Algorithms and Data Structures for Biology
10 May 2019 — Lab Session

Ugo Dal Lago Thomas Leventis

1 Measuring Algorithmic Space Consumption

Sometimes, not only the time consumption, but also space consumption of your algorithms can be of interest. In particular, two algorithms solving the same combinatorial problem and having comparable time complexities, can have *very different* space consumption behaviours. As a consequence, the first one could exhaust the available memory sooner than the other one. Analysing the amount of space algorithms consume as a function of the input size can thus be useful in many cases. Experimentally, thus at the level of the underlying Python implementation, this can be done in many different ways, e.g.,

- The package `memory-profiler` can be used to count the amount of objects allocated on the heap, and the total size of the heap for each code line. An executable `mprof` provided by the package is particularly useful, in that it produces a graph of the memory consumption as a function of time.
- Similarly, the package `guppy` can be used to take a snapshot of the heap, this way having an idea of how large the heap is.

2 Comparison Analysis of Sorting Algorithms

We ask you to compare two sorting algorithms among those you know, e.g. Selection Sort, Merge Sort or Bubble Sort.

- For both of them, measure their time consumption on random lists of natural numbers (generate them as permutations) of size $n = 1000m$ where $m \in \{1, \ldots, 10\}$, and profile it by way of `cProfile`.
- Analyse the space consumption of the two algorithms by way of `mprof`.

Try to answer the following question: is one of the algorithms always better than the other? Is there a critical value n^* of n such that for smaller values of n one algorithm outperforms the other, while for bigger values of n?