HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI
MATEMAATTIS-LUONNONTIETEELLINEN TIEDEKUNTA
MATEMATISK-NATURVETENSKAPLIGA FAKULTETEN
FACULTY OF SCIENCE

String Sorting in Python – Comparison of Several Algorithms

Onni Koskinen, Arturs Polis, and Lari Rasku

TESTING DATA

Dataset	Number of strings	alphabet size	Sum of LCP array
dna.100MB	618	15	4501
dna.200MB	1114	15	8948
proteins.100MB	359505	24	18853436
proteins.200MB	709116	24	50076184
urls.100MB	3284368	114	94113004
urls.200MB	6576059	114	191545831
words.100MB	18502734	211	83643408
words.200MB	37003241	220	168115390

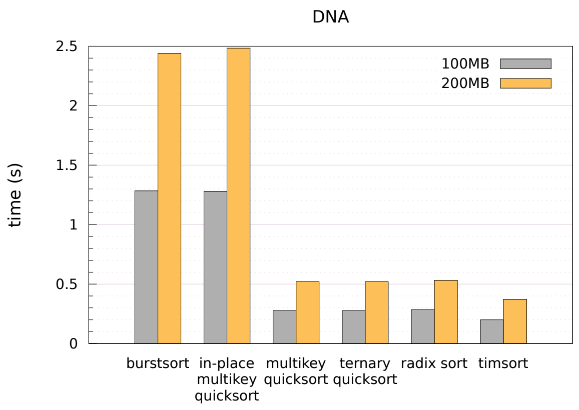
Table 1: Data set used for comparing the algorithms

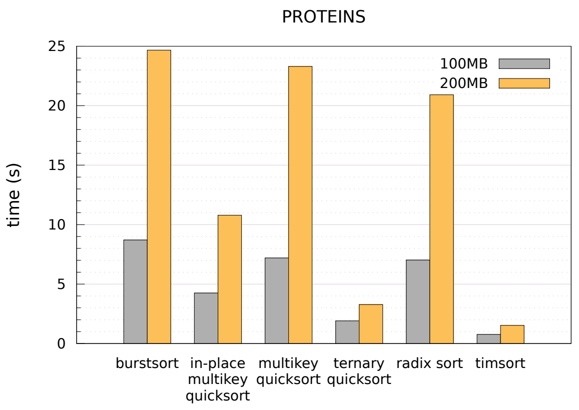
TEST RESULTS

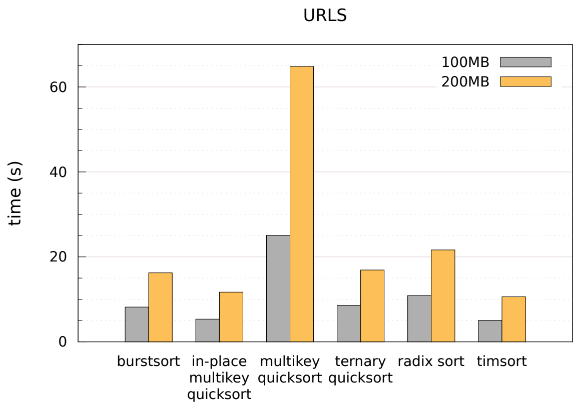
Dataset	timsort (builtin)	MSD radix sort	multikey quicksort	ternary quicksort	burstsort	in-place multikey quicksort
dna.100MB	0.2	0.284	0.276	0.276	1.284	1.28
dna.200MB	0.372	0.532	0.52	0.52	2.44	2.484
proteins.100MB	0.768	7.024	7.2	1.908	8.705	4.252
proteins.200MB	1.532	20.921	23.301	3.272	24.67	10.793
urls.100MB	5.072	10.893	25.062	8.585	8.185	5.348
urls.200MB	10.601	21.641	64.836	16.921	16.245	11.697
words.100MB	21.449	20.357	125.384	34.182	9.193	13.313
words.200MB	45.311	42.147	367.687	71.788	17.361	27.09

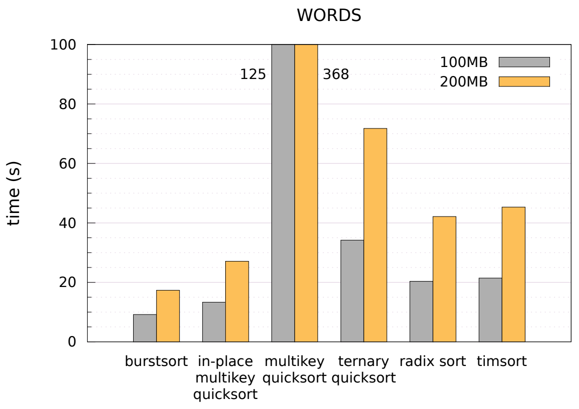
Table 2: Running times for each algorithm with different data sources

PERFORMANCE GRAPHS









RESULTS

The above results were achieved with the PyPy just-in-time compiler for Python on a quad-core 64-bit Intel Core i5 machine with 6 megabytes of cache and 8 gigabytes of RAM. The results should be taken with a grain of salt due to our use of a high-level language with a JIT compiler: non-transparent optimizations may have been performed at any step of the execution process. For a direct comparison, a lower-level language should be used.

Our choice of a fixed alphabet of 256 symbols for MSD radix sort definitely hurt its performance: less than half of the buckets allocated at each partitioning step are ever used to hold any strings with every other dataset besides WORDS.

While fast in theory, quicksort versions that were not sorting in-place were rather sluggish. Some of the peformance degradation could have been caused by $O(n \log n)$ memory requirement. Coupled with read/write access to non-contiguous memory areas the resulting cache-misses caused performance penalties.

It was interesting to note how much better the in-place algorithms (burstsort and in-place multikey quicksort) performed on the more demanding datasets. Despite being a high-level language, it appears there are still performance gains to be had in programming close to the hardware in Python.

REFERENCES

[1] T. Peters. [Python-Dev] Sorting. In *Python Developers Mailinglist*, 2002. Retrieved on 21 Feb 2013.

[2] R. Sinha. URL dataset http://www.cs.mu.oz.au/~rsinha/resources/data/sort.data.zip Retrieved Feb 2013

[3] P. Ferragina and G. Navarro. The Pizza & Chili Corpus, http://pizzachili.dcc.uchile.cl/texts.html, 2005 Retrieved on 11 Feb 2013.

[4] R. Sinha and A. Wirth. Engineering burstsort: Towards fast in-place string sorting. In *Experimental Algorithms*, pages 14–27, Springer, 2008. IEEE, 2001.