Traditionally, the problem of sorting a given array by the key values of its elements is a common problem in computer science and thus has been subject to extensive research throughout the evolution of the field.

Merge sort is a sorting algorithm, that is of special interest due to its O(n\*logn) worst case time bound. Therefore, when a guaranteed worst-case runtime matters merge sort is chosen over, for example, standard implementations of quicksort, which are usually very fast but have an upper bound of O(n²). At the same time, other worst-case efficient sorting algorithms like heapsort are often outperformed, giving merge sort the edge in many situations.

It employs the “divide and conquer” paradigm by dividing the given list into smaller sublists, either to a size of one element, that can always be considered sorted, or until some lower bound is reached, at which point another sorting algorithm is used. Those sorted sublists are then consecutively merged, creating one sorted list from two smaller ones at a time.

This allows very simple recursive top-down implementations, whose functionality is largely affected by the choice of merging strategies.

(Kommentar: sollte erwähnt werde, aber ich bin nicht sicher ob die Stelle richtig ist) As one can see, standard versions of merge sort can also be easily parallelized, but this property does not hold for the implementations we will present here, because they rely on using not yet sorted parts of the list as extra space to speed up the sorting process.

(…)

Despite its desirable properties, one big downside of naive merge sort implementations is the required linear extra space, making in-place algorithms like quicksort or heapsort a more suitable choice, if memory usage is essential.

Therefore, many in-place variants of merge-sorts have been suggested by several authors, ranging from implementations for in-place merges, that can then be easily integrated in a non-in-place merge sort algorithm, to entirely modified sorting schemes.

In this paper we will discuss a full in-place merge sort by Reinhardt and two in-place merging algorithms, one by Chen, another by Huang and Langston. We used the latter ones to implement simple top-down in-place merge sorts.

We will describe our implementations, giving special attention to parts where our implementation differs from the original algorithms, which applies especially to some versions of Reinhardt’s algorithm.

Afterwards we will give an overview on the various tests we ran, comparing our implementations both against one another and against existing in-place merge sort implementations. We focused on measuring the performance in means of time, comparisons and assignments. The latter ones are well established measures of complexity for sorting algorithms as they are independent of the machine the algorithm is running on.

(Evtl: We will also briefly discuss how these implementations perform compared to heapsort, one of the main competitors in their area of application.)