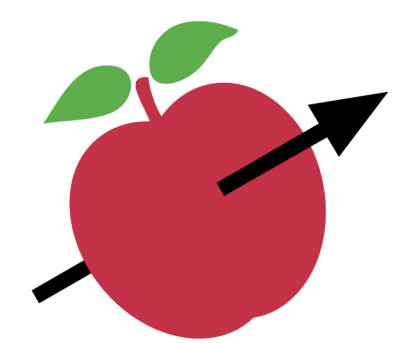


# SUPERSONIC ALGORITHMS

Anton Rodenwald (18), Schillerschule Hannover



## Ideenfindung und Forschungsfrage

- Sortialgorithmen für Zahlen im Informatikunterricht kennengelernt
- Ich fragte mich, wie man Zahlen am schnellsten sortieren kann
- Sortialgorithmen sind bereits bekannt und erforscht, deswegen Fokus auf Implementation bekannter Algorithmen
- Forschungsfrage: Wie lassen sich Programme (z. B. Sortialgorithmen) durch geschickte Implementation in ihrer Ausführung beschleunigen?
- Kein Vergleich von Sortialgorithmen, fast immer Quicksort verwendet
- Fokus auf die Programmiersprachen Python, C++ und C
- In Interquellen wurde C++ um ein vielfaches schneller als Python beschrieben
- Es war auch meine Vermutung, dass C++ deutlich schneller Zahlen sortiert

## Schwierigkeiten

- Meine größte Schwierigkeit war, dass ich noch nicht viel Erfahrung mit der Optimierung und dem wissenschaftlichen Arbeiten hatte
- Erstellung der Dokumentation mit  $\text{\LaTeX}$  teils frustrierend
- Mit einigen Bibliotheken und Sprachen Schwierigkeiten
- Mir war das testen mit 10 Millionen Zahlen in Go nicht möglich, da dort Rekursion nicht so gut funktioniert

## Ergebnisdiskussion

- Ein Ansatz zur Erklärung der Ergebnisse ist die Unterscheidung von interpretierten und kompilierten Programmiersprachen
- Bei interpretierten Sprachen führt ein spezielles Programm, der Interpreter, ein in Textform vorliegendes Programm aus
- Bei kompilierten Sprachen muss vor dem Ausführen der Quellcode von einem Compiler umgewandelt werden in ein ausführbares Programm
- Dieser Prozess kostet Zeit, aber erlaubt viele verschiedene Optimierungen
- Beim Interpreter ist dies bei Beginn der Ausführung alles noch nicht geschehen
- Python ist deswegen langsamer, weil während der Ausführung der Programmcode der Quelldatei noch zeitintensiv analysiert werden muss, bevor dieser ausgeführt werden kann
- Bei kompilierten Sprachen wie C++ ist dies bereits geschehen und der Programmcode wurde vom Compiler optimiert, weswegen diese Programme dann schneller sind
- Python ist nur dann schnell, wenn der Interpreter vorher kompilierte Programmteile nutzen kann, also wenn kaum Python Code interpretiert werden muss

## Material, Vorgehen und Methode

- Ich nutze meinen PC und Laptop zur Implementation
- Ich verwende die IDE Visual Studio Code und die Compiler und Interpreter für meine Programmiersprachen
- Ich implementierte anschließend die Quicksort auf verschiedene Arten und erhielt so viele Variationen zum testen
- Die Tests führte ich auf meinem Desktop PC durch (CPU ist der Ryzen 7 2700, 16 GB Arbeitsspeicher), der immer gleich ausgelastet war, um Vergleichbarkeit zu gewährleisten
- Die Zeit stoppte ich mit den Zeitfunktionen der Sprachen

## Quellcode meiner Timer Implementation in Python

```
import time

class Timer:
    timers = {}
    @staticmethod
    def start(name):
        Timer.timers[name] = time.time_ns()

    @staticmethod
    def stop(name):
        diff = time.time_ns() - Timer.timers[name]
        diffseconds = diff / (10 ** 9)
        print(name, ":", round(diffseconds, 5), "Seconds")
```

## Beantwortung der Forschungsfrage und Ausblick

- Die besten Möglichkeiten zur Optimierung in Python sind NumPy und Numba; Cython schlechter, CTypes komplex
- In C++ bietet AVX2 interessante Möglichkeiten der Optimierung
- Python kann ähnlich schnell wie C++ sein kann
- Insgesamt profitiert Python vor allem von schnellen C Bibliotheken
- Python ist nur dann schnell, wenn kein Python Code, sondern C Funktionen ausgeführt werden
- Weiter erforschen könnte man den Aufbau von Interpretern und Compilern (z. B. am Beispiel von Python, C++)
- Auch denkbar ist der vertiefte Vergleich von Quicksort und Radixsort zur Sortierung verschiedener Daten

