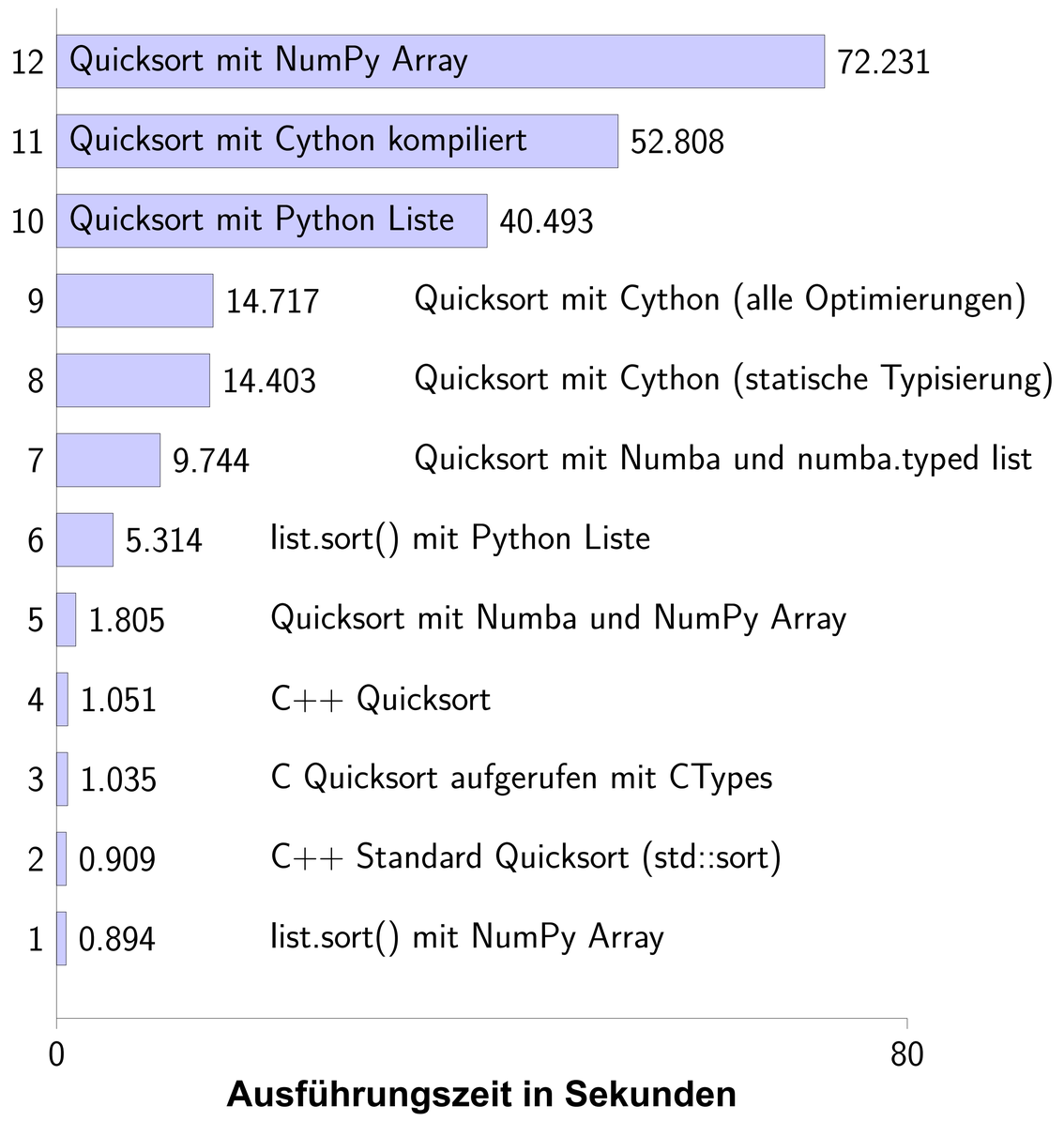
Supersonic Algorithms

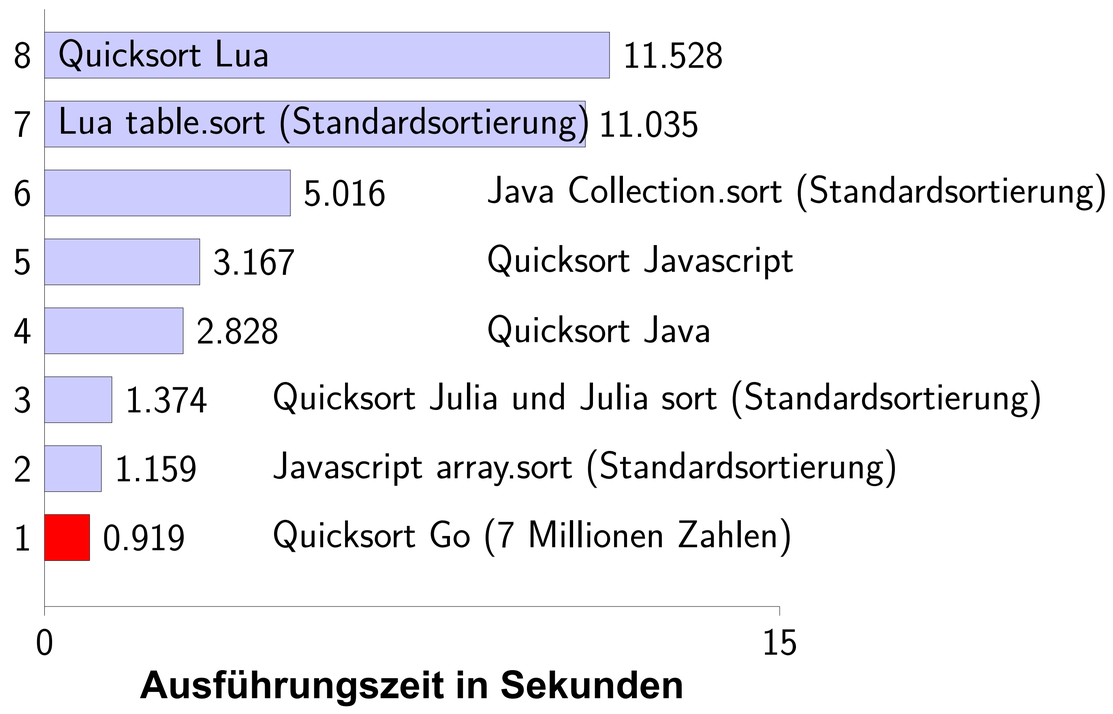
Anton Rodenwald (18), Schillerschule Hannover

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

- Ausführungszeit meiner verschiedenen Implementationen des Sortieralgorithmus Quicksort in Python und C++
- Programme sortierten 10 Millionen vorher generierte, zufällige, unsortierte Ganzzahlen in aufsteigende Reihenfolge
- Erste Version aus dem Informatikunterricht (10) ca. 41 Sekunden
- Optimierungsversuche überraschendeweise aus unbekannten Gründen teils langsamer (12, 11),
- Optimierungen mit Cython (9, 8) bringen deutlichen Performance Boost
- Versionen mit Numba sogar nochmals schneller (5, 7)
- Eine Version (5) sogar schneller als list.sort(), die standardmäßige Python Sortierfunktion (6)
- Versionen mit C++, CTypes und NumPy Arrays nochmals eine Stufe darüber (1 - 4)
- 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 Implementierte ich Quicksort in anderen Programmiersprachen, um deren Geschwindigkeit zu erforschen
- Nicht besonders optimiert, da ich kaum Erfahrung in diesen Sprachen habe
- Lua relativ langsam (8, 7)
- Bereitgestellte Sortierfunktion von Java (6) langsamer als eigene Quicksort Implementation in Java (4), was daran liegen könnte, dass diese vllt. nicht nur auf Zahlen ausgelegt ist
- Javascript Quicksort im Mittelfeld (5)
- Standardsortierung und eigene Implementation in Julia ähnlich schnell (3)
- Javascript array.sort am schnellsten (2)
- Go Quicksort Implementation (1) zwar über Javascript, sortierte aber nur 7 Millionen Zahlen aufgrund von Implementationsproblemen

C++ Radixsort Implementationen

- Versuch der möglichst schnellen Sortierung mit einem anderem Algorithmus (Radixsort)
- Nicht direkt mit der Quicksort vergleichbar, da anderer Algorithmus
- Erste Version noch relativ langsam (3), da ich Implementationsfehler machte; Implementationen (1, 2) deutlich schneller
- AVX2 (“Advanced Vector Instructions”, spezielle Befehle für die CPU) als Optimierungsmöglichkeit genutzt