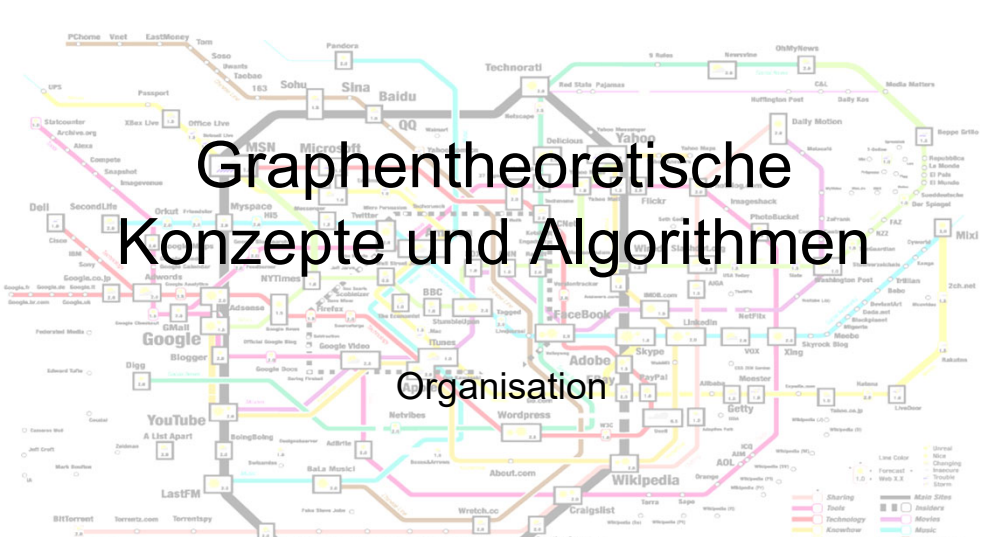


Graphentheoretische Konzepte und Algorithmen

Organisation

1

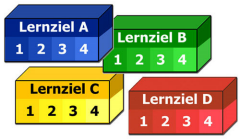
Informatik

Theoretische Informatik	Technische Informatik	Konstruktionslehre der Informatik	Praktische Informatik	Anwendungen der Informatik
- Formale Sprachen - Abstraktions-theorie etc.	- Rechner-technologie - Netze etc.	- Methoden u. Werkzeuge - System- u. Anwendungs-komponenten - Qualitäts-sicherung etc.	- Betriebs-systeme - Datenbank-systeme - Kommu-nikations-systeme etc.	- Kaufmännische Informations-systeme - Technische Informations-systeme etc.
Grundlagen	*Hardware*	*Methodenlehre*	*Basissoftware*	*Anwendungen*

Abb2 in Informatik-Spektrum, Heft 25/1, S. 39-51, 2002: Erich Ortner, *Sprachingenieurwesen, Empfehlung zur inhaltlichen Weiterentwicklung der (Wirtschafts-)Informatik*

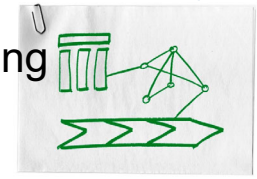
2

Lernziele



- Kennen lernen von in praktischen Anwendung erfolgreichen graphentheoretischen Modellierungs-paradigmen und Formalismen
- Kenntnis und Verständnis der grundlegenden Konzepte, Formalismen und Notationen sowie der wichtigsten Algorithmen
- Fähigkeit zum **eigenständigen** Modellieren und Lösen von praxisorientierten Problemen mit graphentheoretischen Methoden
- Fähigkeit zum **eigenständigen** Modellieren, einfachen Analyse und einem Redesign von nebenläufigen Prozessen mittels Petri-Netzen

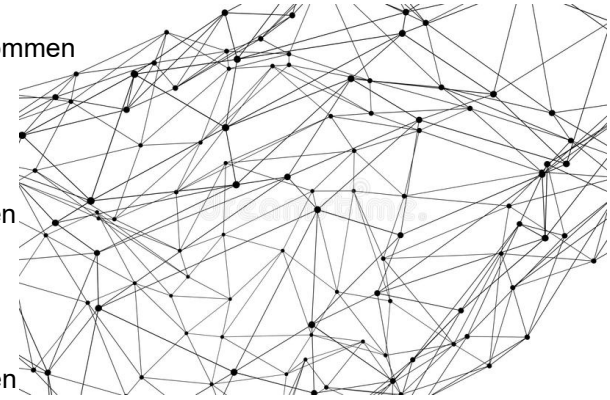
Organisation der Veranstaltung



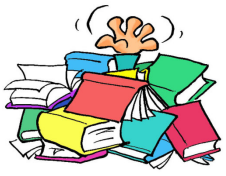
- www.informatik.haw-hamburg.de/~klauck/graphkoal.html
- 4 Stunden (2 Viertel) pro Woche
  - 4 Stunden Vorlesung
    - Raum NB 01.10 Montag 08^{15}-11^{30}
  - 4 Stunden (2 Viertel) Praktikum
    - Raum 1101a Dienstag 08^{15}-11^{30}
  - Prüfungsform: Referat (Terminvergabe am **07.10**!)
- Wie üblich
  - Vorlesung (20%) zur *tieferen* Vermittlung der *theoretischen* Grundlagen (mittels Fragen und Beispielaufgaben)
  - Praktikum (6,67%) zur betreuten praktischen Erfahrung
  - Eigenstudium (73,33%) zur Vorbereitung der Vorlesung, des Praktikums und dem Einüben der Kompetenzen

Inhalt der Vorlesung

1. **Grundbegriffe über Graphen**
   Gerichtet, zusammenhängend und gespeichert
2. **Optimale Wege**
   Günstig von A nach B kommen
3. **Bäume und Wälder**
   Spezialisierte Graphen
4. **Flussprobleme**
   Den Durchsatz optimieren
5. **Tourenprobleme**
   Briefe optimal austragen
6. **Petri Netze**
   Nebenläufigkeit verstehen



Basis-Literatur



- C. Klauck, C. Maas:
  Graphentheorie und Operations Research für Studierende der Informatik,
  HAW Hamburg,
- S.O. Krumke, H. Noltemeier:
  Graphentheoretische Konzepte und Algorithmen,
  Teubner,

- J. Clark, D.A. Holton: *Graphentheorie*, Spektrum,
- R. Diestel: *Graphentheorie*, Springer-Verlag,
- A. Habel: *Graphersetzungssysteme*, Universität Oldenburg,
- H.J. Kreowski: *Algorithmen auf Graphen*, Universität Bremen,
- J. Padberg: *Petrinetze*, HAW Hamburg,
- V. Turau: *Algorithmische Graphentheorie*, Oldenbourg,
- L. Volkmann: *Graphen an allen Ecken und Kanten*, RWTH Aachen,
- K. Reich: *Konstruktivistische Didaktik*, Beltz,

Hinweis zu den Folien

- Die Folien sind **kein vollständiges Skript** und genügen normalerweise nicht zur Prüfungsvorbereitung oder als Nachschlagewerk!
- Sie sollten sich deshalb auf jeden Fall zumindest mit der aufgeführten **Basis-Literatur beschäftigen** und sich von Zeit zu Zeit auch **weiterführende Literatur** und aktuelle **Zeitschriftenartikel anschauen**.
- **Hinweis**: Diese Folien sind zum großen Teil aus Folien/Skripten anderer Kollegen (auch anderer Hochschulen) zusammengestellt! Sie dürfen nur für den eigenen Gebrauch genutzt werden, also nicht an Dritte weiter gegeben werden oder in einem Netzwerk zugänglich gemacht werden.

Graphentheoretische Konzepte und Algorithmen

Praktikum / PVL

http://users.informatik.haw-hamburg.de/~klauck/prakt_pvl.html

Praktikum / PVL

- Gruppenaufteilung: **2 Studierende** in einem Team (**Meldung bis zum 01.10.19 16^{00} Uhr!**)
- **PVL-Bedingungen**
  - **Baut stark auf vorbereitende Arbeit** auf!
  - **Empfehlung**: **vor** dem Praktikumstermin: Ziele festlegen, Arbeitsplan erstellen etc.; das Praktikum als Präsentation/Anwendung nutzen
  - **Details** siehe Aufgabenstellungen!
    - Bis **Donnerstag Abend 20^{00} Uhr vor** dem Praktikumstermin: Entwurf zusenden.
    - Am Tag **vor** dem Praktikumstermin bis **20^{00} Uhr** : aktuellen Stand (als *.zip) zusenden.
    - **Während des Praktikumstermins** : Befragung / Programmieraufgaben
    - **Während des Praktikumstermins**: Disputation indirekt über Referate.
    - Die Frage ist **nicht ob** die Anforderungen erfüllt sind, **sondern wie**!
    - Am **selben Tag bis 20^{00} Uhr**: Abgabe der geforderten Unterlagen (digital!)
  - Anwesenheitspflicht (**gesamte Praktikumszeit**!)
  - Erfolgreiche Bearbeitung **aller** Aufgaben
  - Einhaltung **aller** Termine

Praktikumsaufgaben

- **Programmiersprache**
  Erlang/OTP
- **Teams** (je zwei) **verantwortlich für den gesamten Code** der Aufgabe: Architektur und Programmcode müssen gut (frei) erklärbar sein.
- 4 **Praktikumsaufgaben (2 für die PVL, 2 für das Referat):**
  - Abstraktion: ADT Graph (**bereits online!**)
  - Optimale Wege: manchmal ist es der kleine Unterschied
  - Flussprobleme: den Durchsatz optimieren
  - Tourenprobleme: mal einfach gehalten
- **Übernahme von Code Dritter**: Ist **nicht zulässig**! (auch nicht von anderen Teams, mit denen man zusammen gearbeitet hat, weder in Wortlaut noch dem Sinn nach!)
- **Ein nicht beachten der Informationen geht zu Ihren Lasten!!**

Dokumentationskopf

Details siehe Vorlage

Team: <Teamnummer>, <Namen der Teammitglieder>

Aufgabenaufteilung:

1. <Aufgaben, für die Teammitglied 1 verantwortlich ist>
2. <Aufgaben, für die Teammitglied 2 verantwortlich ist>

Quellenangaben: <Angabe aller genutzten Quellen>

Bearbeitungszeitraum: <Datum und Dauer der Bearbeitung an der Aufgabe von allen Teammitgliedern>

Aktueller Stand: <Welche Teile der Software sind fertig inklusive Tests, welche sind fertig, aber noch nicht getestet, welche müssen noch implementiert werden>

Änderungen in der Skizze: <Vor dem Praktikum auszufüllen: Welche Änderungen sind bzgl. der Vorabskizze vorgenommen worden.>

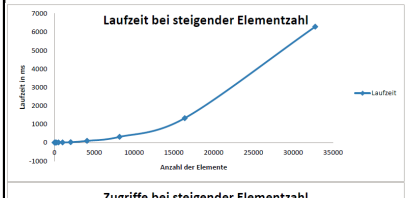
Entwurf: <Entwurfsskizze nach den Vorgaben gemäß Aufgabenstellung.>

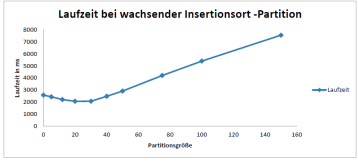
Ergebnis des Prozesses des Definierens von Architektur, Komponenten, Schnittstellen, Abstrakten Datentypen und anderen Charakteristika eines Systems oder einer Komponente. Dient als einziges Dokument bei der Implementierung!

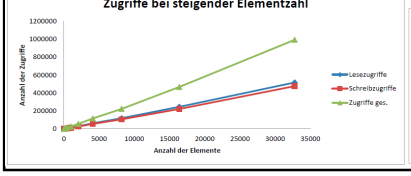
Messergebnisse

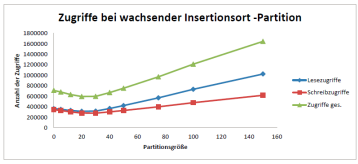
5000 MedianOfS	random	1	30	547	105376	532	16430	103107	107863
5000 Random	random	1	30	511	110252	807	15331	106243	119665
5000 Rechts	random	2	30	351	108091	340	10555	103251	116726
5000 Links	random	2	30	493	110729	501	14819	105986	121424
5000 MedianOfS	random	2	30	339	106158	339	10173	102972	110553
5000 Random	random	2	30	353	111436	372	10590	106783	117969
5000 Rechts	random	3	30	336	108134	344	10082	102754	113425
5000 Links	random	3	30	488	110452	504	14655	106814	115008
5000 MedianOfS	random	3	30	304	105768	302	9137	102921	112643
5000 Random	random	3	30	312	111666	309	9381	107682	117012
5000 Rechts	random	4	30	339	108347	343	10170	103557	116172
5000 Links	random	4	30	487	111052	482	14617	107002	118536

Zeitmessungen	Zeit in ns					
1	16418690	5728651984		101955974	646516	663640442
2	295984	7674994		51982052	295216	252765694
3	129694	5513707		20277634	305553	221819676
4	60133	4334657		14212454	192411	205394459
5	56990	3823215		[illegible]	196183	190127269
6	50844	4144063		[illegible]	185357	314059545
7	57130	4815231		[illegible]	185148	318884774
8	61110	5979334		[illegible]	206309	333303048
9	59853	5059114		[illegible]	12921726	154631128
10	61879	4484955		12243782	151275	131882667
Durchschnitt	1725230,7	577448125		30686217,7	1528569,4	278650870









Graphentheoretische Konzepte und Algorithmen

Referat

http://users.informatik.haw-hamburg.de/~klauck/referat.html

Prüfungsform Referat

- 10. Referat (R)
  Ein Referat ist ein Vortrag über 15 bis 45 Minuten Dauer (hier 15 Minuten) anhand einer selbst gefertigten schriftlichen Ausarbeitung (hier gemäß Vorgabe aus Praktikumsaufgabe). An das Referat schließt sich unter Führung einer Diskussionsleitung ein Gespräch an (hier ca. 5 Minuten). Das Referat soll in freien Formulierungen gehalten werden. Die bei dem Vortrag vorgestellten Präsentationen bzw. Grafiken sind dem Prüfer in schriftlicher oder elektronischer Form (hier digital als *.pdf, den Code als source-Datei) zu übergeben. In der zusätzlichen schriftlichen Ausarbeitung (hier digital als *.pdf), die dem Prüfer zu übergeben ist, sind die wichtigsten Ergebnisse zusammenzufassen (hier Beschreibung des Lösungsweges vom Entwurf bis zur Implementierung).

Erklärung zur schriftlichen Ausarbeitung des Referates

Hiermit erkläre ich, dass ich diese schriftliche Ausarbeitung meines Referates **selbstständig und ohne fremde Hilfe** verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe sowie die aus fremden Quellen (dazu zählen auch Internetquellen) direkt oder indirekt **übernommenen Gedanken oder Wortlaute als solche kenntlich gemacht habe**. Zudem erkläre ich, dass der zugehörige **Programmcode** von mir selbständig implementiert wurde **ohne diesen oder Teile davon von Dritten im Wortlaut oder dem Sinn nach übernommen zu haben**. Die Arbeit habe ich bisher keinem anderen Prüfungsamt in gleicher oder vergleichbarer Form vorgelegt. Sie wurde bisher nicht veröffentlicht.

Regularien für das Referat

1. **Ein Studierender** (Studierende eines Praktikums sind automatisch für die Referate gemeldet. Alle anderen Studierenden senden bitte per E-Mail ihre Meldung spätestens bis zum Termin zur Vergabe der Referatstermine an mich) Gemeinsame Referate, also auch gemeinsamer Programmcode, sind nicht zulässig.
2. Zu bearbeiten ist die **komplette zum Referatstermin gehörende Praktikumsaufgabe**.
3. In der schriftlichen Ausarbeitung ist der **Lösungsweg** vom **Entwurf bis zur Implementierung** (nach Prüfungsordnung die "wichtigsten Ergebnisse" genannt) aufzuführen. Sofern in der Aufgabenstellung z.B. Messungen, Interpretationen etc. gefordert sind, sind diese, wie auch z.B. der Entwurf, in die schriftlichen Ausarbeitung zu integrieren.
4. Studierende mit Referatstermin sind an diesem Tag ab Praktikumsbeginn bis zum Ende der Veranstaltung **anwesend**.
5. Die Vorträge finden gemäß dem im Internet veröffentlichtem Terminplan statt. Die vorgegebene Zeit (maximal 15 Minuten Vortrag + ca. 5 Minuten Diskussion) ist einzuhalten und darf nicht überschritten werden!

Regularien für das Referat

6. Damit der Vortrag und damit das gesamte Referat (also auch die schriftliche Ausarbeitung) bewertet werden kann, muss dieser **mindestens 12 Minuten** dauern. Bei geringerer Vortragszeit wird das Referat als nicht gehalten gewertet und insgesamt mit 0 Punkten bewertet.
7. Am Abend **vor dem Prüfungstag** sind alle geforderten Unterlagen bis 20:00 Uhr digital abzugeben! **Später abgegebene Unterlagen werden nicht berücksichtigt!** Sofern die Erklärung zur schriftlichen Ausarbeitung des Referates nicht eingescannt werden kann, kann diese unmittelbar vor dem zugehörigen Praktikumstermin in Papierform abgegeben werden.
8. Mehrfachabgaben sind unzulässig, es zählt die erste Abgabe.
9. Zur **Abgabe** gehören neben den in der Aufgabenstellung geforderten Unterlagen die Folien zum Vortrag, während des Referats erstellte Dokumente (sind ggf. nachzureichen) sowie die schriftliche Ausarbeitung. Komprimiert darf die gesamte Abgabe nicht 6MB überschreiten. Die in der Aufgabenstellung geforderten Vorabgaben (Entwurf/Vorabendabgabe) entfallen für das Referat.

Referat

- **1 Studierender** (**Meldung bis 07.10.19**)
- **Bedingungen**
  - **vorbereitende Arbeit**, keine Möglichkeit zur Korrektur!
  - **Empfehlung**: **vor** dem Referatstermin:
    Ziele festlegen, Arbeitsplan erstellen etc.; schriftliche Dokumente rechtzeitig erstellen, Programme ausgiebig testen, Präsentation üben etc.
  - Aufgabe zu dem eigenen Referatstermin gehört nicht zur eigenen PVL! PVL und Referat werden **unabhängig** voneinander bewertet!
  - Die Befragung / Besprechung entfällt. Die fachliche Auseinandersetzung wird über die Referate vorgenommen.
  - Die Bewertung wird aus organisatorischen Gründen erst zum Ende des Semesters (ca. letzte Vorlesungswoche) bekannt gegeben.

Prüfungsform Referat

Szenario für Referat:

- Projektleiter/Projektgruppe erteilt Mitarbeiter*in eine Aufgabe.
- Mitarbeiter*in präsentiert dem Projektleiter/der Projektgruppe die Ergebnisse.
- Um ähnliche Aufgaben in Zukunft schneller lösen zu können, wird der Lösungsweg (Entwurf bis Implementierung) in adäquatem Detail dargestellt.
- Die Ausarbeitung wird einer „Falldatenbasis“ zugeführt und muss entsprechend gestaltet sein.
- Die mündliche Präsentation kann auf Grund der zeitlichen Vorgabe nur auf die wichtigsten Elemente des Lösungsweges eingehen.

Bewertungsfragen zum Referat

Einfache Wertung

Vortrag

- Bezug zum Publikum (z.B. Regeln der freien Rede beachtet, flüssig, fast (frei), sinnvolle Pausen, geht auf Publikum ein)
- Aufbau – Inhalt – Gliederung (z.B. Zwingend, logisch, klar)
- Verständliche Aufbereitung des Stoffes für die Zuhörer (z.B. fachangemessen, Fachwörter werden ggf. erklärt, erklärende Einschübe/Hinweise)
- Umfang (z.B. sehr ausgewogen, alle wesentlichen Aspekte angesprochen)
- Eigentätigkeit – Fachwissen (z.B. sehr gut, umfangreich)
- Beantwortung von Fragen (z.B. Fachlich ausgezeichnete kurze Antworten)

Schriftliche Ausarbeitung

- Visueller Gesamteindruck bezüglich Darstellung (z.B. Genügt üblichen wissenschaftlichen Kriterien)
- leserfreundliche Form (z.B. variationsreich differenziert)
- Graphiken, Tabellen, Source-Code etc. (z.B. Gute visuelle Unterstützung, sind gut im Text erläutert)
- Einhalten von Vorgaben (z.B. Termine und Unterlagen vollständig)

Bewertungsfragen zum Referat

Schriftliche Ausarbeitung — Doppelte Wertung

- Aufgabenstellung (z.B. Sehr gut verstanden und alle wesentlichen Aspekte erfasst)
- Aufbau / Gliederung (z.B. logisch, klar erkennbar, systematisch folgerichtig)
- Qualität (z.B. wesentliche Informationen und Zusammenhänge)
- Quantität (z.B. angemessen)
- Fachwissen / Fachkompetenz (z.B. Wissen, Verstehen, Lösungsweg; Urteil klar erkennbar)
- Auswahl / Quellen / Materialien / Textstellen / Auswertung (z.B. informativ ansprechend funktional)
- Programmierung: - Aufgabe, - Strukturierung (Architektur/Algorithmus/ Datenstrukturen), - Implementierung (z.B. logisch, klar erkennbar systematisch folgerichtig)
- Programm (z.B. Erfüllt Vorgaben)
- Programmstruktur (z.B. Gute Strukturierung, sinnvolle Kommentierung)
- Kreativität (z.B. gute eigene Ideen) — Dreifache Wertung

Achtung: rot markierte Punkte müssen mit mehr als 0 Punkten bewertet werden, da sonst die gesamte Arbeit mit 0 Punkten bewertet wird.

Graphentheoretische Konzepte und Algorithmen

Entwurf

Wozu Aufgaben?

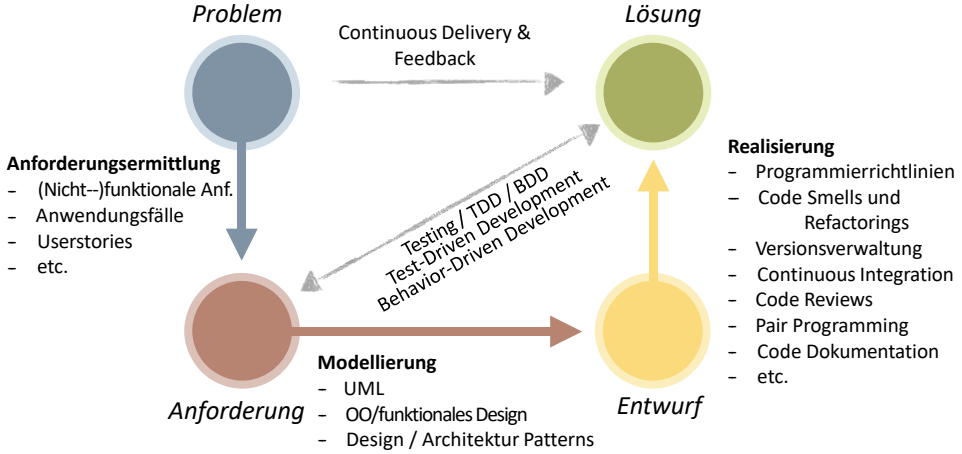
Machst du es richtig?
(Geschlossene Aufgabe: *Wie viel ist 49 * 51?*
Defizitperspektive)
oder
Wie machst du es?
(Offene Aufgabe: *Wie rechnest du 49 * 51?*
Entwicklungsperspektive)

Was will man erfahren, wenn eine Aufgabe gestellt wird?

Clean Code Developer

Sollte vielleicht **auf Entwurf verzichtet werden**, wenn letztlich in der Implementation die „Strukturwahrheit“ liegt? **Nein, sicher nicht.** Entwurf muss sein. **Ohne Planung gibt es keine Zielvorstellung.** Aber Entwurf und Implementation müssen dem „**D**on't **R**epeat **Y**ourself“-Prinzip gerecht werden. Deshalb sollten Entwurf und Implementation sich so wenig überlappen wie möglich. […] Wenn das der Fall ist, stellen sie keine Wiederholungen mehr dar, sondern beschreiben unterschiedliches. Das bedeutet: Entwurf/Architektur kümmert sich nicht um die Implementation und Implementation kümmert sich nicht um Architektur.
Und wo verläuft diese Trennlinie? Bei den so genannten Komponenten. Architekten kümmern sich nicht um den internen Aufbau von Komponenten. Für sie sind es Black Boxes, deren Klassenstruktur nicht architekturrelevant ist. Umgekehrt ist für einen Komponentenimplementierer die Architektur irrelevant. Was er zu implementieren hat, ergibt sich aus den Komponentenkontrakten, die seine Komponente importiert und exportiert. Einen größeren Zusammenhang muss er nicht kennen.
Die Aufgabe der Architektur ist es mithin, Software in Komponenten zu zerlegen, deren Abhängigkeiten zu definieren und Leistungen in Kontrakten zu beschreiben. […] Und die Aufgabe der Implementation ist es, die von der Architektur definierten Komponenten zu realisieren. Wie sie das tun, ist nicht architekturrelevant. Ihre innere Struktur ist für die Architektur unsichtbar.

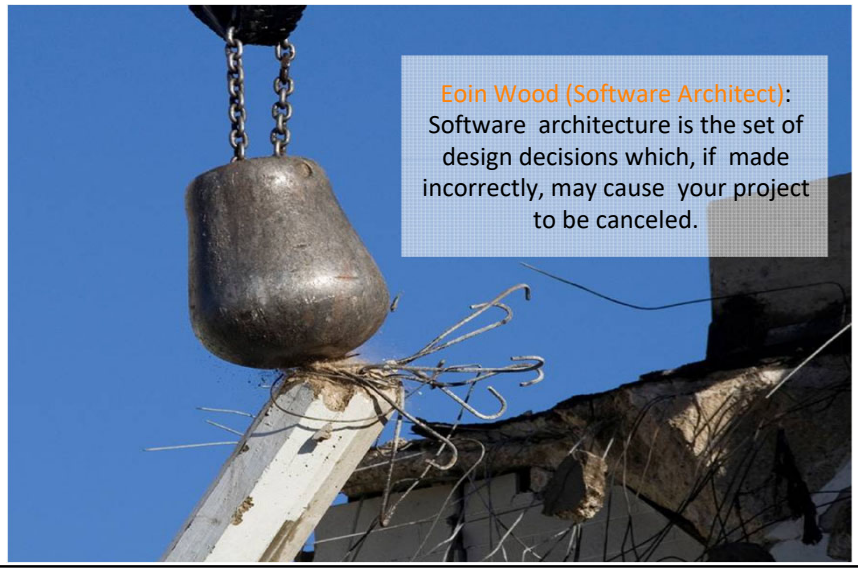
„Richtige“ Software braucht systematischen, strukturierten Aufbau



Kriterien für einen „guten“ Entwurf

- Korrektheit
  - Erfüllung aller Anforderungen
  - Wiedergabe aller Funktionen des Systemmodells
  - Sicherstellung der nichtfunktionalen Anforderungen
- Verständlichkeit & Präzision
  - Gute Dokumentation
- Anpassbarkeit
- Hohe Kohäsion innerhalb der Komponenten, d.h. Komponenten erleichtern Verständnis, Wartung und Anpassung.
- Schwache Kopplung zwischen den Komponenten, d.h. schwache Abhängigkeiten zwischen Komponenten
- Wiederverwendung, d.h. Ausnutzung von Gemeinsamkeiten zwischen Komponenten

Entwurf!



Eoin Wood (Software Architect):
Software architecture is the set of design decisions which, if made incorrectly, may cause your project to be canceled.

Entwurf!

„Wer zu früh anfängt zu codieren, braucht am Ende zu lange für das Projekt.“ (Whiscy-Syndrom, „Why isn’t Sam coding yet?“)

Grady Booch (Founder UML):
Architecture represents the significant design decisions that shape a system, where significant is measured by cost of change.

Entwurf!

GreenSoft
Green Software Engineering

ANSI/IEEE 1471-2000

The fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution.

Siemens AG ★★★★☆ 6.110 Bewertungen - Amberg

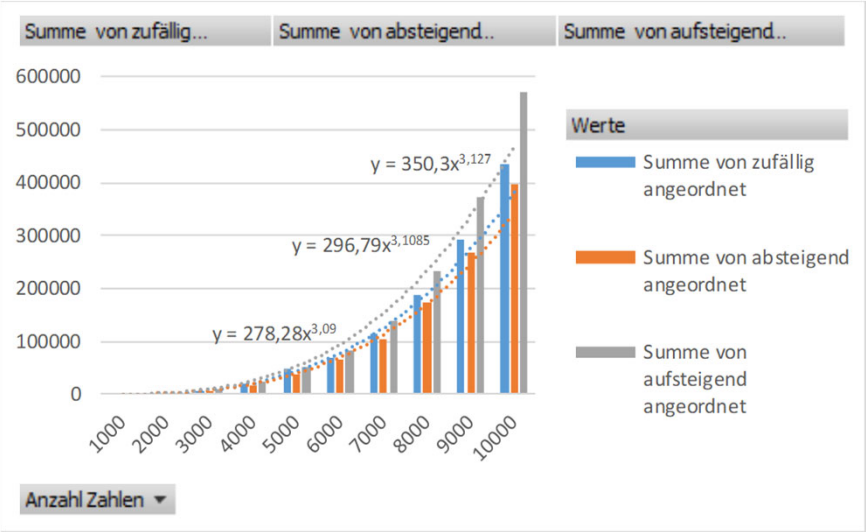
Echte Trendsetter in allen Sprachen.

Bevor unsere Software-Entwickler die erste Zeile eines Codes schreiben, machen sie sich damit vertraut, was der Kunde tatsächlich braucht. Erst dann werden verschiedene Wege ausprobiert und getestet, um die finale Lösung zu finden. In unseren Teams genießen Sie

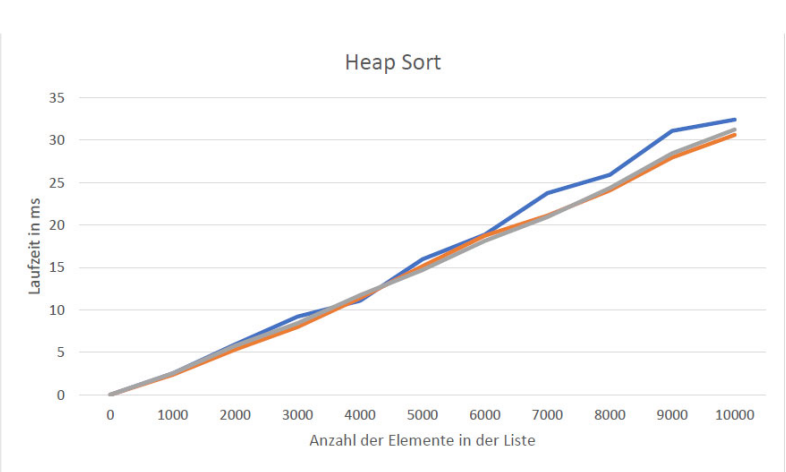
Entwurf: hier

- Der Entwurf enthält die detaillierte Beschreibung der inhaltlichen/funktionalen Aufgaben des Problems (**Lösungs-Algorithmen**). Jeder Schritt ist im Detail durchdacht, d.h. es gibt bzgl. der inhaltlichen/funktionalen Aufgabe keine offenen Fragen mehr!
- Der Entwurf enthält eine detaillierte Beschreibung der benötigten Daten in den jeweiligen Schritten des Lösungs-Algorithmus (**Lösungs-Datenstrukturen**). Es wird genau beschrieben, welche Daten in welcher Form in den jeweiligen Schritten benötigt werden. Die Beschreibung orientiert sich an den inhaltlichen/funktionalen Anforderungen, nicht jedoch an den technischen Begebenheiten.
- Der Entwurf enthält die Beschreibung aller wichtigen Schnittstellen und beschreibt damit auch im Detail die Architektur des Systems (**Lösungs-Architektur**).
- Der Entwurf ist **vollständig in seiner Beschreibung**, d.h. für die technische Umsetzung (Implementierung) ist der Entwurf als einziges Dokument ausreichend, insbesondere auch ausreichend für andere Teams! (Also **keine versteckten Informationen** in den Köpfen der Autor_innen des Entwurfs.)

Design Entscheidungen



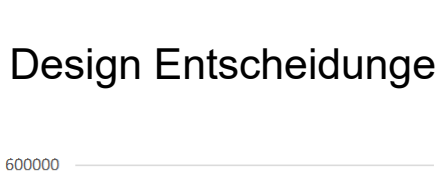
Design Entscheidungen



39

Design Entscheidungen

ca. 420000ms
10000 Elemente



X^3

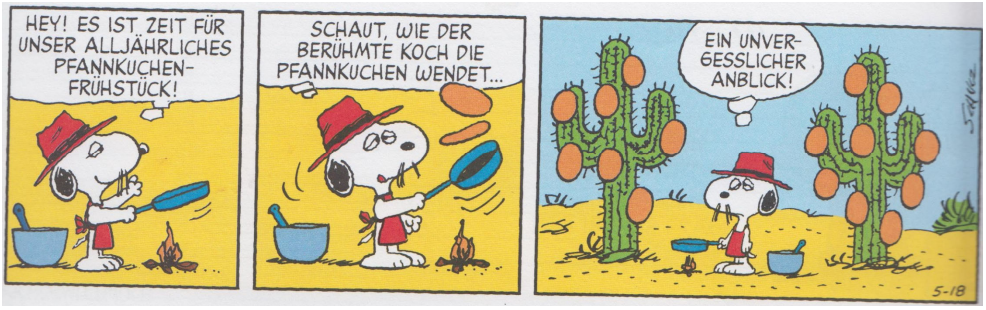
$X^1 \log X$

ca. 32ms
10000 Elemente

GreenSoft
Green Software Engineering

40

Beispiel Entwurf



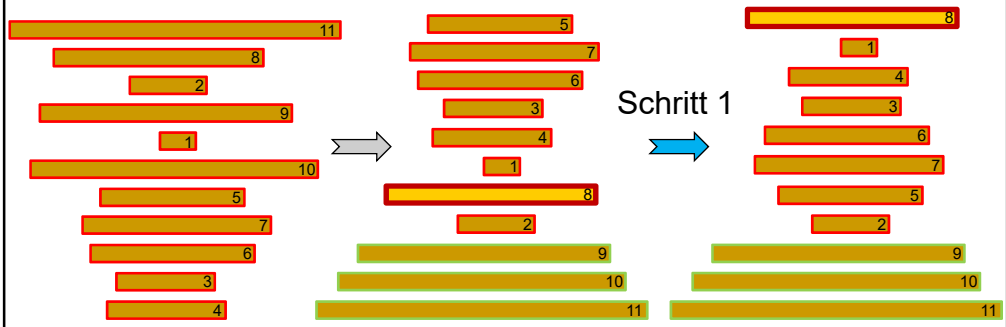
Aufgabenstellung

Ein Koch backt Pfannkuchen von unterschiedlicher Größe und legt sie als Stapel unsortiert auf einen Teller.

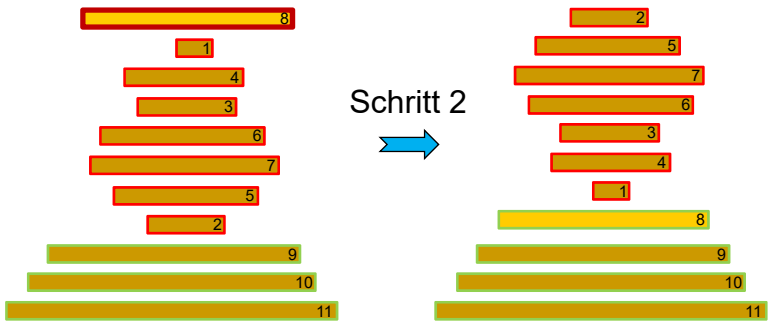
Der Kellner möchte die Pfannkuchen aber in einem der Größe nach sortierten Stapel (oben der kleinste Pfannkuchen) präsentieren.

Als Hilfsmittel für das Sortieren der Pfannkuchen hat der Kellner lediglich einen Pfannenwender zur Verfügung. Diesen kann er unter einen beliebigen Pfannkuchen schieben, um dann den darüber liegenden Stapel (komplett) umzudrehen.

Vorüberlegung Algorithmus



Vorüberlegung Algorithmus

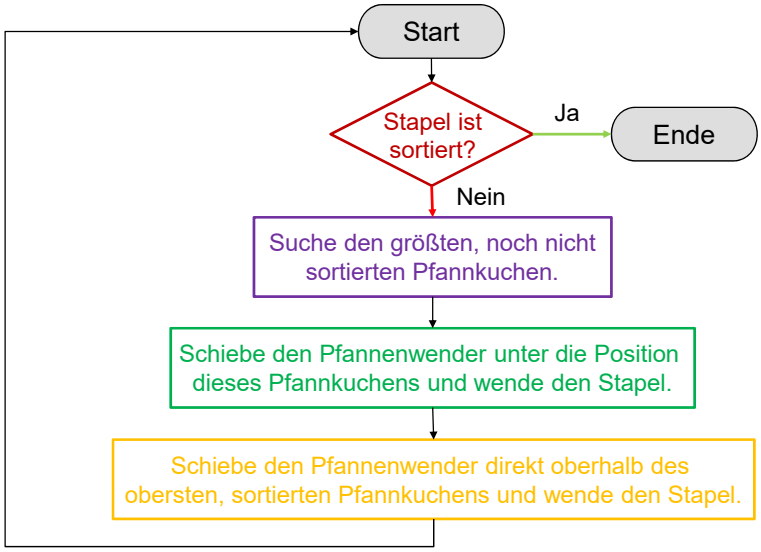


Algorithmus der Aufgabenstellung

Der Algorithmus orientiert sich an dem größtem, noch nicht sortiertem Pfannkuchen und einem bereits sortierten Teil (unterer Teil).

1. Solange der Stapel nicht sortiert ist
   a) Suche den größten, noch nicht sortierten Pfannkuchen.
      Der größte, noch nicht sortierte Pfannkuchen liegt irgendwo im unsortierten oberen Bereich des Stapels.
   b) Schiebe den Pfannenwender unter die Position dieses Pfannkuchens und wende den Stapel.
      Der größte, noch nicht sortierte Pfannkuchen liegt nun ganz oben.
   c) Schiebe den Pfannenwender direkt oberhalb des obersten, sortierten Pfannkuchens und wende den Stapel.
      Nun ist der größte, bisher nicht sortierte Pfannkuchen sortiert. Er ist der kleinste sortierte Pfannkuchen und im sortierten Teil der oberste Pfannkuchen.

Algorithmus der Aufgabenstellung



Strukturierung des Algorithmus

Datenstruktur (Anforderungen)

1. Die Datenstruktur muss eine Reihenfolge wiedergeben.
2. Man sollte schnell zumindest im unsortierten Bereich schnell durch die Struktur durchlaufen können, da man das dortige größte Element benötigt.
3. Beim sortierten Bereich wäre ein Zugriff auf das dort aktuell kleinste Element wichtig, um schnell das neu sortierte Element dort anhängen zu können.
4. Eine schnelle Umkehrung der Reihenfolge wäre gut, um diesen häufigen Schritt effizient durchführen zu können.
   (ACHTUNG: man darf nicht die Umkehrung auslassen und einfach das zu sortierende Element verschieben. Die Reihenfolge des Reststapels ändert sich in jedem dieser Schritte!! Hier in diesem Fall ist dies wichtig, weil man sonst nicht wie der Kellner sortiert!!)

Die Schritte sind kritisch, daher wird eine detaillierte Betrachtung der Datenstruktur vorgenommen.

Strukturierung des Algorithmus

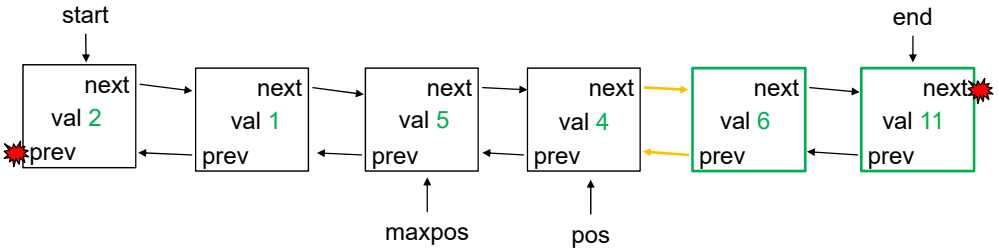
Datenstruktur (Lösung 1)

Wir wählen hier eine Liste, ohne diese genauer zu spezifizieren, da

1. Listen eine Reihenfolge wiedergeben.
2. Bei Listen ist der direkte Nachbar bei allen konkreten Listen meist direkt ansprechbar.
3. Da Listen mit Positionen assoziiert sind, sollte bei allen konkreten Listen die Position des zuletzt sortierten Elementes einfach darzustellen sein.
4. Eine schnelle Umkehrung der Reihenfolge wird von manchen konkreten Listen unterstützt. Evtl. wird dies nicht unterstützt. Ggf. müsste für die Implementierung eine (in der Summe) effizientere Struktur in der Programmiersprache verwendet werden.

Strukturierung des Algorithmus

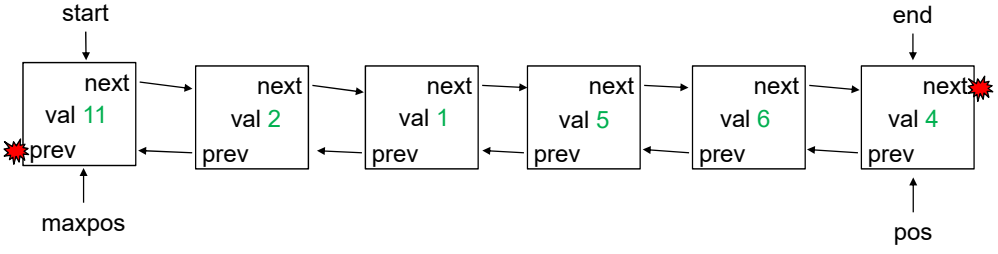
In dem beschriebenen Algorithmus verwenden wir eine doppelt verkettete Liste, um im Detail den Aufwand der einzelnen Schritte besser bewerten zu können.



Strukturierung des Algorithmus

n sei die Anzahl der Pfannkuchen (Flapjacks). next liefert den nächsten Nachbarn eines Elementes in der Liste, prev den Vorgänger. pos ist die Position vor dem sortierten Bereich, d.h. pos.next ist das kleinste sortierte Element. start, end zeigen auf das erste bzw. letzte (n-te) Element der Liste. Diese sind bei Eingabe bereits korrekt gesetzt (ggf. beim einlesen der Daten zu bewerkstelligen)

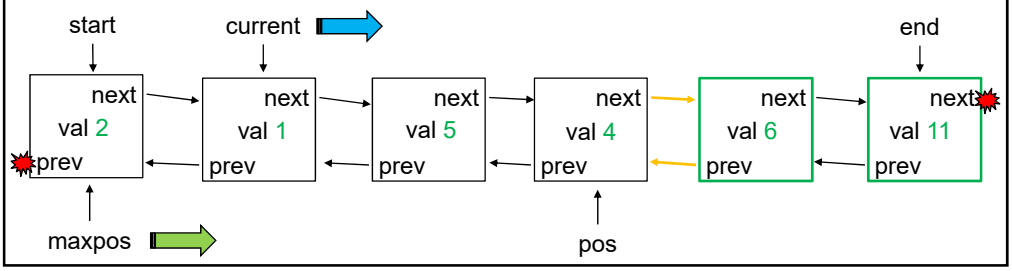
1. Setze **pos = end; sorted = false.**
2. …



Strukturierung des Algorithmus

2. Solange sorted ≠ true tue
   a) Setze current = start, maxpos = start.
      i. Solange current ≠ pos.next tue
         a) Falls current.val > maxpos.val dann maxpos = current.
         b) current = current.next.

   maxpos ist das größte nicht sortierte Element

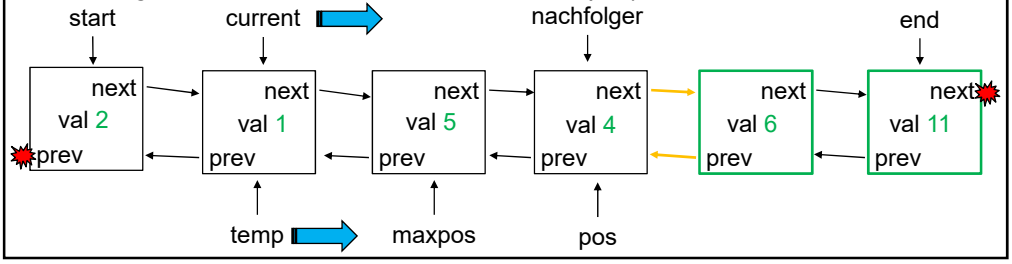


Strukturierung des Algorithmus

   b) temp = start.next, start.next = maxpos.next, start.prev = temp.
   current = temp, nachfolger = maxpos.next.
   c) Solange current ≠ nachfolger tue
      i. temp = current.next, current.next = current.prev.
      ii. current.prev = temp, current = current.prev.

   nachfolger.prev = start, start = maxpos, start.prev = null.

   maxpos ist das größte nicht sortierte Element und steht ganz oben (also ganz links in der Liste, d.h. start = maxpos)



Strukturierung des Algorithmus

d) temp = start.next, start.next = pos.next, start.prev = temp.
current = start.prev, nachfolger = pos.next.

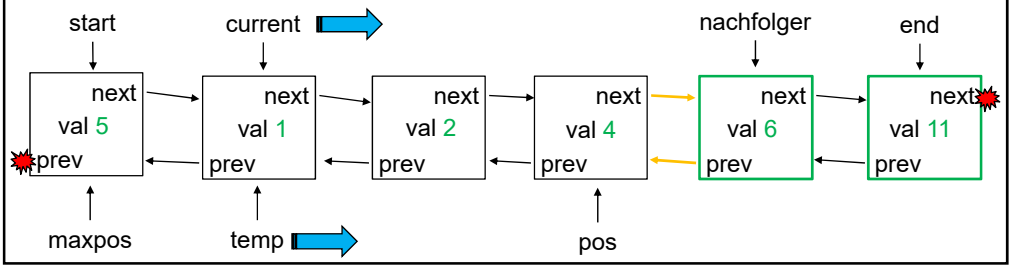
e) Solange current ≠ nachfolger tue

i. temp = current.next, current.next = current.prev.

ii. current.prev = temp, current = current.prev.

nachfolger.prev = start, start = pos, start.prev = null.
pos = nachfolger.prev.prev. Falls pos == null dann sorted = true

pos.next ist das kleinste nicht sortierte Element

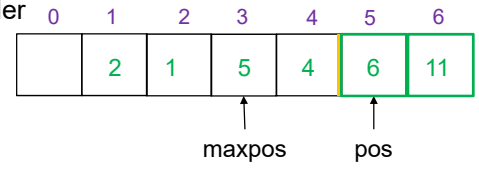


Strukturierung des Algorithmus

Datenstruktur (Lösung 2)

Wir wählen hier ein Feld, da

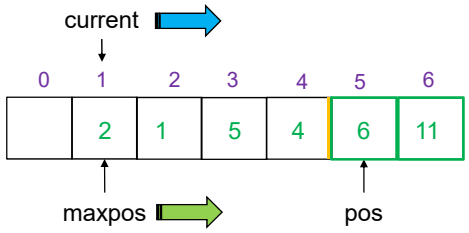
1. Felder eine Reihenfolge wiedergeben.
2. Bei Feldern liegt der direkte Nachbar im Speicher direkt neben dem aktuellen Feld und sein Index ist um eins größer, als der aktuelle.
3. Da Felder über den Index mit Positionen assoziiert sind, ist die Position des zuletzt sortierten Elementes einfach durch sein Index darzustellen.
4. Eine schnelle Umkehrung der Reihenfolge muss explizit durch kopieren der einzelnen Felder realisiert werden. Eine Umkehrung durch eine besondere Verwaltung der Indizes wird sehr schwierig, da die erste Umkehrung innerhalb des Stapels der zweiten Umkehrung liegt, d.h. ohne explizite Umkehrung besteht eine andere konkrete Reihenfolge der Elemente.



Strukturierung des Algorithmus

2. Solange pos ≠ start tue
   a) Setze current = start, maxpos = start.
      i. Solange current ≠ pos tue
         a) Falls feld[current] > feld[maxpos] dann maxpos = current.
         b) current = current + 1.

   maxpos ist das größte nicht sortierte Element

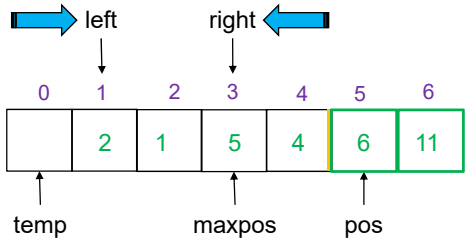


Strukturierung des Algorithmus

b) temp = 0, left = start, right = maxpos.
c) Solange left < right tue
   i. feld[temp] = feld[left], feld[left] = feld[right], feld[right] = feld[temp].
   ii. left = left + 1, right = right - 1.

feld[temp] = 0.

maxpos ist das größte nicht sortierte Element und steht ganz oben (also ganz links in der Liste, d.h. start = maxpos)



Strukturierung des Algorithmus

d) temp = 0, left = start, right = pos - 1.

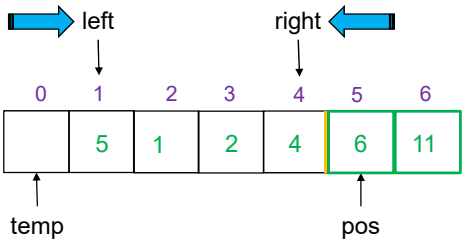
e) Solange left < right tue

i. feld[temp] = feld[left], feld[left] = feld[right], feld[right] = feld[temp].

ii. left = left + 1, right = right - 1.

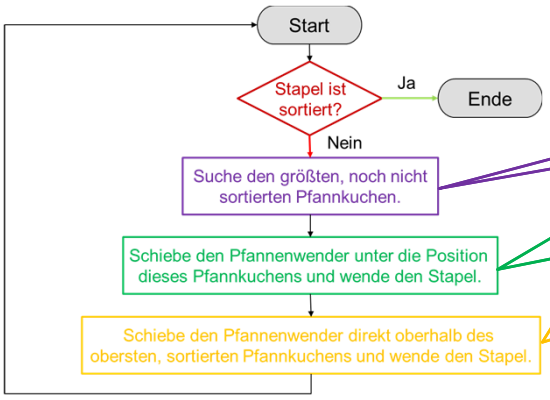
f) feld[temp] = 0, pos = pos - 1.

pos.next ist das kleinste nicht sortierte Element



Komplexität des Algorithmus

n: gesamte Anzahl an Elementen
i: Anzahl sortierter Elemente
j:= n - i



Zunächst n, dann n-1 … n-n Schritte:

$\sum_{i=0}^{n} i = \frac{n\cdot(n+1)}{2} = \frac{n^2+n}{2}$

best case: 1; worst case: j; insgesamt
best case: $n \cdot 1$
worst case: $\frac{1}{2} \cdot (n^2 + n)$

Zunächst n, dann n-1 … n-n Schritte:

$\sum_{i=0}^{n} i = \frac{n\cdot(n+1)}{2} = \frac{n^2+n}{2}$

In der Summe

best case : $n + (n^2 + n)$
worst case: $3/2 \cdot (n^2 + n)$

$O(n^2) = \Omega(n^2) = \Theta(n^2)$

Implementierung in Erlang/OTP

```
flapsort(Flapjack) -> flapsortR(Flapjack,[]).
flapsortR([],SortedStack) -> SortedStack;
flapsortR(Unsorted,Sorted) ->
   [Max|Rest] = flip(Unsorted,findlargest(Unsorted)),
   flapsortR(lists:reverse(Rest),[Max|Sorted]).

findlargest([H|Tail]) -> findlargest(Tail,H).
findlargest([],Max) -> Max;
findlargest([H|Tail],Max) when Max >= H -> findlargest(Tail,Max);
findlargest([H|Tail],Max) when Max < H -> findlargest(Tail,H).

flip(InListe,PosElem) -> flip(InListe,PosElem,[]).
flip([PosElem|Tail],PosElem,InFlipList) ->
    lists:append([PosElem|InFlipList],Tail);
flip([H|Tail],PosElem,InFlipList) ->
    flip(Tail,PosElem,[H|InFlipList]).
```


Implementierung in Prolog

```
flapsort(Flapjack,SortedStack) :- flapsortR(Flapjack,[],SortedStack).
flapsortR([],SortedStack,SortedStack):-!.
flapsortR(InFlapjack,SortedStack,FinalSortedStack):-
    findlargest(InFlapjack,MaxElem),
    flip(InFlapjack,MaxElem,[Max|Rest]),
    reverse(Rest,TmpFlapjack),
    flapsortR(TmpFlapjack,[Max|SortedStack],FinalSortedStack).

findlargest([H|Tail],OutMax) :- findlargest(Tail,H,OutMax).
findlargest([],InMax,InMax):-!.
findlargest([H|Tail],InMax,OutMax) :-
            InMax >= H,!, findlargest(Tail,InMax,OutMax).
findlargest([H|Tail],InMax,OutMax) :-
            InMax < H, findlargest(Tail,H,OutMax).

flip(InListe,PosElem) -> flip(InListe,PosElem,[]).
flip([PosElem|Tail],PosElem,InFlipList) ->
    lists:append([PosElem|InFlipList],Tail);
flip([H|Tail],PosElem,InFlipList) ->
    flip(Tail,PosElem,[H|InFlipList]).
```

Implementierung in Java

```
static int findLargest( int[] array, int start, int int pos ) {
    int current = start; int maxpos = start;
    while (current < pos) {
        if( array[current] > array[maxpos]) {
            maxpos = current;}
        current++;}
    return maxpos;}

static void flip( int[] array, int left, int right) {
    int temp = 0;
    while ( left < right ) {
        array[temp] = array[left];
        array[left] = array[right];
        array[right] = array[temp];
        left++; right--;}
    array[temp] = 0;}
```


Implementierung in Java

```
public static void sortFlapJacks(int[] array, int start, int end) {
    int pos = end + 1;
    while (pos > start) {
        int maxpos = findLargest(array, start, pos);
        flip( array, start, maxpos);
        flip( array, start, pos-1);
        pos--; } } }
```