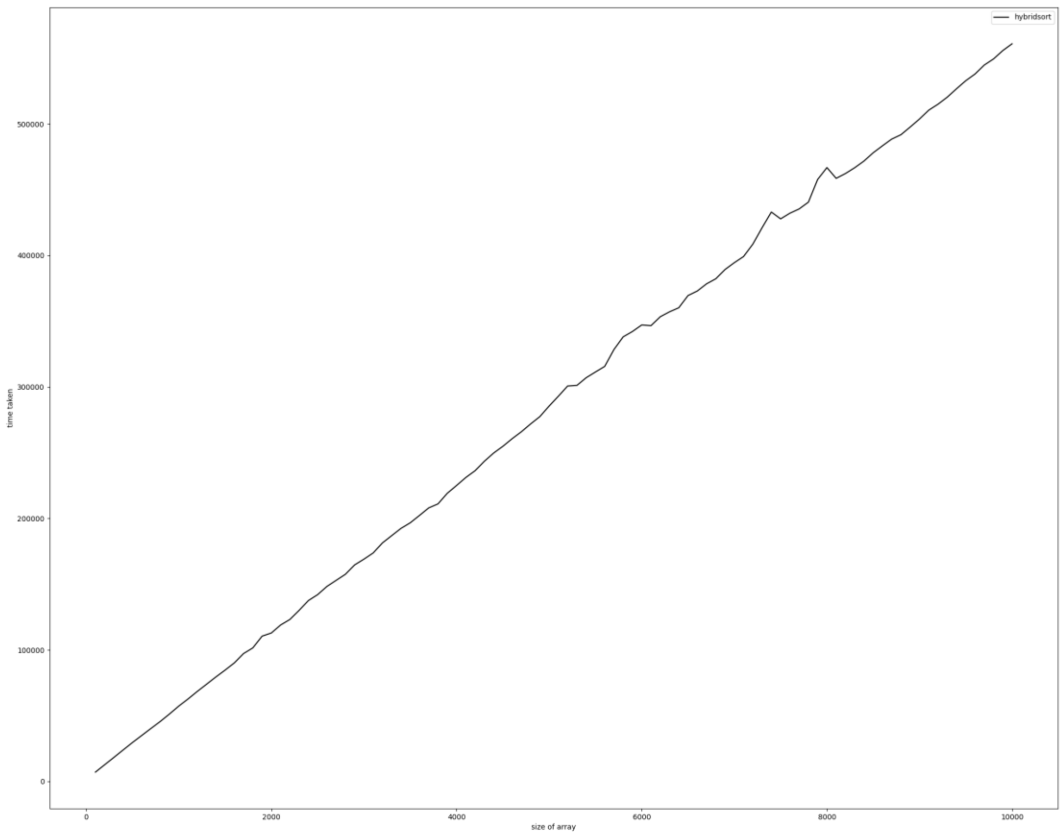
I decided to combine the best parts of his three algorithms.

If array is not-sorted and have a big range of values we use quicksort

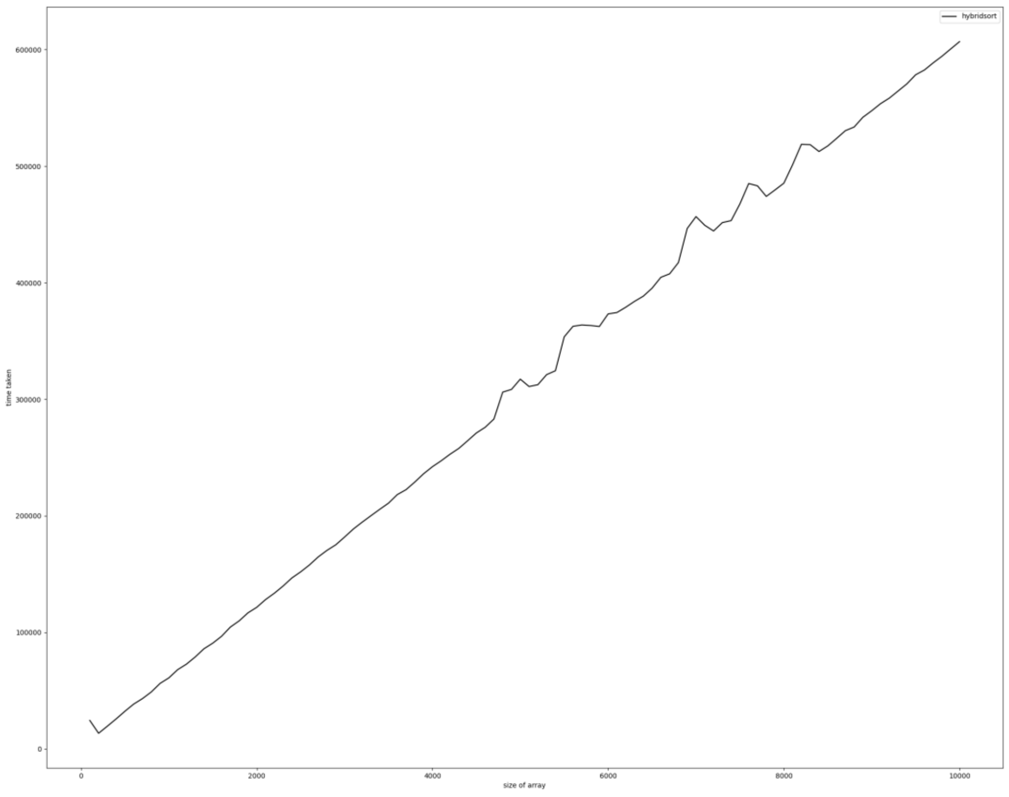


It takes a little bit more time than the original quicksort, because we additionally check some conditions, but still, it goes faster than heapsort or radixsort.

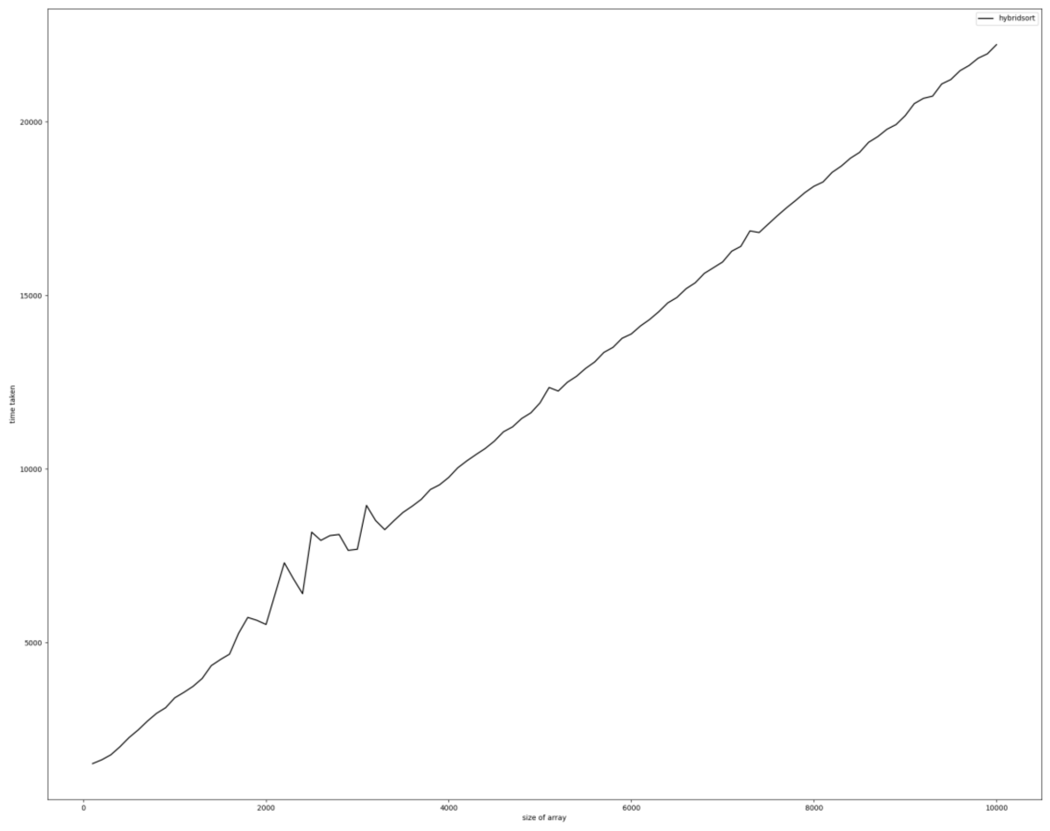
On a random array with positive values on a small range, it uses radixsort_positive, so it makes it suitable for small ranges.



On a random array with negative values on a small range, it uses radixsort.



And the last one sorted array, this sort is an absolute unit here, because it doesn't even sort an array, so it has time complexity of O(n)



Best case O(n), Average case O(n * log n), Worst case O(n^2) - only because of quicksort, but chances that it will happen are close to zero.