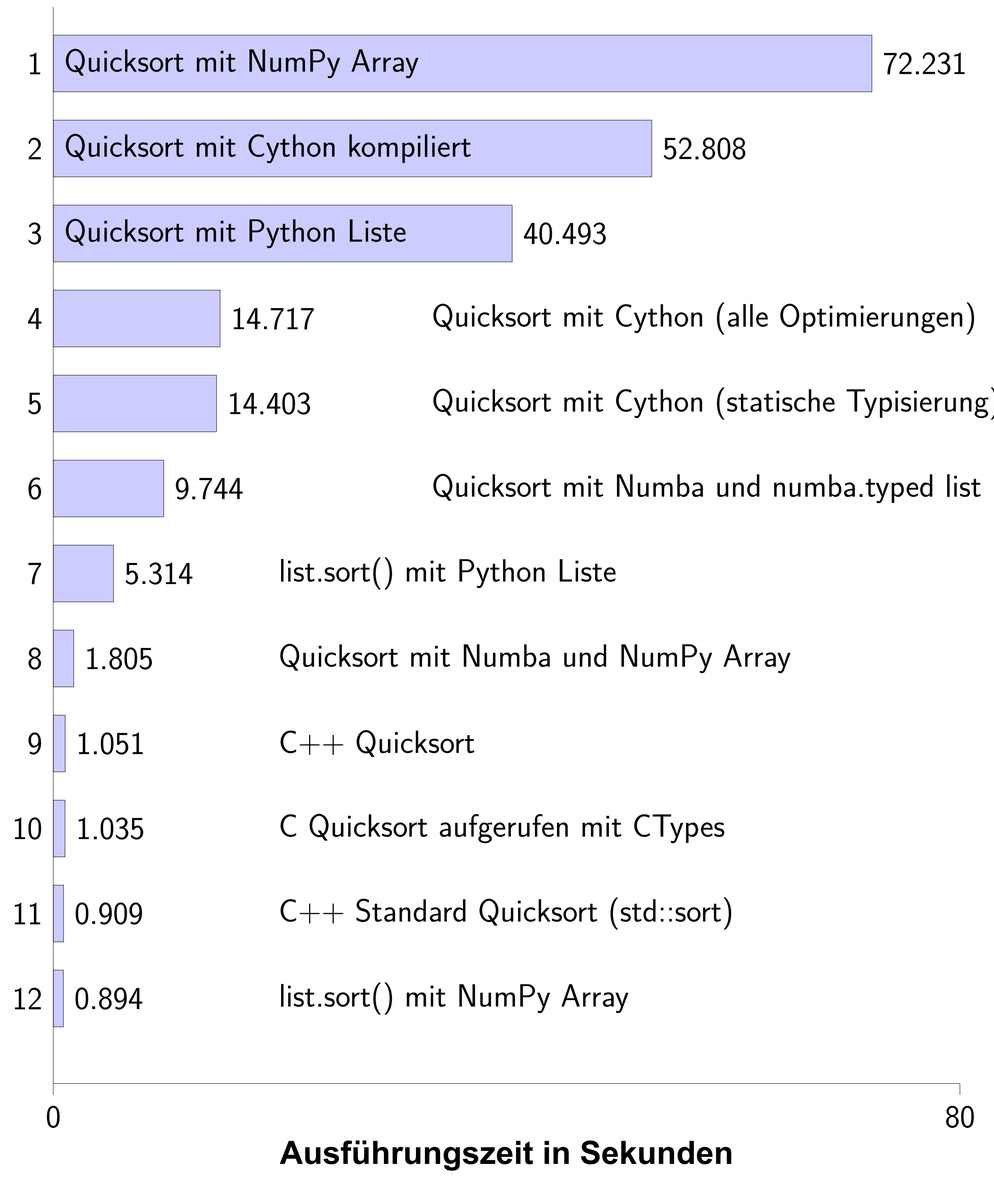
Supersonic Algorithms

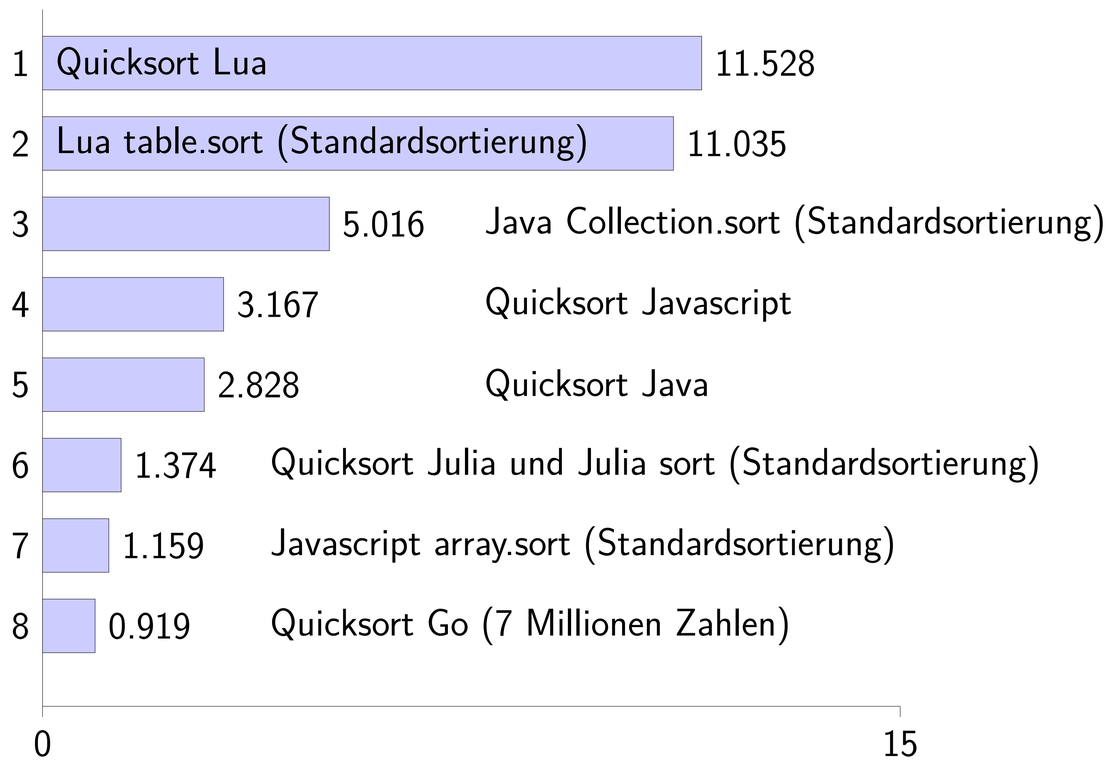
Anton Rodenwald (18), Schillerschule Hannover

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

- Ausführungzungszeit meiner verschiedenen Implementationen des Sortieralgorithmus Quicksort
- Programme sortierten 10 Millionen vorher generierte, zufällige Ganzzahlen
- Erste Version (3) ca. 41 Sekunden
- Optimierungsversuche überraschendeweise teils langsamer (1, 2)
- Optimierungen mit Cython (4, 5) bringen deutlichen Performance Boost
- Versionen mit Numba sogar nochmals schneller (6, 8)
- Eine Version (8) sogar schneller als list.sort(), der standardmäßigen Python Sortierfunktion
- Versionen mit C++, CTypes und NumPy Arrays nochmals eine Stufe darüber (9 - 12)
- Version mit CTypes benötigt allerdings noch zusätzlich Konvertierungszeit von ca. 2 Sekunden
- Python ähnlich schnell wie C++, was ich so nicht erwartet hatte



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.