Algorithms and Data Structures

Marius Kloft
Who am I

Marius Kloft

2006  Diploma in Mathematics, U Marburg
Minor: Computer Science

2007-2009  Doctoral Researcher, Fraunhofer & TU Berlin,
*Machine Learning for Intrusion Detection*

2009-2010  Visiting Scholar, University of California

2010-2011  Doctorial Student, TU Berlin

2011  Dissertation on *Multiple Kernel Learning*

2011-2012  Postdoc, TU Berlin *ML for Genomics*

2012-2014  Postdoc, Courant Institute, Sloan-Kettering
Cancer Center & Google Research

2014-  Junior Professor of *Machine Learning (ML)*,
HU Berlin
Lehrstuhl “Maschinelles Lernen”

• Our topics in research
  – Development of novel machine learning algorithms
  – Speeding up machine learning algorithms to big data (e.g., via distributed computing)
  – Statistical learning theory
  – Applications in the biomedical domain

• Our topics in teaching
  – Machine Learning
  – Data Modeling
  – Algorithms & Data Structures
What is Machine Learning?

• Central question
  – “How to develop computer programs that learn from data to make accurate predictions?”

• Example
  – Image classification
Once upon a Time ...

- **IT company A** develops software for **insurance company B**
  - Volume: ~4M Euros
- **B** not happy with delivered system; doesn’t want to pay
- **A and B** call a **referee** to decide whether requirements were fulfilled or not
  - Volume: ~500K Euros
- **Job of referee** is to understand requirements (~60 pages) and specification (~300 pages), survey software and manuals, judge whether the **contract was fulfilled** or not
One Issue

- Requirement: „Allows for smooth operations in daily routine“

This is hardly testable
One Issue

- Requirement: „Allows for smooth operations in daily routine“
- Claim from B
  - I search a specific contract
  - I select a region and a contract type
  - I get a list of all contracts sorted by name in a drop-down box
  - This sometimes takes minutes! A simple drop-down box! This performance is inacceptable for our call centre!
Discussion

• A: We tried and it worked fine
• B: OK, most of the times it works fine, but sometimes it is too slow
• A: We cannot reproduce the error; please be more specific in what you are doing before the problem occurs
• B: Come on, you cannot expect I log all my clicks and take notes on what is happening
• A: Then we conclude that there is no error
• B: Of course there is an error
• A: Please pay as there is no reproducible error
• ...
A Closer Look

• System has classical two-tier architecture

• Upon selecting a region and a contract, a query is constructed and send to the database

• Procedure for “query construction” is used a lot
  – All contracts in a region, ... running out this year, ... by first letter of customer, ... sum of all contract revenues per year, ...
  – “Meta” coding: very complex, hard to understand
Requirement

- Recall

One Issue

- Requirement: "Allows for smooth operations in daily routine"
- Observation from A
  - I search a specific contract
  - I select a region and a contract type
  - I get a list of all contracts sorted by name in a drop-down box
  - "It sometimes takes minutes! A simple drop-down box!"

- After retrieving the list of customers, it has to be sorted
Code used for Sorting the List of Customer Names

- **S**: array of Strings, $|S|=n$
- **Sort S alphabetically**
  - Take the first string and compare to all others
  - Swap whenever a later string is smaller
  - Repeat for 2\textsuperscript{nd}, 3\textsuperscript{rd}, ...
  - After 1\textsuperscript{st} iteration of outer loop: S[1] contains \textit{smallest string} from S
  - After 2\textsuperscript{nd} iteration of outer loop: S[2] contains 2\textsuperscript{nd} smallest string from S
  - etc.

```plaintext
S: array_of_names;
n := |S|;
for i = 1..n-1 do
  for j = i+1..n do
    if S[i]>S[j] then
      tmp := S[i];
      S[i] := S[j];
      S[j] := tmp;
    end if;
  end for;
end for;
```
Example

S: array_of_names;
n := |S|;
for i = 1..n-1 do
  for j = i+1..n do
    if S[i]>S[j] then
      tmp := S[i];
      S[i] := S[j];
      S[j] := tmp;
    end if;
  end for;
end for;
Example continued

- Seems to work
- This algorithm is called “selection sort”
  - Select smallest element and move to front, select second-smallest and move to 2\textsuperscript{nd} position, ...
Analysis

• How **long will it take** (depending on n)?
• Which **parts of the program** take CPU time?

1. Very little, constant time
2. Probably very little, constant time
3. n-1 assignments
4. n-i assignments
5. One comparison
6. One assignment
7. One assignment
8. One assignment
9. No time
10. One increment (j+1); one test
11. One increment (i+1); one test

```
1. S: array_of_names;
2. n := |S|;
3. for i = 1..n-1 do
4.   for j = i+1..n do
5.     if S[i]>S[j] then
6.       tmp := S[i];
7.       S[i] := S[j];
8.       S[j] := tmp;
9.     end if;
10. end for;
11. end for;
```
Slightly More Abstract

- Assume **one assignment/test costs** \(c\), **one addition** \(d\)
- **Which parts of the program take time?**

```
1. 0
2. c
3. (n-1)*c
   4. (n-i)*c (hmmm ...)
   5. c
   6. c
   7. c
   8. c
   9. 0
10. c+d
11. c+d
```

```plaintext
1. S: array_of_names;
2. n := |S|;
3. for i = 1..n-1 do
4.   for j = i+1..n do
5.     if S[i]>S[j] then
6.       tmp := S[i];
7.       S[i] := S[j];
8.       S[j] := tmp
9.     end if;
10. end for;
11. end for;
```
Slightly More Compact

- Assume one assignment/test costs $c$, one addition $d$

- Which parts of the program take time?
  - Let’s be **pessimistic**: We always swap
    - How would the list have to look like in first place?
      - $c$
      - $(n-1)\cdot c$ (• $n-i$ (• $S\cdot c$
        • $c+d$) +
      • $c+d$)

```
1. S: array_of_names;
2. n := |S|;
3. for i = 1..n-1 do
4.   for j = i+1..n do
5.     if S[i]>S[j] then
6.       tmp := S[i];
7.       S[i] := S[j];
8.       S[j] := tmp;
9.     end if;
10. end for;
11. end for;
```

This is not yet clear
Even More Compact

- Assume one assignment/test costs $c$, one addition $d$
- Which parts of the program take time?
  - We have some cost **outside the loop** (out_loops)
  - And some cost **inside the loop** (in_loops)
  - How often do we need to perform in_loops?
  - $c + (n-1)c^* ((n-i)^*...) =$ out_loops + (n-1)c^*?*in_loops

```
1. S: array_of_names;
2. n := |S|;
3. for i = 1..n-1 do
4.   for j = i+1..n do
5.     if S[i]>S[j] then
6.       tmp := S[i];
7.       S[i] := S[j];
8.       S[j] := tmp;
9.     end if;
10. end for;
11. end for;
```
Observations

- The number of comparisons is independent of the number of swaps
  - We always compare, but we do not always swap
Observations

- The **number of comparisons** is independent of the number of swaps
  - We always compare, but we do not always swap
- How many comparisons do we perform in total?
Observations

• The **number of comparisons** is independent of the number of swaps
  - We always compare, but we do not always swap

• How many comparisons do we perform in total?
Observations

- First string is compared to \textit{n-1} other strings
  - First row
- Second is compared to \textit{n-2}
  - Second row
- Third is compared to \textit{n-3}
  - ...
- \textit{n-1}'th is compared to 1
Together

\[(n - 1) + (n - 2) + (n - 3) + \ldots + 1 = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2}\]

- This leads to the following total cost estimation:
  \[\text{out_loops} + (n^2 - n) \times \frac{\text{in_loops}}{2}\]
- Let’s assume \(c=d=1\), then:
  \[n + 1 + (n^2 - n) \times \frac{8}{2}\]

<table>
<thead>
<tr>
<th>n</th>
<th>out_loops</th>
<th>in_loops</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>11</td>
<td>360</td>
<td>371</td>
</tr>
<tr>
<td>100</td>
<td>11</td>
<td>39.600</td>
<td>39.611</td>
</tr>
<tr>
<td>500</td>
<td>11</td>
<td>998.000</td>
<td>998.011</td>
</tr>
<tr>
<td>1,000</td>
<td>11</td>
<td>3.996.000</td>
<td>3.996.011</td>
</tr>
<tr>
<td>2,000</td>
<td>11</td>
<td>15.992.000</td>
<td>15.992.011</td>
</tr>
</tbody>
</table>
What Happened?

- Most combinations (region, contract type) select only a handful of contracts
- A few combinations select many contracts (2000-5000)
- Time it takes to fill the drop-down list is not proportional to the number of contracts \( (n) \), but proportional to \( n^2/2 \)
  - Required time is “quadratic in \( n \)”
  - Assume one comparison takes 10 nanoseconds (0.000001 sec)
  - A handful of contracts (\(~10\)): \(~500\) operations \(\Rightarrow\) 0,0005 sec
  - Many contracts (\(~5000\)) \(\Rightarrow\) \(~125M\) operations \(\Rightarrow\) 125 sec
  - Humans always expect linear time ...

- Question: Could they have done it better?
Of course

- **Efficient sorting algorithms** need $\sim n \cdot \log(n) \cdot x$ operations
  - Quick sort, merge sort, ... see later
  - For comparability, let’s assume $x=8$
  - Under certain reasonable assumptions, one cannot sort faster than with $\sim n \cdot \log(n)$ operations

![Graph showing almost linear relationship between n and operations](image)
So there is an End to Research in Sorting?

• We didn’t consider how long it takes to compare 2 strings
  – We used c=d=1, but we need to compare strings char-by-char
  – Time of every comparison is proportional to the length of the shorter string
• We want methods requiring less operations per inner loop
• We want algorithms that are fast even if we want to sort 1.000.000.000 strings
  – Which might not fit into main memory
• We made a pessimistic estimate – what is a realistic estimate (how often do we swap in the inner loop?)?
• …
Terasort Benchmark

• 2009: 100 TB in 173 minutes
  – Amounts to **0.578 TB/min**
  – 3452 nodes x (2 Quadcore, 8 GB memory)
  – Owen O'Malley and Arun Murthy, Yahoo Inc.

• 2010: 1,000,000,000,000 records in 10,318 seconds
  – Amounts to **0.582 TB/min**
  – 47 nodes x (2 Quadcore, 24 GB memory), Nexus 5020 switch
  – Rasmussen, Mysore, Madhyastha, Conley, Porter, Vahdat, Pucher

• Other goals
  – PennySort: Amount of data sorted for a penny's worth of system time
  – JouleSort: Minimize amount of energy required during sorting
Content of this Lecture

- This lecture
- Algorithms and ...
- Data Structures
- Concluding Remarks
Algorithms and Data Structures

- Slides are English
- Vorlesung wird auf Deutsch gehalten
- Acknowledgement: Prof. Ulf Leser
- Lecture: 4 SWS; exercises 2 SWS
- Contact:
  - Marius Kloft
  - RUD 25, Raum 4.215
  - Office hours: Fridays, 15:00-16:00
  - “Email”: only via Goya
  - Always cc your TA (=Übungsleiter(in)) when you write me a message
Exercises & TAs:

- Monday, 9-11, RUD 26, 1'303, Marc Bux
- Monday, 13-15, RUD 26, 1'305, Marc Bux
- Monday, 13-15, RUD 26, 1'303, Florian Tschorsch
- Tuesday, 9-11, RUD 26, 1'303, Patrick Schäfer
- Tuesday, 13-15, RUD 26, 0'313, Kim Völlinger
- Wednesday, 9-11, RUD 26, 1'306, Berit Grußien
- Thursday, 13-15, RUD 26, 1'305, Kim Völlinger
- Thursday, 13-15, RUD 26, 0'313, Patrick Schäfer
- Friday, 9-11, RUD 26, 1'305, Berit Grußien
- Friday, 11-13, RUD 26, 1'305, Florian Tschorsch
Schedule

• Tutorial: Michael R. Jung
  – Mondays, 17-19, RUD 26, 1'303
  – Wednesdays, 17-19, RUD 26, 1'303
  – Thursdays, 15-17, RUD 26, 1'306
  – Fridays, 11-13, RUD 25, 3.101

• Mathematics refresher course:
  – Wednesday, 9-11, RUD 26, 1'306, Berit Grußien
  – Thursday, 13-15, RUD 26, 0'313, Berit Grußien

• Exam:
  – Aug 1, 9:30-12:00, RUD 26, 0'115 & 0'110
    • „Klausureinsicht“: Aug 4, 11-13, RUD25, 3.101 & 3.113
  – Oct 4, 9:30-12:00, RUD 26, 0'115 (Wiederholungsklausur)
Lecture

• Mondays & Wednesdays, RUD 26, 0’115
  – 11:00-11:45 & 12:00-12:45
• We will make 15mins break
Exercises

- You will build teams of usually two students (maximally three) students registered in GOYA
- There will six bi-weekly assignments in total
- Each assignment gives 50 points
- Only groups having $\geq 50\%$ of the maximal number of points over the entire semester are admitted to the exam
Exercises (continued)

• Text-based homework assignments to be submitted in paper until **10:55 before the Monday lecture**
  – Or earlier in the letterbox at RUD 25, 3.321
  – New problem sheet available on the same day
• One-time exception: this week’s problem sheet will be released on Wednesday, April 20
  – You have time for submission until **Wednesday, May 4**
• First assignment available on Wednesday (is due May 2)
• Programming assignments to be tested with Java 1.6 on gruenau2 and **submitted in GOYA** (same deadline)
Literature

- **Ottmann, Widmayer**: Algorithmen und Datenstrukturen, Spektrum Verlag, 2002-2012
  - 20 copies in library

- **Other**
  - Saake / Sattler: Algorithmen und Datenstrukturen (mit Java), dpunkt.Verlag, 2006
    - 20 copies in library
  - Güting, Dieker: Datenstrukturen und Algorithmen, Teubner, 2004
    - 10 copies in library
Web
Website: Lecture

- https://hu.berlin/vl_algodat16
Website: Exercises

<table>
<thead>
<tr>
<th>Übung Algorithmen und Datenstrukturen</th>
</tr>
</thead>
<tbody>
<tr>
<td>Informationen und Materialien zur begleitenden Übung der Vorlesung Algorithmen und Datenstrukturen im Sommersemester 2016</td>
</tr>
</tbody>
</table>

**Bearbeitung und Abgabe**

**Erster Übungstermin: 25.04.2016**

Das erste Übungsblatt wird am Mittwoch, den 20.04. auf dieser Seite und in GOYA veröffentlicht. Die Abgabe des ersten Übungsblatts erfolgt dann am 04.05. Anschließend werden die Übungsblätter alle zwei Wochen montags online gestellt, beginnend mit dem zweiten Übungsblatt am 02.05. Die Abgabe erfolgt von dann an jeweils am übernächsten Montag. Sie haben also immer zwei Wochen Bearbeitungszeit für jedes Aufgabenblatt, wobei sich die ersten beiden Übungsblätter um zwei Tage überschneiden. Jedes Übungsblatt hat vier Aufgaben, für die insgesamt 50 Punkte erhältlich sind. Insgesamt wird es sechs Übungsblätter geben.


Die Übungsaufgaben sind in Gruppen von je zwei, in Ausnahmen auch mit drei, Studenten zu bearbeiten. Studenten die nach der ersten Woche noch keinen Übungspartner haben, bekommen einen Übungspartner zugewiesen.


Abgaben die sich nicht kompilieren und ausführen lassen werden mit 0
Pseudo Code

- You need to program all exercises in Java
- I will use informal pseudo code
  - Much more concise than Java
  - Goal: You should understand what I mean
  - Syntax is not important; don’t try to execute programs from slides
- Translation into Java should be simple
Topics of the Course

- Intro (~2)                      April
- Complexity (~1)                 Mai
- Abstract data types (~2)        June
- Lists (~2)                      July
- Sorting (~3)                    
- Searching in lists (~4)         
- Queues & Hashing (~3)           
- Search trees (~4)               
- Graphs (~5)                     
- The end (~1)
Questions?
Questions

- BSc CS?
- Diplom CS?
- BSc Mathematics?
- Kombibachelor?
- INFOMIT? Biophysics? Beifach?
- Semester?
- Who heard this course before?
Content of this Lecture

• This lecture
• Algorithms and ...
• Data Structures
• Concluding Remarks
What is an Algorithm?

• An algorithm is a **recipe for doing something**  
  – Washing a car, sorting a set of strings, preparing a pancake, 
    employing a student, ...
• The recipe is given in a ([formal](#), clearly defined) language
• The recipe consists of **atomic steps**  
  – Someone (the machine) must know what to do
• The recipe must be precise  
  – After every step, it must be **uniquely decidable** what comes next  
  – Does not imply that every run has the **same sequence of steps**
• The recipe must not be infinitely long
More Formal

• Definition (general)
  An algorithm is a **precise and finite description** of a process consisting of **elementary steps**.

• Definition (Computer Science)
  An algorithm is a **precise and finite description** of a process that is (a) **given in a formal language** and (b) **consists of elementary and machine-executable steps**.

• Usually we also want: “and (c) solves a **given problem**”
  – But algorithms can be wrong ...
Almost Synonyms

• Rezept
• Ausführungsvorschrift
• Prozessbeschreibung
• Verwaltungsanweisung
• Regelwerk
• Bedienungsanleitung
  – Well ...
• ...
  

[444x109] – Well ...
[477x73] • …
History

• Word presumably dates back to “Muhammed ibn Musa abu Djafar alChoresmi”,
  – Published a book on calculating in the 8th century in Persia
  – See Wikipedia for details

• Given the general meaning of the term, there have been algorithms since ever

• One of the first prominent one in math: Euclidian algorithm for finding the greatest common divisor (gcd) of two ints
  – Assume $a,b \geq 0$; define $\text{gcd}(a,0)=a$
Euclidian Algorithm

- Recipe: Given two integers $a$, $b$. As long as neither $a$ nor $b$ is 0, take the smaller of both and subtract it from the greater. If this yields 0, return the other number.
- Example: $(28, 92)$
  - $(28, 64)$
  - $(28, 36)$
  - $(28, 8)$
  - $(20, 8)$
  - $(12, 8)$
  - $(4, 8)$
  - $(4, 4)$
  - $(4, 0)$

- Will this always work?

```
1. a,b: integer;
2. if a=0 return b;
3. while b ≠ 0
4.   if a>b
5.     a := a-b;
6.   else
7.     b := b-a;
8. end if;
9. end while;
10. return a;
```
Proof (sketch) that an Algorithm is Correct

- Assume our function “euclid” returns $x$
- We write “$b | a$” if $(a \mod b) = 0$
  - We say: “$b$ teilt $a$”
- 1st step: $x$ is a common divisor of $a$ and $b$
  - Last step: $b=0$ and $x=a\neq 0 \Rightarrow x|a, x|b$
  - Pre-last: It must hold: $a=b \Rightarrow x|a, x|b$
  - Previous: Either $a=2x$ or $b=2x \Rightarrow x|a, x|b$
  - Previous: Either $(a,b)=(3x,x)$ or $(a,b)=(2x,3x)$ or $(a,b)=(x,3x)$ or $(a,b)=(3x,2x) \Rightarrow x|a, x|b$
  - ...
Proof (sketch) that an **Algorithm is Correct**

- **Note:** if $c|a$ and $c|b$ and $a>b \implies c|(a-b)$
- **2nd step:** $x$ is the greatest divisor
  - Assume some $y$ with $y|a$ and $y|b$
  - It follows that $y|(a-b)$ (or $y|(b-a)$)
  - It follows that $y|((a-b)-b)$ (or $y|((b-a)-b)$ ...)
  - ...
  - It follows that $y|x$
  - Thus, $y\leq x$

```python
1. func euclid(a,b: int)
2. if a=0 return b;
3. while b ≠ 0
4. if a>b
5. a := a-b;
6. else
7. b := b-a;
8. end if;
9. end while;
10. return a;
11. end func;
```
Properties of Algorithms

• Definition

An algorithm is called terminating if it stops after a finite number of steps for every valid input

• Definition

An algorithm is called deterministic if it always performs the same series of steps given the same input

• We only study terminating and mostly deterministic algs
  – Operating systems are “algorithms” that do not terminate
  – Algs randomly deciding about next steps are not deterministic
Algorithms and Runtimes

• Usually, one seeks efficient (read: fast) algorithms
• We will analyze the efficiency of an algorithm as a function of the size of its input; this is called its (time-)complexity
  – Selection-sort has time-complexity “O(n²)”
• The real runtime of an algorithm on a real machine depends on many additional factors we gracefully ignore
  – Clock rate, processor, programming language, representation of primitive data types, available main memory, cache lines, ...
• But: Complexity in some sense correlates with runtime
  – It should correlate well in most cases, but there may be exceptions (especially on small inputs)
Algorithms, Complexity and Problems

• An (correct) algorithm solves a given problem
• An algorithm has a certain complexity
  – Which is a statement about the time it will take to finish as a function on the size of its input
• Also problems have complexities
  – The complexity of a problem is a lower bound on the complexity of any algorithm that solves it
  – If an algorithm has the same complexity as the problem it solves, it is optimal – no algorithm can solve this problem faster
• Proving the complexity of a problem usually is much harder than proving the complexity of an algorithm
  – Needs to make a statement about any possible algorithm
Relationships

- There are problems for which we know their complexity, but no optimal algorithm is known.
- There are problems for which we do not know the complexity yet more and more efficient algorithms are discovered over time.
- There are problems for which we only know lower thresholds on their complexity, but not the precise complexity.
- There are problems of which we know that no algorithm exists.
  - Undecidable problems
  - Example: “Halteproblem”
  - Implies that we cannot check in general if an algorithm is terminating.

Source: S. Albers, Alg&DS; SoSe 2010
Properties of Algorithms

1. Efficiency – how long will it take?
   - Time complexity
   - Worst-case, average-case, best-case
   - Alternative: Run on reference machine using reference data set
     - Done a lot in practical algorithm engineering
     - Not so much in this introductory course

2. Space consumption – how much memory will it need?
   - Space complexity
   - Worst-case, average-case, best-case
   - Can be decisive for large inputs

3. Correctness – does the algorithm solve the problem?

Often, one can trade space for time – look at both
In This Module

- We will only occasionally look at space complexity
- We will mostly focus on worst-case time complexity
  - Best-case is not very interesting
  - Average-case often is hard to determine
    - What is an „average string list“?
    - What is the average length of an arbitrary string?
    - May depend in the semantic of the input (person names, DNA sequences, job descriptions, book titles, language, …)
- Keep in mind: Worst-case often is overly pessimistic
Content of this Lecture

- This lecture
- Algorithms and ...
- **Data Structures**
- Concluding Remarks
What is a Data Structure?

- Algorithms work on input data, generate intermediate data, and finally produce result data
- A data structure is a way how data is represented inside the machine
  - In memory or on disc (see Database course)
- Data structures determine what algs may do at what cost
  - More precisely: ... what a specific step of an algorithm costs
- Complexity of algs is tightly bound to the ds they use
  - So tightly that one often subsumes both concepts under the term “algorithm”
Example: Selection Sort (again)

- We assumed that S is
  - a list of strings (abstract), represented
  - as an array (concrete data structure)
- Arrays allow us to access the i’th element with a cost that is independent of i (and |S|)
  - Constant cost, “O(1)”
- Let’s use a linked list for storing S
  - Create a class C holding a string and a pointer to an object of C
  - Put first $s \in S$ into first object and point to second object, put second $s$ into second object and point to third object, ...
  - Keep a pointer $p_0$ to the first object

```plaintext
1. S: array_of_names;
2. n := |S|;
3. for i = 1..n-1 do
4.   for j = i+1..n do
5.     if S[i]>S[j] then
6.       tmp := S[i];
7.       S[i] := S[j];
8.       S[j] := tmp;
9.     end if;
10.  end for;
11. end for;
```
Selection Sort with Linked Lists

• How much do the algorithm’s steps cost now?
  – Assume following a pointer costs \( c \)
  1. One assignment
  2. Nothing
  3. One assignment, \( n-1 \) times
  4. Nothing
  5. One comparison, \( \ldots \) times
  6. \( \ldots \)

• Apparently no change in complexity
  – Why? Only sequential access
Example Continued

- No change in complexity, but
  - Previously, we accessed array elements, performed additions of integers and comparisons of strings, and assigned values to integers
  - Now, we assign pointers, follow pointers, compare strings and follow pointers again
- These differences are not reflected in our “cost model”, but may have a big impact in practice

```
1. i := p0;
2. repeat
3.   j := i.next;
4.   repeat
5.     if i.val > j.val then
6.       tmp := i.val;
7.       i.val := j.val;
8.       j.val := tmp;
9.     end if;
10.   j = j.next;
11. until j.next = null;
12. i := i.next;
13. until i.next = null;
```
Content of this Lecture

- This lecture
- Algorithms and Data Structures
- Concluding Remarks
Why do you need this?

- You will learn things you will need a lot through **all of your professional life**
- Searching, sorting, hashing – cannot Java do this for us?
  - Java libraries contain efficient implementations for most of the (basic) problems we will discuss
  - But: Choose the **right algorithm / data structure** for your problem
    - TreeMap? HashMap? Set? Map? Array? ...
    - “Right” means: Most efficient (space and time) for the expected operations: Many inserts? Many searches? Biased searches? ...
- Few of you will design new algorithms, but all of you often will need to decide **which algorithm** to use when
- To prevent problems like the ones we have seen earlier
Exemplary Questions

- Give a definition of the concept “algorithm”
- What different types of complexity exist?
- Given the following algorithm ..., analyze its worst-case time complexity
- The following algorithm ... uses a double-linked list as basic set data structure. Replace this with an array
- When do we say an algorithm is optimal for a given problem?
- How does the complexity of an algorithm depend on (a) the data structures it uses and (b) the complexity of the problem it solves?