Assignment 1

Task 1: "Ordnung must sein” 3-4

A. Implement bubble sort, insertion sort, merge sort, quick sort. Test and compare their running times for a number of

- small (ca. 5–50) and
- large input sizes.

Carefully choose (or even average) appropriate inputs to ensure you justify the right conclusion. Include a graph with the results of your measurements for all algorithms, e.g. running time as a function of array size.

B. The Olsen Gang has obtained a set of 20k credit card details. Unfortunately, when transferring through the dark web, due to a sloppy internet connection, the data set got split into two parts. Moreover, the card records in the second part got randomly shuffled, so, before they can sell the data for big money, they must match them up with card numbers from the first part. Luckily, this is possible, as it is known that the records in the first dataset go in the order of increasing expiration dates and the PIN. Choose a linear time algorithm for the job and help the gand cash out.

Empirically investigate at what data volume the linear solution gains supremacy over a log-linear algorithm.

Task 2: "Dangerous Minds” 4-5

- Implement your own version of C++ vector - a dynamically reallocated array with table doubling. This could be implemented just for integers. Your implementation should include functions for read and write access of elements, appending, resizing (truncating if downsizing), and deletion of one element or a segment of elements.
- Test accumulated running times for large sequences of append operations. Compare it with the same test for the C++ vector (or whatever library solution) and your own implementation of a linked list. Is it possible for you to see when the vector implementations do the reallocations?

Remember that the time of element access should be $O(1)$ and the amortized time for element append has to be $O(1)$.