We started by implementing some versions of the standard merge sort with extra storage with size of the list-to-sort. In the files rec\_mergesort\_reference.cpp and rec\_mergesort\_iter.cpp, we implemented the recursive standard merge sort on a reference of a vector and on begin-/end-iterators. They work as well-known: Two lists are pairwise merged two one bigger list. Thereby we merge alternating per iteration between the extra storage and the original list. We implemented insertion sort and a small sort which can be used for small sizes in the last recursion step. The last step can be performed within the original list or between the lists so that a final copy of the whole list can be avoided in any case.

We also implemented an iterative merge sort (iterative\_mergesort.cpp) which similarly needs additional storage with size of the original list.

A disadvantage of the recursive variant is the logarithmic space for the internal stack. But on the other side, it is a simple and elegant way to split a long list into parts of equal length throughout all recursion levels. So it is important to remark that true in-place sort algorithms may also use extra space for recursive execution (Even though they just work on the original list per definition).

For test cases, we implemented different test environments. We used the main function in the “TestAndEvaluation”-project for our results given later. The user can choose between different types to sort, the list sizes, and a lot of other options (main.h). The user can start the test by executing the main procedure (main.cpp) and the results (average time, comparisons, and assignments per size of the list) are written to CSV-Files. We implemented a general BigType (structs.h) and a general PointerType to weight assignments and/or comparisons differently: Sorting the BigType makes the assignments more expensive and the user can choose whether the computation of the comparisons should also be expensive. To get expensive comparisons and cheap assignments, the user can choose the PointerType on the BigType with expensive comparisons.