Home Exercise 3: Sorting

Algorithms and Complexity lecture
at CentraleSupélec / ESSEC

Dimo Brockhoff
firstname.lastname@inria.fr

due: Friday, October 9, 2020

Abstract

Please send your solutions by email to Dimo Brockhoff (preferably in PDF format) with a clear indication of your full name until the submission deadline on October 9, 2020 (a Friday). Groups of 5 students are explicitly allowed and highly encouraged. In the case of group submissions, please make sure that you submit maximally three times with the same partner!

1 Insertion Sort with binary search (5 points)

Run the Insertion Sort algorithm with binary search on the following (integer) array:

503	87	512	61	908	170	897	275	654	426	154	509	612	653	765	703

Please indicate how the array looks after each step. How many comparisons did the algorithm perform?

2 Mergesort (5 points)

Run the Mergesort algorithm on the following array and, similar to the above, show the array content after each step.

510	57	512	38	909	241	897	250	653	499	154	511	612	677	865	777

3 Implementing Mergesort and Comparison with Timsort (5+5 points)

Part I (5 points) Implement your own Mergesort algorithm that is able to sort any lists of floats. For this, please use the code in the jupyter notebook that is provided at `http://www.cmap.polytechnique.fr/~dimo.brockhoff/algorithmsandcomplexity/2020/exercises/sorting_template.ipynb`. Concretely, implement the Mergesort algorithm within the function `def mergesort(ll)` and make sure that the test is passing (i.e. the output is something like `TestResults(failed=0, attempted=4)`).

Part II (5 points) Compare the differences in runtime between your own Mergesort implementation and python's internal Timsort (via the `sorted(...)` function) on randomly generated lists of integers. To this end, plot the times to sort 100 lists of equal length n with both algorithms for different values of $n \in \{10, 100, 1\,000, 10\,000\}$.
Tip:

```
>>> import timeit
>>> timeit.timeit('your code', number=100)
```

Another (even more important) Tip: use the `?` to get help on a module (and `??` to inspect the code).

Questions:
What do you observe? Which algorithm is faster? Is one algorithm asymptotically faster?
For both parts, please send your python code as well!