Algorithms and Data Structures

Ulf Leser
Wir suchen SHKs!!!!
Who am I

- Ulf Leser

- 1995 Diploma in Computer Science, TU München
- 1996-1997 Database developer at MPI-Molecular Genetics
- 1997-2000 Dissertation in Database Integration, TU Berlin
- 2000-2003 Developer and project manager at PSI AG
- 2003- Prof. Knowledge Management in Bioinformatics

- I read emails a lot
Wissensmanagement in der Bioinformatik

- **Our topics in research**
  - Management of biomedical data and knowledge
  - Scientific database systems
  - Text Mining
  - Biomedical data analysis

- **Our topics in teaching**
  - Grundlagen der Bioinformatik
  - Algorithmische Bioinformatik
  - Text Analytics
  - Data Warehousing und Data Mining
  - Informationsintegration
Once upon a Time ...

- **IT company A** implements software for an **insurance company B**
  - Volume: ~4M Euros
- Customer B is 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“
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 is unacceptable performance 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, but anyway
Deeper Inside the System

• 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
  – „This 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\(^{nd}\), 3\(^{rd}\), ... name
- After 1\(^{st}\) iteration of outer loop, \( S[1] \) is contains the **smallest string** in \( S \)
- 2\(^{nd}\) iteration: \( S[2] \) contains the second smallest string
- 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
            temp := S[j];
            S[i] := S[j];
            S[j] := temp;
        end if;
    end for;
end for;
```

Ulf Leser: Alg&DS, Summer semester 2011
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
      temp := S[j];
      S[i] := S[j];
      S[j] := temp;
    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 2nd position, ...
Analysis

• How **long will it take** (depending on n)?
• Which **parts of the program** take time?
  1. No time
  2. Not sure ... maybe n additions
  3. n-1 times one assignment
  4. n-i+1 times one assignment
  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

```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. temp := S[j];
7. S[i] := S[j];
8. S[j] := temp;
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. $n \cdot d + c$
3. $(n-1) \cdot c$
4. $n-i+1 \ (\text{hmmm ...})$
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.       temp := S[j];
7.       S[i] := S[j];
8.       S[j] := temp
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?
      - \( n \cdot d + c \)
      - \( (n-1) \cdot c \cdot (n-i+1) \cdot (\cdot 4^c \cdot c + d) + \)
      - \( c + d \)
• 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?
  – \( n*d + c + (n-1)*c* (n-i+1*(6*c+2*d)) = \) out_loops+ (n-1)*c*?*in_loops
Observations

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

- The **number of comparisons** is independent from 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 from 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 from the number of swaps
- First string is compared to $n-1$ other strings
- Second is compared to $n-2$
- Third is compared to $n-3$
- …
- $n-1$’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

\[\text{out\_loops} + (n^2-n)*\text{in\_loops}/2\]

- Let’s assume \(c=d=1\)

\[n+1+(n^2-n)*8/2\]

<table>
<thead>
<tr>
<th>out_loops</th>
<th>in_loops</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>11</td>
<td>360 371</td>
</tr>
<tr>
<td>100</td>
<td>101</td>
<td>39.600 39.701</td>
</tr>
<tr>
<td>500</td>
<td>501</td>
<td>998.000 998.501</td>
</tr>
<tr>
<td>1000</td>
<td>1001</td>
<td>3.996.000 3.997.001</td>
</tr>
<tr>
<td>2000</td>
<td>2001</td>
<td>15.992.000 15.994.001</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 roughly to \(n^2/2\)
  – We say the time is not linear in \(n\), but “quadratic in \(n\)”
  – Assume one comparison takes 10 nanoseconds (0.000001 sec)
  – A handful of contracts (~10): ~400 operations => 0.0004 sec
  – Many contracts (~5000) => ~100M operations => 100 sec
  – Humans always expect linear time ... no clue what’s going on

• Question: Could they have done better?
Of course

- An **efficient sorting algorithm** needs app. n*\log(n)*x ops
  - Quick sort, merge sort, ... – see later
  - For comparability, let’s assume x=8

“Almost” linear
So there is an End to Research in Sorting?

- We didn’t consider how long it takes to compare 2 strings
  - We assumed „c=d=1“, but we need to compare them character by character, so the time is proportional to the length of the shorter string in a comparison
- We want algorithms that require less than 8 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
- Lecture: 4 SWS; exercises 2 SWS
- Contact
  - Ulf Leser,
  - Raum IV.103
  - Tel: 2093 – 3902
  - eMail: leser (...) informatik . hu...berlin . de
Dates

- Lectures: Dienstag 11-13, Donnerstag 11-13, EZ 0115
- Exercises
  - Mo. 09-11, 1305, Pollex
  - Di. 13-15, 1303, Gierds
  - Di. 15-17, 1303, Gierds
  - Mi. 11-13, 0313, Killat
  - Mi. 13-15, 0313, Killat
  - Do. 13-15, 1303, Lenzner
  - Do. 15-17, 1303, Lenzner
  - Fr. 09-11. 1303, Pollex
Exercises

- You will build teams of two students
- There will be an assignment every two weeks
- You need to work on every assignment
- Each assignment gives 40 points max
- Only groups having $\geq 60\%$ of the maximal number of points over the entire semester are admitted to the exam
- For every assignment, 2-3 students are selected at random (in each slot) and must present their solution
- Failing to do so more than two times implies exclusion from exercise
- It will be fun (mostly)!
Literature

- **Ottmann, Widmayer**: Algorithmen und Datenstrukturen, Spektrum Verlag, 2002
  - 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
Algorithmen und Datenstrukturen

Vorlesung im Sommersemester 2011
Professor Ulf Leser


Die Vorlesung wird durch eine Übung begleitet. Die Einschreibung im S Vita erfolgt ausschließlich über die Übungen.

Voraussetzungen
Voraussetzung für den Besuch sind gute Kenntnisse in Java.

Prüfungen
Das Modul wird mithilfe einer Klausur abgeschlossen. Voraussetzung zur Zulassung ist die Erreichung von mindestens 60% der Punkte in der Übung.

Anrechnung
Das Modul (Vorlesung + Übung) kann angerechnet werden für
- III. Beibachelor Informatik (Kursbereiche im zweiten Semester, 8 SP)
- II. Beibachelor Informatik, Kern- und Datenschicht (Kursbereiche im vierten Semester, 8 SP)
- Für einige Fächer auch im Informatik Bachelor

Literatur zur Vorlesung
- Ottmann, Widmesser: Algorithmen und Datenstrukturen, Spektrum Verlag
- Sadeh, Spiteri: Algorithmen und Datenstrukturen (mit Java), Springer-Verlag
- Sedgewick: Algorithmen in Java, Teil 1 - 4, Pearson Studium
- Cormen, Leiserson, Rivest, Stein: Introduction to Algorithms, MIT Press

Themen der Vorlesung
Die Folien werden hier jeweils nach der Vorlesung als PDF erhältlich sein.
- Einführung
- Abstrakte Datentypen
- Effizienz und O-Notation
- Listen, Stacks, Queues
- Sortieren
Pseudo Code

- You need to program all exercises in Java
  - Exercises will also be taught in Java
- However, as you noticed I will use pseudo code
  - Goal: You should understand what I mean
  - Syntax is not important; don’t try to execute programs from slides
  - Much more concise than Java
  - Will some times use “telling names” to abbreviate things
    - “S: array_of_names”
- But: Translation into Java should be simple
  - The “expressiveness” of my pseudo code is derived from Java
  - Assignments; primitive data types; for, foreach, while, repeat; comparisons; if...then...else; pointers; ...
Topics of the Course

- Abstract data types (~1)
- Machine models and complexity (~2)
- Lists (~3)
- Sorting (~4)
- Hashing (~3)
- Trees (~5)
- Graphs and graph algorithms (~4)
- Unusual classes of algorithms (~3)
Questions?
Questions

- Diplom informatiker?
- Bachelor?
- Semester?
- Kombibachelor?
- Who heard this course before?
- Studies other than Computer Science?
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 should 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 consists of **atomic steps**
- Every single step should be “primitive”
  - Someone (the machine) must know what to do
- 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 ...
- ...

Ulf Leser: Alg&DS, Summer semester 2011
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 in math: Euclidian algorithm for finding the greatest common divisor (gcd) of two integers
  - Define gcd(a,0)=a
Euclidean 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 \(b\neq 0\)
4. if \(a>b\)
5. \(a := a-b\);
6. else
7. \(b := b-a\);
8. end if;
9. end while;
10. return \(a\);
We need to Proof 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”
- Note: if c|a and c|b and a>b ⇒ c|(a-b)
- Thus, if euclid(a,b)=x ⇒ x|a and x|b
- But is it the greatest?
  - 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≤x
Properties of Algorithms

- Definition
  We say an algorithm $A$ is terminating, if $A$ stops after a finite number of steps for every valid input.

- Definition
  We say an algorithm $A$ is deterministic, if $A$ always performs the same series of steps for the same input.

- We only study terminating and mostly deterministic algs
  - Counter example: operating systems are “algorithms” that do not terminate
  - Counter example: algorithms using random numbers to decide 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
  – Our selection-sort had time-complexity “O(n^2)”
• The runtime of an algorithm depends on many factors most of which we gracefully ignore
  – Clock rate, processor, programming language, representation of primitive data types, available main memory, cache lines, ...
• However, the goal is that the complexity “somehow” correlates to the runtime of the algorithm
  – It should correlate well in most cases, but there may be pathological cases (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
- Beware: 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 algorithms
Anything Goes

- 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 always check if an algorithm is terminating.

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

1. Efficiency – how long will it take?
   - **Time complexity** – changes in runtime with growing input
   - 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 Course

- 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 average number of twisted sorts in an arbitrary 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, …)
- But keep in mind: Worst-case often is overly pessimistic
  - We shall see some examples
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 a result (data)
- A **data structure** is a way how all this data is represented inside the machine
  - In memory or on disc
- Data structures determine what **algorithms may do at what cost**
  - More precisely: ... what a specific step of an algorithm costs
- The complexity of an algorithm is tightly bound to the data structures it uses
  - 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 (ADT), 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, in $O(1)$
- Let’s change the representation to a linked list
  - 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
Selection Sort with Linked Lists

- How much do the algorithm’s steps cost now?
  - Assume following a pointer costs c
    1. One assignment
    2. One comparison, n times
      3. One assignment, n-i+1 times
      4. One comparison
      5. ...

- Apparently no change in complexity

```plaintext
1. i := p0;
2. while i.next != null do
3.   j := i.next;
4.   while j.next != null do
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. end while;
12. i := i.next;
13. end while;
```
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 make a big difference in practice

```plaintext
1. i := p0;
2. while i.next != null do
3.     j := i.next;
4.     while j.next != null do
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.   end while;
12. i := i.next;
13. end while;
```
Clarifications

- Data structures for **primitive data types** (integer, real, boolean) are not in focus here
  - We will studies data structures for lists, sets, graphs,…
  - Let’s say: “complex” things (note: Strings are complex!)
- We make a difference between an **“abstract data types” (ADT)** and a concrete data type, implementing an ADT
- ADT are **associated to operations**
  - List: insert an element, delete an element, search an element, …
  - We call the set of operations the **signature of the ADT**
- The data structure used to represent an ADT determines the complexity of these operations
  - The operations of an ADT have no complexities
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 (very basic) problems we will discuss
  - But: Chose 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 most of you often will need to decide which algorithm to use
- To prevent problems like the ones we have seen earlier
My Goal

- **Make you understand**
  - Syntax is not important – understand what’s going on
  - See that there is a relevant problem
  - Think in abstract categories – *essence of problems*

- **Make you believe**
  - Acknowledge the *beauty of the solutions* (at least sometimes)
  - Algorithms are a *wonderful part* of Computer Science