

Algorithmen und Datenstrukturen in C/C++

Prof. Dr. rer. nat. Peter Gerwinski

2. April 2020

Hardwarenahe Programmierung

<https://gitlab.cvh-server.de/pgerwinski/hp.git>

- 1 Einführung**
- 2 Einführung in C**
- 3 Bibliotheken**
- 4 Hardwarenahe Programmierung**
- 5 Algorithmen**
 - 5.1 Differentialgleichungen
 - 5.2 Rekursion
 - 5.3 Aufwandsabschätzungen
- 6 Objektorientierte Programmierung**
- 7 Datenstrukturen**
 - 7.1 Stack und FIFO
 - 7.2 Verkettete Listen
 - 7.3 Bäume

Hardwarenahe Programmierung

<https://gitlab.cvh-server.de/pgerwinski/hp.git>

- 1 Einführung
- 2 Einführung in C
- 3 Bibliotheken
- 4 Hardwarenahe Programmierung
- 5 **Algorithmen**
 - 5.1 Differentialgleichungen
 - 5.2 Rekursion
 - 5.3 Aufwandsabschätzungen
- 6 Objektorientierte Programmierung
- 7 **Datenstrukturen**
 - 7.1 Stack und FIFO
 - 7.2 Verkettete Listen
 - 7.3 Bäume

Algorithmen

Differentialgleichungen

- Pendel
- Basketball

Rekursion

- Türme von Hanoi

→ *Wie rechnet man das überhaupt aus?*

Aufwandsabschätzungen

- Selectionsort
- Bubblesort
- Quicksort

→ *Wie rechnet man das möglichst effizient aus?*

- ? möglichst schnell
- ? mit möglichst wenig Speicherplatzverbrauch
- ? unter Berücksichtigung gegebener Randbedingungen

Datenstrukturen

Stack und FIFO

- effizientes Anfügen und Entfernen vorne und hinten
- effizienter direkter Zugriff auf Elemente in der Mitte
- ineffizientes Einfügen und Entfernen in der Mitte

Verkettete Listen

- effizientes Einfügen und Entfernen in der Mitte
- ineffizienter direkter Zugriff auf Elemente in der Mitte

Bäume

- Kompromiß

→ *Wie speichert man das möglichst effizient?*

- ? möglichst schnell
- ? mit möglichst wenig Speicherplatzverbrauch
- ? unter Berücksichtigung gegebener Randbedingungen

Datenstrukturen

Structs und Objekte

- zusammengehörige Daten gemeinsam speichern

Stack und FIFO – Arrays

- effizientes Anfügen und Entfernen vorne und hinten
- effizienter direkter Zugriff auf Elemente in der Mitte
- ineffizientes Einfügen und Entfernen in der Mitte

Verkettete Listen – Structs mit Pointern

- effizientes Einfügen und Entfernen in der Mitte
- ineffizienter direkter Zugriff auf Elemente in der Mitte

Bäume – Structs mit Pointern

- Kompromiß

→ *Wie speichert man das möglichst effizient?*

- ? möglichst schnell
- ? mit möglichst wenig Speicherplatzverbrauch
- ? unter Berücksichtigung gegebener Randbedingungen

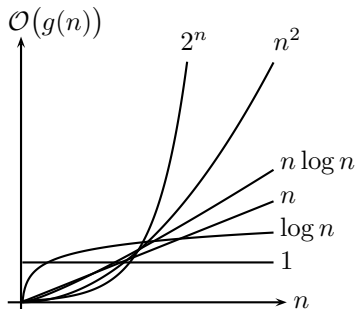
Aufwandsabschätzungen

- Türme von Hanoi: $\mathcal{O}(2^n)$

Für jede zusätzliche Scheibe
verdoppelt sich die Rechenzeit!

$$\rightarrow \frac{32,712 \text{ s} \cdot 2^{32}}{3600 \cdot 24 \cdot 365,25} \approx 4452 \text{ Jahre}$$

für 64 Scheiben



n : Eingabedaten

$g(n)$: Rechenzeit

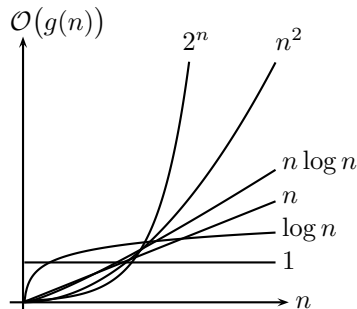
Aufwandsabschätzungen

- Türme von Hanoi: $\mathcal{O}(2^n)$
- Minimum suchen: $\mathcal{O}(n)$
- ... mit Schummeln: $\mathcal{O}(1)$
- Minimum an den Anfang tauschen, nächstes Minimum suchen:
→ Selectionsort: $\mathcal{O}(n^2)$
- Während Minimumsuche prüfen und abbrechen, falls schon sortiert
→ Bubblesort: $\mathcal{O}(n)$ bis $\mathcal{O}(n^2)$
- Rekursiv sortieren
→ Quicksort: $\mathcal{O}(n \log n)$ bis $\mathcal{O}(n^2)$

Faustregel:

Schachtelung der Schleifen zählen

x Schleifen → $\mathcal{O}(n^x)$

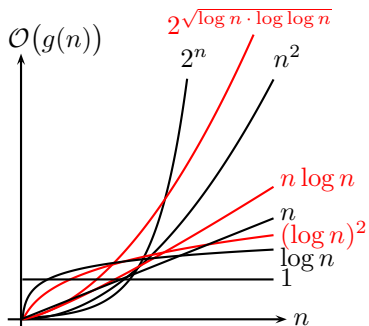


n : Eingabedaten

$g(n)$: Rechenzeit

Aufwandsabschätzungen

- Türme von Hanoi: $\mathcal{O}(2^n)$
- Minimum suchen: $\mathcal{O}(n)$
- ... mit Schummeln: $\mathcal{O}(1)$
- Minimum an den Anfang tauschen, nächstes Minimum suchen:
→ Selectionsort: $\mathcal{O}(n^2)$
- Während Minimumsuche prüfen und abbrechen, falls schon sortiert
→ Bubblesort: $\mathcal{O}(n)$ bis $\mathcal{O}(n^2)$
- Rekursiv sortieren
→ Quicksort: $\mathcal{O}(n \log n)$ bis $\mathcal{O}(n^2)$



n : Eingabedaten

$g(n)$: Rechenzeit

Faustregel:

Schachtelung der Schleifen zählen

x Schleifen → $\mathcal{O}(n^x)$

RSA: Schlüsselerzeugung (Berechnung von d): $\mathcal{O}((\log n)^2)$,

Ver- und Entschlüsselung (Exponentiation): $\mathcal{O}(n \log n)$,

Verschlüsselung brechen (Primfaktorzerlegung): $\mathcal{O}(2^{\sqrt{\log n \cdot \log \log n}})$

Algorithmen und Datenstrukturen in C/C++

<https://gitlab.cvh-server.de/pgerwinski/ad.git>

1 Einführung

2 ...

Rasterung

$\mathcal{O}(n \log n)$

array

struct

Wegfindung

CORDIC

FFT

B-Baum

Verschlüsseln

pointer

verkettete Liste

Datenbanken

Datenkompression

digitale Signatur

Hash-Tabelle

Prüfsumme

kryptographische Hash-Funktion

Algorithmen und Datenstrukturen in C/C++

<https://gitlab.cvh-server.de/pgerwinski/ad.git>

- 1 Einführung
- 2 Datenorganisation
- 3 Optimierung
- 4 Hardwarenahe Algorithmen
- 5 Datenkodierung
- 6 Numerik



Änderungen
vorbehalten

Übungsaufgabe: Verkettete Listen

Schreiben Sie eine Funktion, die eine verkettete Liste in die umgekehrte Reihenfolge bringt.

- (a) Die Funktion soll möglichst schnell arbeiten.
Auf den Speicherverbrauch kommt es nicht an.
- (b) Die Funktion soll möglichst wenig Speicher verbrauchen.
Auf die Rechenzeit kommt es nicht an.
- (c) Wandeln Sie die einfach verkettete Liste in eine doppelt verkettete Liste um.

Geben Sie für alle Funktionen das Landau-Symbol für die Laufzeit und für den Speicherverbrauch an.

Algorithmen und Datenstrukturen in C/C++

<https://gitlab.cvh-server.de/pgerwinski/ad.git>

- 1 Einführung
 - i Gesellschaftliche Auswirkungen von Algorithmen und Datenstrukturen
- 2 Datenorganisation
- 3 Optimierung
- 4 Hardwarenahe Algorithmen
- 5 Datenkodierung
- 6 Numerik



Änderungen
vorbehalten

i Gesellschaftliche Auswirkungen von Algorithmen und Datenstrukturen

... am Beispiel von *Upload-Filtern*

Vorab:

- Vorkenntnisse?
- Eigener Standpunkt?

→ anonyme Erhebung

Wie stehen Sie zur EU-Urheberrechtsreform?

- ☐ Ich habe mich noch nicht damit befaßt.
- ☐ Ich habe mich damit befaßt, vertrete dazu aber keine eigene Meinung.
- ☐ Ich unterstütze die EU-Urheberrechtsreform vollumfänglich in ihrer derzeitigen Form – inklusive Artikel 11 und 13. Diese sollte vom EU-Parlament angenommen werden.
- ☐ Ich halte die Artikel 11 und 13 für falsch, befürworte jedoch die Annahme der EU-Urheberrechtsreform notfalls auch mit diesen Artikeln.
- ☐ Ich halte die Artikel 11 und 13 für falsch. Sofern es nicht gelingt, diese Artikel herauszunehmen, sollte die EU-Urheberrechtsreform abgelehnt werden.
- ☐ Ich halte die EU-Urheberrechtsreform vollumfänglich für falsch. Diese sollte vom EU-Parlament abgelehnt werden.

Persönliche Meinung des Dozenten

Wie stehen Sie zur EU-Urheberrechtsreform?

- ☐ Ich habe mich noch nicht damit befaßt.
- ☐ Ich habe mich damit befaßt, vertrete dazu aber keine eigene Meinung.
- ☐ Ich unterstütze die EU-Urheberrechtsreform vollumfänglich in ihrer derzeitigen Form – inklusive Artikel 11 und 13. Diese sollte vom EU-Parlament angenommen werden.
- ☐ Ich halte die Artikel 11 und 13 für falsch, befürworte jedoch die Annahme der EU-Urheberrechtsreform notfalls auch mit diesen Artikeln.
- ☒ Ich halte die Artikel 11 und 13 für falsch. Sofern es nicht gelingt, diese Artikel herauszunehmen, sollte die EU-Urheberrechtsreform abgelehnt werden.
- ☐ Ich halte die EU-Urheberrechtsreform vollumfänglich für falsch. Diese sollte vom EU-Parlament abgelehnt werden.

Persönliche Meinung des Dozenten und der Parteien

Wie stehen Sie zur EU-Urheberrechtsreform?

- ☐ Ich habe mich noch nicht damit befaßt.
- ☐ Ich habe mich damit befaßt, vertrete dazu aber keine eigene Meinung.
- ☐ Ich unterstütze die EU-Urheberrechtsreform vollumfänglich in ihrer derzeitigen Form – inklusive Artikel 11 und 13. Diese sollte vom EU-Parlament angenommen werden.
- ☐ Ich halte die Artikel 11 und 13 für falsch, befürworte jedoch die Annahme der EU-Urheberrechtsreform notfalls auch mit diesen Artikeln.
- ☒ Ich halte die Artikel 11 und 13 für falsch. Sofern es nicht gelingt, diese Artikel herauszunehmen, sollte die EU-Urheberrechtsreform abgelehnt werden.
- ☐ Ich halte die EU-Urheberrechtsreform vollumfänglich für falsch. Diese sollte vom EU-Parlament abgelehnt werden.

Grüne
CDU/CSU

Grüne
SPD
Piraten
FDP
AFD
Die PARTEI
Die Linke