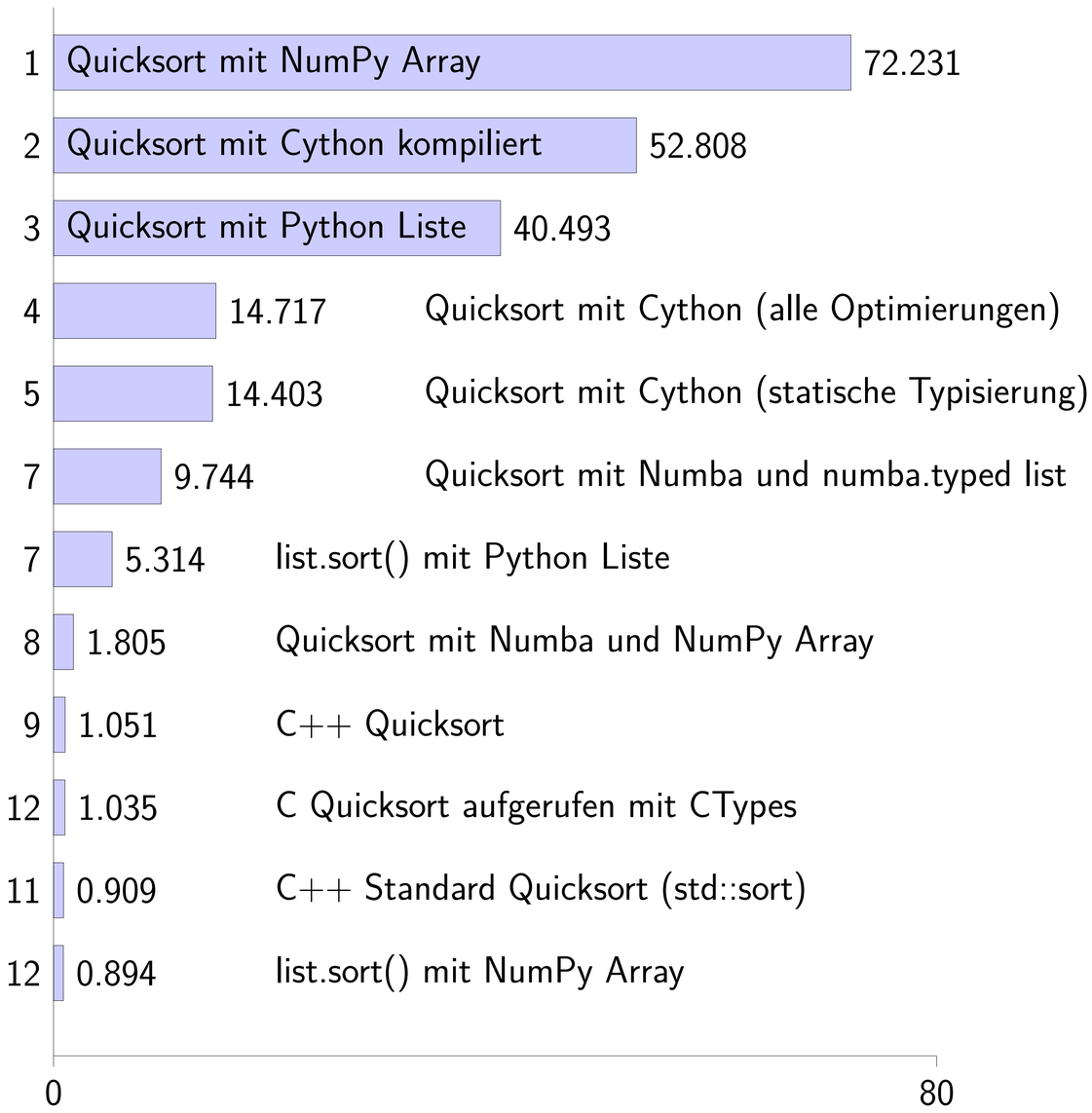
SUPERSONIC ALGORITHMS

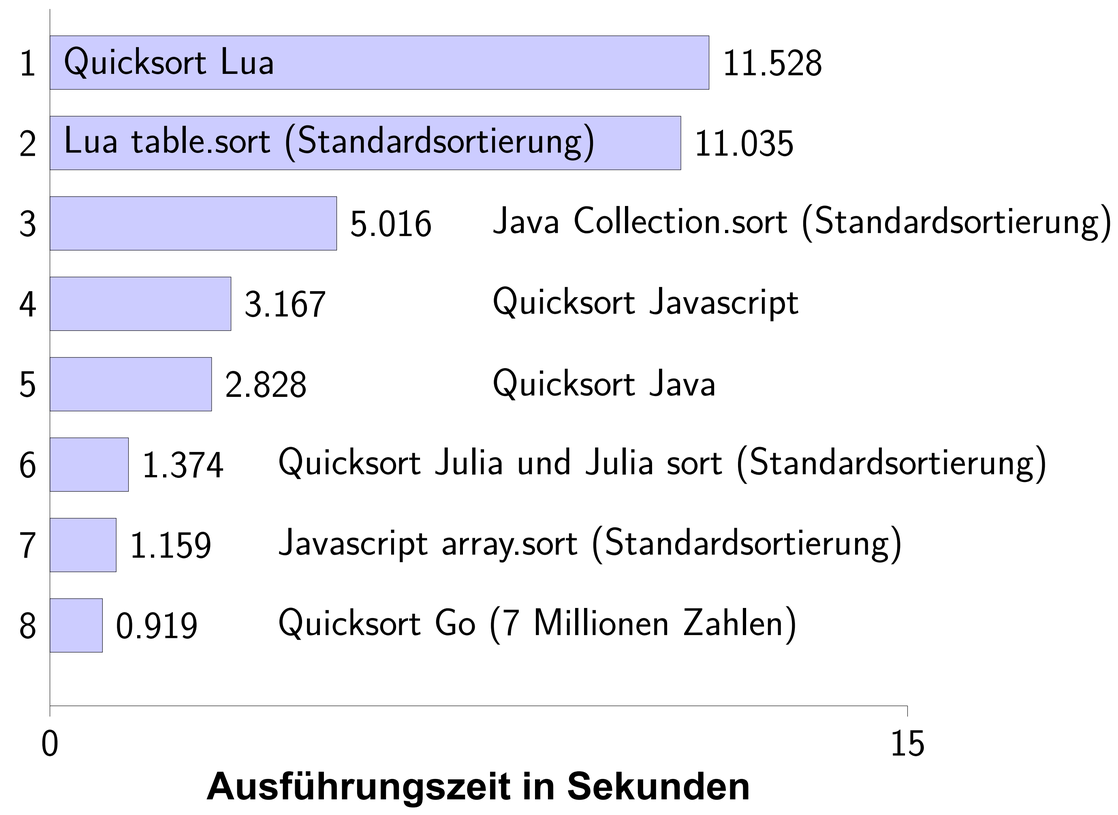
Anton Rodenwald (18), Schillerschule Hannover

Quicksort Implementationen in Python mit verschiedenen Optimierungen und Vergleich mit C++

Das Diagramm zeigt die Ausführungzungszeit meiner verschiedenen Implementationen des Sortieralgorithmus Quicksort in den Programmiersprachen Python und C++, die ich eine zuvor generierte, unsortierte Liste aus 10 Millionen zufälligen Ganzzahlen sortieren ließ. Zusätzlich sind noch die spracheigenen Sortierfunktionen als Vergleichswerte aufgeführt. Meine Erste Version (3) brauchte ca. 41 Sekunden zur Sortierung. Überraschenderweise brauchten einige Versionen mit Cython und NumPy (1, 2) teils länger, was ich mir nicht erklären kann. Als ich dann Cython zur statischen Typisierung nutzte, zeigte sich ein guter Zuwachs an Performance (5), wobei weitere Optimierungen mit Cython (4) keinen Effekt hatten. Bei Versionen mit Numba (6, 8) gab es einen noch größeren Zuwachs an Geschwindigkeit, der sich durch Nutzung eines NumPy Arrays verstärken ließ (8). Meine Version mit Numba und NumPy (8) war sogar schneller als die normale Sortierfunktion für Python Listen (7). Einzig schneller waren nur einige C/C++ Versionen (9, 10, 11) und die standardmäßige Sortierfunktion für NumPy Arrays, was ich so erwartet hatte. Es zeigt sich, dass NumPy eine genauso schnelle Sortierung wie C++ ermöglicht. Dies überraschte mich sehr, weil es meiner Hypothese und den Foren-Beiträgen, die ich gelesen hatte, komplett widersprach.



Quicksort Implementationen in weiteren Sprachen im Vergleich



Auch interessierten mich Implementationen von Quicksort in anderen Sprachen, wo ich aber aufgrund meines limitierten Wissens in diesen Sprachen keine Optimierungen vornahm. Es fällt direkt auf, dass Lua (1, 2) sowohl mit der Quicksort als auch mit der standardmäßigen Sortierung im Vergleich sehr schlecht abschneidet. Darauf folgt die standardmäßige Sortierung in Java (3). Diese ist überraschenderweise langsamer als meine Quicksort Implementation in Java (5). Zwischen diesen beiden liegt meine Quicksort in Javascript (4). Darauf folgt dann Julia (6), wo meine eigene Quicksort fast genauso schnell war wie die standardmäßige Sortierfunktion. Einzig schneller sind nur die standardmäßige Sortierfunktion in Javascript (7) und meine Quicksort Implementation in Go (8), wobei die Versionen in Go nur 7 Millionen Zahlen sortiert aufgrund von Implementationsschwierigkeiten.

C++ Radixsort Implementationen

Um eine niedrig möglichste Zeit zu erreichen schaute ich mir noch die Radixsort als anderen Sortieralgorithmus an, die allerdings nicht direkt mit der Quicksort vergleichbar ist. Implementationen dieser brauchten zuerst 2.269 Sekunden (1), doch ich konnte die Performance noch steigern (2, 3) und die Quicksort überholen. Dies zeigte mir, dass korrekte Implementation und Wahl des Algorithmuvon eine wichtige Rolle spielen. Auf die resultierende Frage, wieso die Radixsort nicht häufiger genutzt wird, fand ich bisher noch keine Antwort.