北京航空航天大學
BEIHANG UNIVERSITY

Fundamentals of Sofware Systems

Lecture 1

Xiaoqian Sun / Sebastian Wandelt

Beihang University

Outline

- Introduction
  - About you and us
  - What is Computer Science?
  - A (very) brief history of Computer Science
  - About this Course
  - Administrative issues

About you?

- Name?
- Where are you from?
- Major?
- CS (computer science) background?
- What do you expect from attending this course?
- How do you think you will use CS techniques in your future?
- What else do you want everybody to know?

About us

- Name: Xiaoqian Sun
- Chinese Name: **孙小倩**
- Nationality: Chinese
- sunxq@buaa.edu.cn

- Name: Sebastian Wandelt
- Chinese Name: 小塞
- Nationality: German
- wandelt@buaa.edu.cn

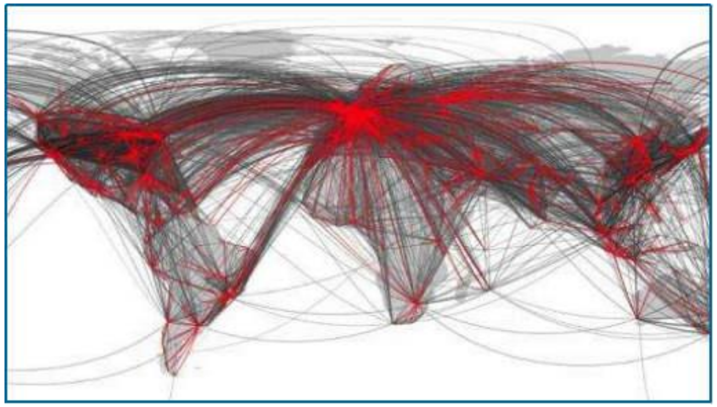
About our group

M3Nets (Multi-Modal Mobility Networks)

Our focus is on developing scalable algorithms and analysis techniques for large transportation networks

1) Improvement of Global Transportation Networks
2) Network Resilience
3) Compressed Storage and Indexing of Big Data

More information can be found here: http://m3nets.de



What is CS?

What is computer science?

Computer Science,

= Telescope science ?

"Computer science is no more about computers than astronomy is about telescopes"

-- Edsger W. Dijkstra

"The computer is not our object of study, It's our observational instrument"

What is computer science?

Some misconceptions.

- **Misconception 1**: I can put together my own PC, am good with Windows, and can surf the net with ease, so I know CS.
- **Misconception 2**: Last night I did 20 head shots in Counter Strike, so I know CS.
- **Misconception 3**: Computer science is the study of how to write computer programs.
- **Misconception 4**: Computer science is the study of the uses and applications of computers and software.

What is computer science?

- **Computer science is the study of algorithms, including**
  - Their formal and mathematical properties
  - Their linguistic realizations
  - Their applications

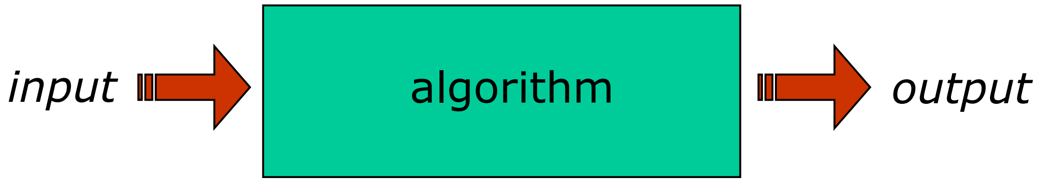
Terminology

- **Algorithm:** A set of steps that defines how a task is performed
- **Program:** A representation of an algorithm
- **Programming:** The process of developing a program
- **Software:** Programs and algorithms
- **Hardware:** Physical equipment

Algorithms

- Informally, an algorithm is …

A <u>well-defined computational procedure</u> that takes some value, or set of values, as ***input*** and produces some value, or set of values, as ***output***.



A <u>sequence</u> of <u>computational steps</u> that transform the ***input*** into ***output***.

Algorithms

- Empirically, and algorithm is …

A tool for solving a well-specified computational problem.

Problem specification includes what the input is, what the desired output should be.

Algorithm describes a specific computational procedure for achieving the desired output for a given input.

Algorithms

The Sorting Problem:

Input: A sequence of n numbers $[a_1, a_2, \dots, a_n]$.

Output: A permutation or reordering $[a'_1, a'_2, \dots, a'_n]$ of the input sequence such that $a'_1 \leq a'_2 \leq \dots \leq a'_n$.

An instance of the Sorting Problem:

Input: A sequence of 6 number [31, 41, 59, 26, 41, 58].

Expected output for given instance:

Expected Output: The permutation of the input [26, 31, 41, 41, 58 , 59].

Algorithms

- Task: Please come up with an algorithm which computes the maximum difference in age for all pairs of students in the room!

Algorithms

- Task: Please come up with an algorithm which computes the maximum difference in age for all pairs of students in the room!

Do you think your algorithm is efficient?

Algorithms

- Some definitions …

An algorithm is said to be ***correct*** if, for every input instance, it halts with the correct output.

A correct algorithm ***solves*** the given computational problem.

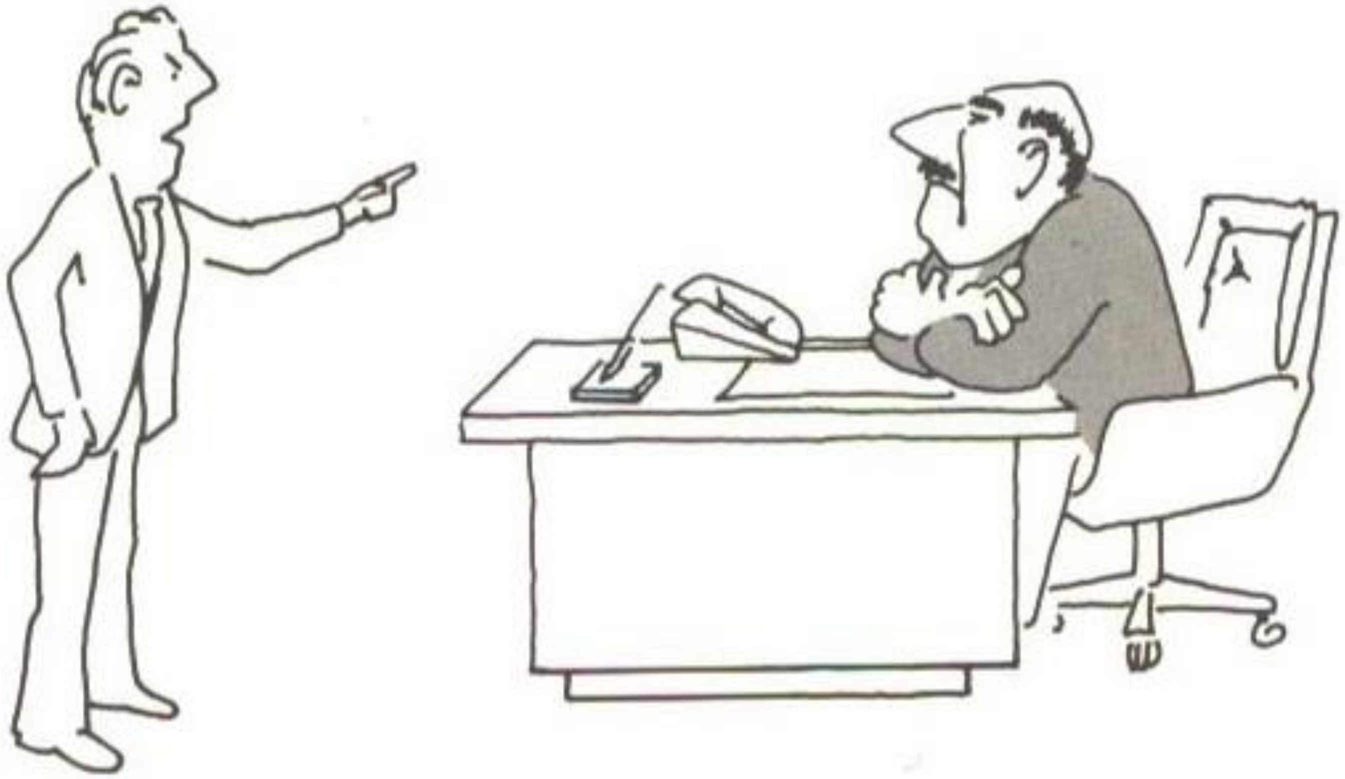
Focus will be on correct algorithms; incorrect algorithms can sometimes be useful.

Algorithm specification may be in English, as a computer program, even as a hardware design.

About algorithm correctness

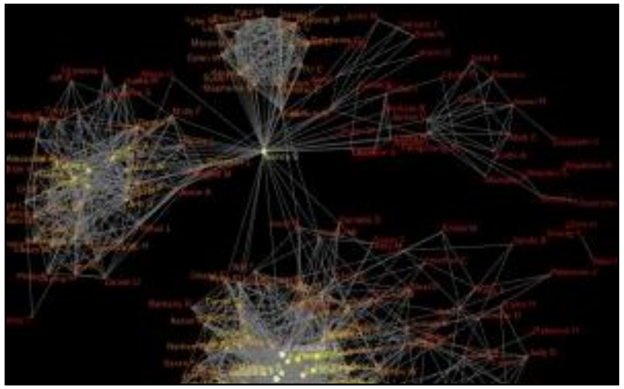
"I can't find an efficient algorithm, but neither can all these famous people."

About algorithm correctness


“I can’t find an efficient algorithm, because no such algorithm is possible!”

Some algorithms

- Shortest path algorithm
  - Given a weighted graph and two distinguished vertices -- the source and the destination
    -- compute the most efficient way to get from one to the other
- Matrix multiplication algorithm
  - Given a sequence of conformable matrices, compute the most efficient way of forming the product of the matrix sequence



Hard problems

- Usual measure of efficiency is speed
  - How long does an algorithm take to produce its result?
  - Define formally measures of efficiency
- Problems exist that, in all probability, will take a long time to solve
  - Exponential complexity
  - NP-complete problems
- Problems exist that are unsolvable

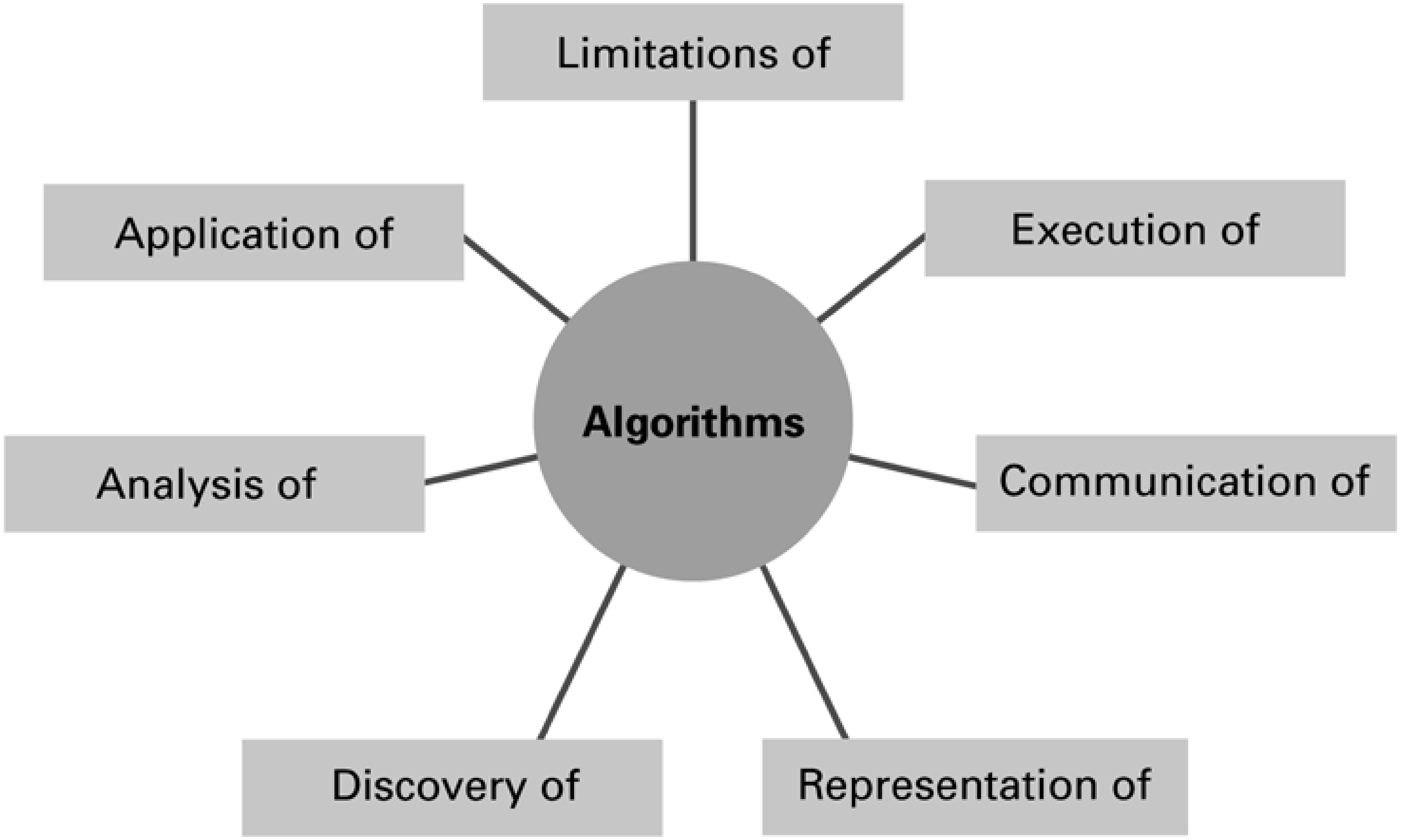
Algorithms

- Even if computers were infinitely fast and memory was plentiful and free
  - Study of algorithms still important – still need to establish algorithm correctness
  - Since time and space resources are infinite, any correct algorithm would do
- Real-world computers are fast but not infinitely fast
- Memory is cheap but not unlimited

Algorithms

- Time and space efficiency are the goal
- Algorithms often differ dramatically in their efficiency
  - Example?

Summary: The central role of algorithms



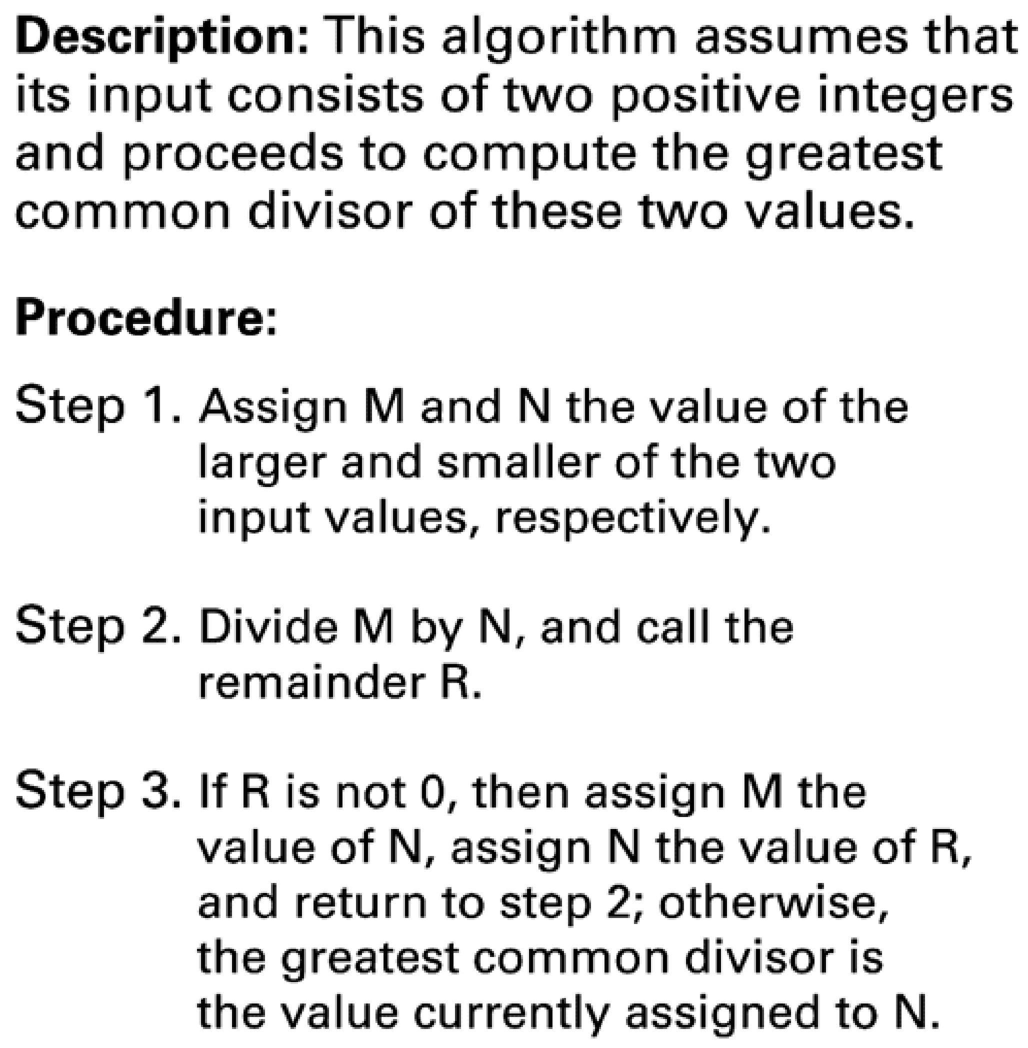
Central questions in CS

- Which problems can be solved by algorithmic processes?
- How can algorithm discovery be made easier?
- How can techniques of representing and communicating algorithms be improved?
- How can characteristics of different algorithms be analyzed and compared?
- How can algorithms be used to manipulate information?
- How can algorithms be applied to produce intelligent behavior?
- How does the application of algorithms affect society?
- …

History of algorithms

- The study of algorithms was originally a subject in mathematics.
- Early examples of algorithms
  - Long division algorithm
  - Euclidean Algorithm
- **Gödel's Incompleteness Theorem**: Some problems cannot be solved by algorithms.

A simple example: Euclid‵s algorithm



Description: This algorithm assumes that its input consists of two positive integers and proceeds to compute the greatest common divisor of these two values.

Procedure:

Step 1. Assign M and N the value of the larger and smaller of the two input values, respectively.

Step 2. Divide M by N, and call the remainder R.

Step 3. If R is not 0, then assign M the value of N, assign N the value of R, and return to step 2; otherwise, the greatest common divisor is the value currently assigned to N.

A brief history of CS

The very roots of CS

The great thinkers of CS are:

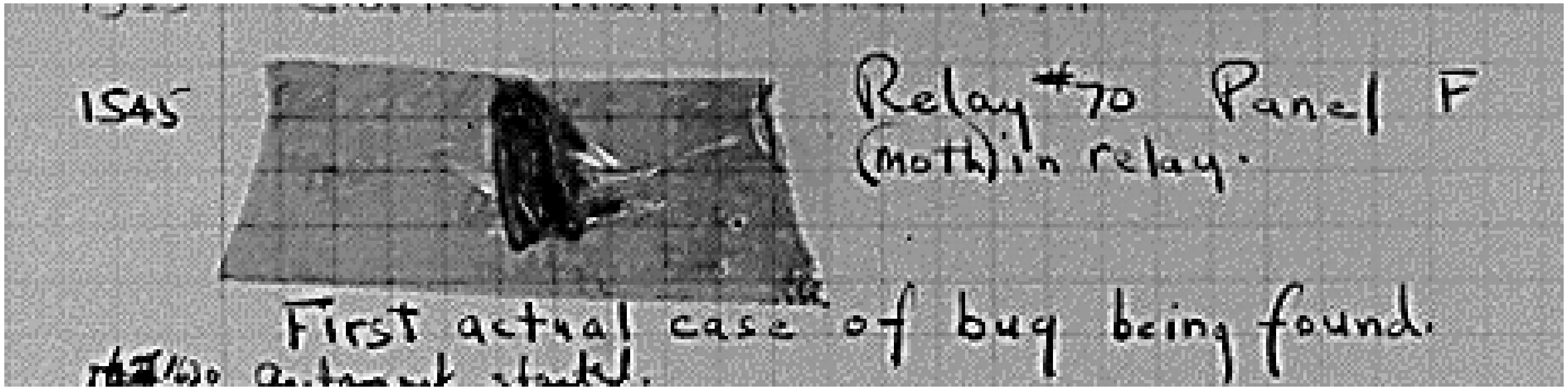
- Euclid, 300 BC
- Bhaskara, 6th century
- Al Khwarizmi, 9th century
- Fibonacci, 13th century
- Babbage, 19th century
- Turing, 20th century
- von Neumann, Knuth, Karp, Tarjan, …

The roots of computing

- 1945: John von Neumann defines his architecture for an "automatic computing system"
  - Basis for architecture of modern computing
    - Computer accepts input
    - Processes data using a CPU
    - Stores data in memory
      - Stored program technique, storing instructions with data in memory
    - Produces output

The roots of computing

- When we say there is a “bug” in the program, we mean it doesn’t work right… the term originated from an actual moth found in the UNIVAC by Grace Hopper



1545 Relay #70 Panel F (moth) in relay.

First actual case of bug being found.

1630 [illegible] started.

Minicomputer Era

- Made possible by DEC and Data General Corporation, IBM
- Medium-sized computer, e.g. DEC-PDP
- Much less expensive than mainframes, computing more accessible to smaller organizations
- Used transistors with integrated circuits

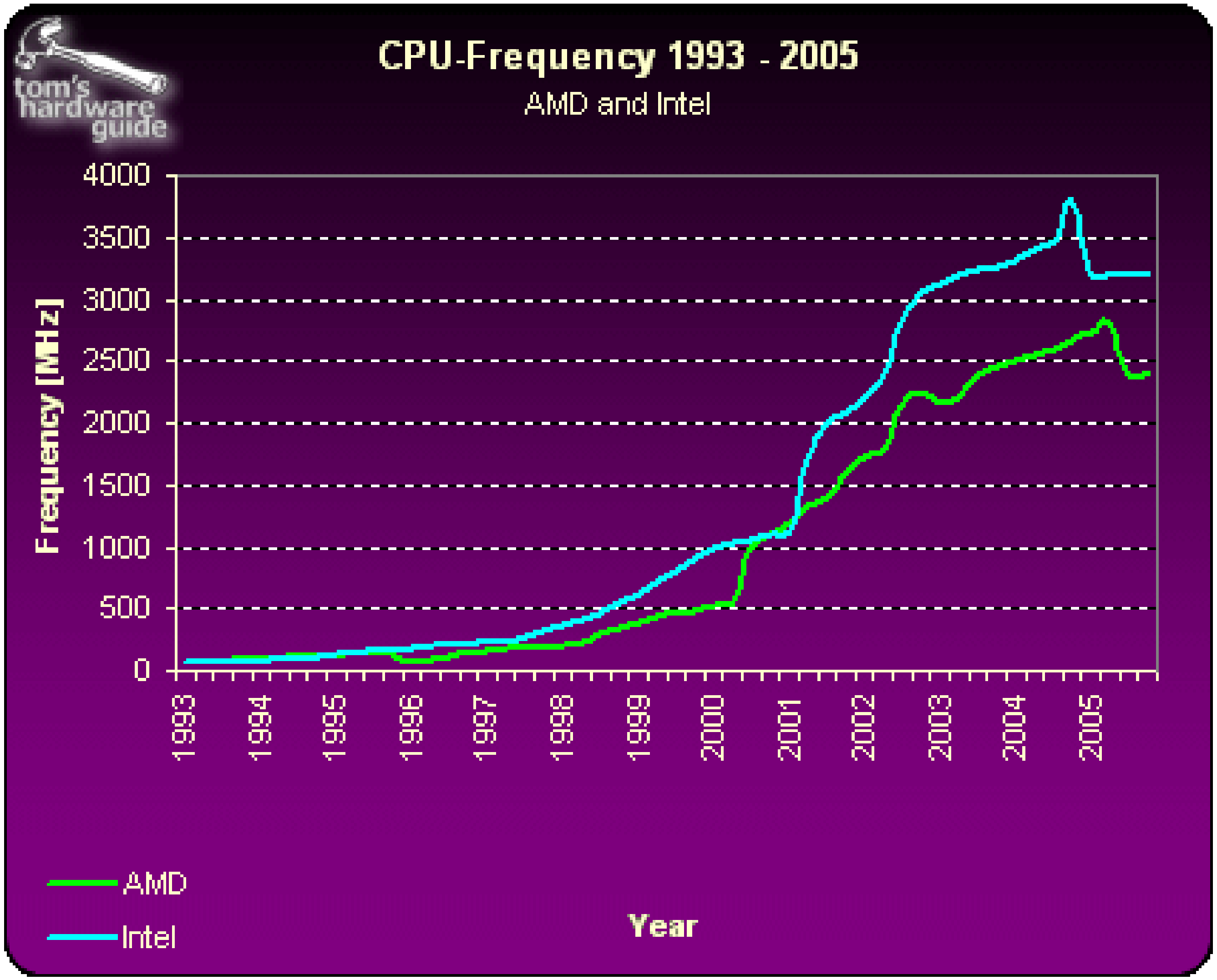
Personal Computer Era

- First microprocessor, Intel 4004 in 1971
- MITS Altair “kit” in 1975
- Apple in 1976
- IBM PC in 1981 using 8086
- Macintosh in 1984, introduced the GUI (Graphical User Interface) we still use today

Today: Internetworking/Cloud Era?

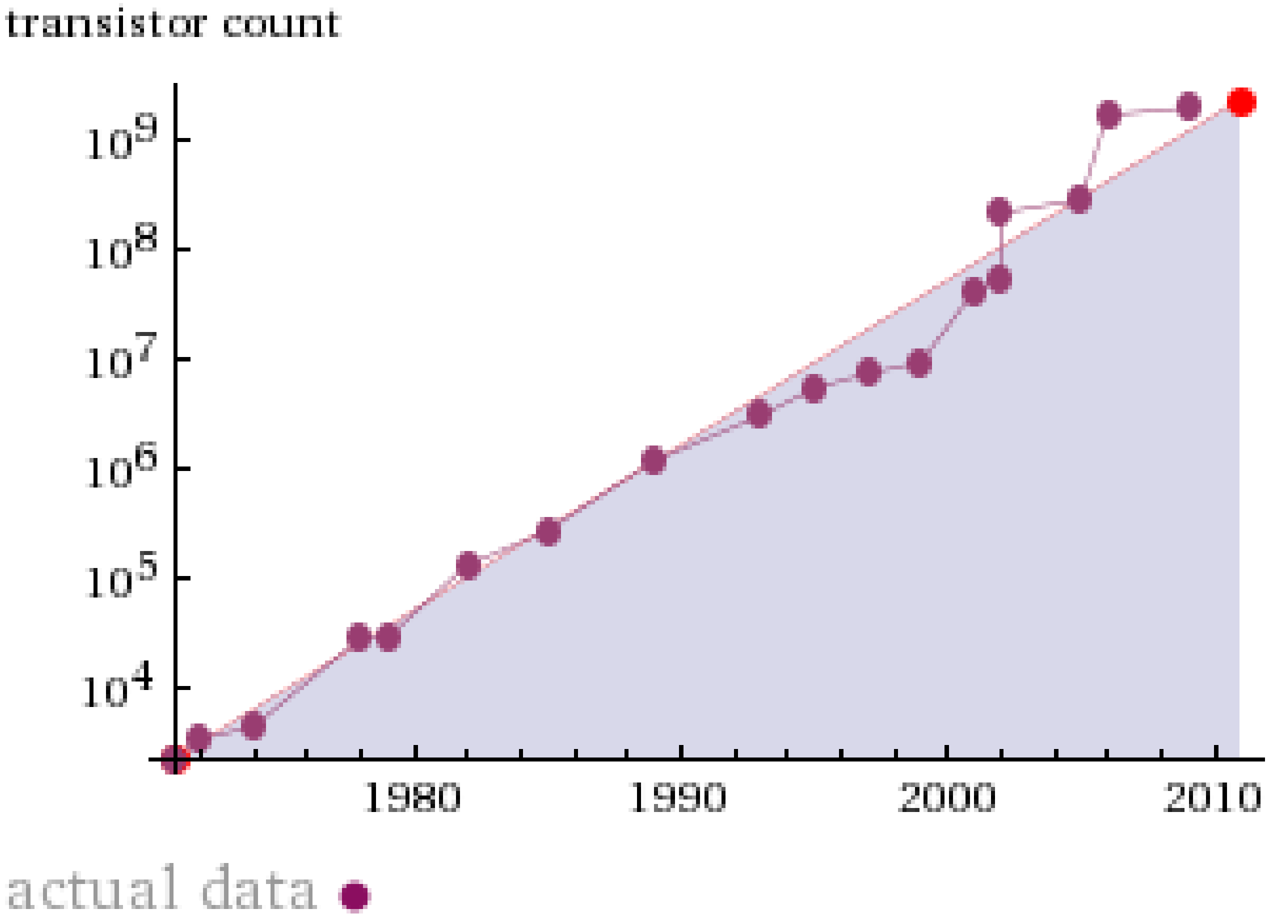
- Computer as communication device across networks
- World Wide Web, Internet
- Publishing, data sharing, real-time communications
- Seamless devices

CPU clock speeds

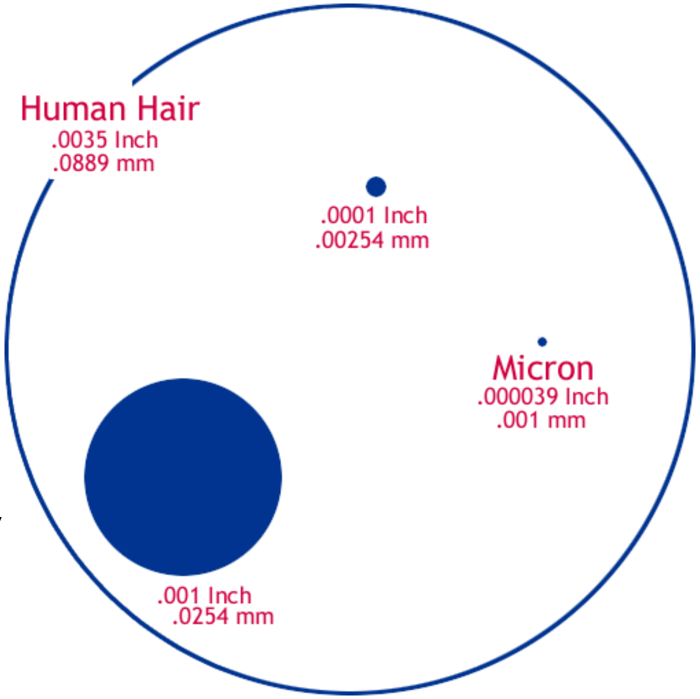


Moore‛s law

- 1965: Computing power doubles ~ every 18 months



Size matters?



- Human Hair: 100 microns wide
  - 1 micron is 1 millionth of a meter
- Bacterium: 5 microns
- Virus: 0.8 microns
- Early microprocessors: 10-15 micron technology
- 1997: 0.35 Micron
- 1998: 0.25 Micron
- 1999: 0.18 Micron
- 2001: 0.13 Micron
- 2003: 0.09 Micron
- 2007: 0.065 Micron
- 2009: 0.045 Micron
- 2017: 0.014 Micron
- Physical limits ?

Outline of the course

General tracks in CS (computer science)

- Theoretical computer science
  - Algorithms and data structures
  - Theory of computation
  - Programming language theory
  - Information and coding theory
  - Formal methods
- Applied computer science
  - Computer architecture
  - Computer networks
  - Computer graphics
  - Computer intelligence (AI)
  - Databases
  - Software engineering

General tracks in CS (computer science)

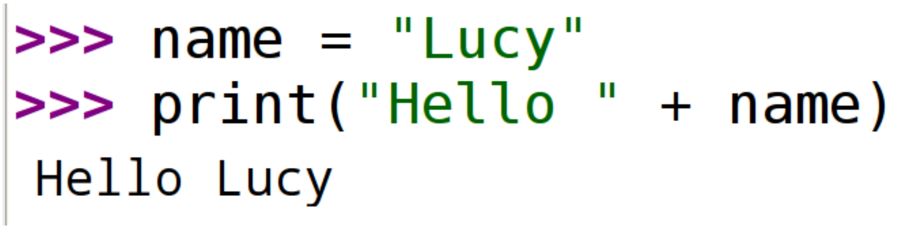
- Theoretical computer science
  - **Algorithms and data structures**
  - **Theory of computation**
  - Programming language theory
  - Information and coding theory
  - Formal methods
- Applied computer science
  - Computer architecture
  - Computer networks
  - Computer graphics
  - Computer intelligence (AI)
  - Databases
  - **Software engineering**

Covered in this course!

Outline of the lecture

Week	Slot1	Slot2
1	Introduction, Administrative issues	Computer Architecture, Algorithms, Programming
2	Python: Overview, Syntax, Simple datatypes	Python: Execution, Debugging, etc.
3	Python: Loops	Python: Composite data structures
4	Python: Functions, Recursion	Python: Files, Exceptions
5	Complexity Analysis: Big Oh	Complexity Analysis: Recursion Trees
6	Complexity Analysis: Solving Recurrences	Complexity Analysis: Master method
7	Sorting: Problem, Naive solution	Sorting: Lower bounds
8	Sorting: Mergesort	Sorting: Quicksort
9	Search trees: Concept, Binary Search	Search trees: Complexity
10	Searching: Hashing	Searching: Hashing
11	Graphs: Notation, DFS	Graphs: BFS, Djikstra
12	Graphs: Strongly connected components	Graphs: Minimum cuts
13	String matching: Overview, Problem	String matching: Solution methods
14	Greedy: Overview, Exchange arguments	Greedy: Minimum spanning trees
15	Greedy: Prim/Kruskal	Dynamic Programming: Bellman Ford
16	Dynamic Programming: Floyd Warshall	Summary

Overview: Python



```
>>> name = "Lucy"
>>> print("Hello " + name)
Hello Lucy
```

- We will use Python to illustrate different programming concepts this semester:
  - My examples will be written in Python
  - Your assignments will be written in Python
- Some advantages of Python
  - Free
  - Powerful, many extensions
  - Widely used (Google, NASA, Yahoo, Electronic Arts, Linux operating system scripts etc.)
- What you will learn in this course?
  - Designing and implementing small programs yourself
  - Automatize repetitive tasks
  - Maintaining your program, finding bugs

Overview: Complexity Analysis

- Problems can be put into complexity classes
  - There are simple problems and rather (provably) hard problems
  - Examples?
- Algorithms also come with worst-case complexities
  - There are efficient and inefficient algorithms
  - Examples?
- Combining hard problems with inefficient algorithms for solving them leads to unfeasible computations
- What you will learn in this course?
  - Compute the inherent complexity of a problem
  - Estimate the actual complexity of a solution algorithm
  - Taking into account scalability from the beginning

Overview: Sorting

- Given a list of elements, say L=[1,5,3,7,9,8,4], rearrange the list in increasing order
  - Obvious result: [1,3,4,5,7,8,9]
- Sorting if often used in CS education for learning about algorithms
  - Easy to understand, many different algorithms for solving
- What you will learn in this course?
  - The strengths and weaknesses of multiple sorting algorithms
  - How to implement sorting algorithms efficiently
  - Understanding list data structures
  - Understanding upper and lower bounds for complexity

Overview: Searching

- Given a list of elements, say L=[1,5,3,7,9,8,4], check whether the list contains an element, say 8
  - Obvious result: yes!
- Searching is another frequently used operation in many programs, with different levels of complexity
  - Examples?
- What you will learn in this course?
  - The strengths and weaknesses of multiple search algorithms
  - Finding elements in lists in less than linear time
  - Using tree data structures
  - Understanding upper and lower bounds for complexity

Overview: Graphs

- Model elements and their relationships as a graph (or network)
  - Examples?
- A fundamental data structure in CS applications
- Almost anything can be understood as a network (nowadays)
- What you will learn in this course?
  - How to model problems using graphs
  - Implementing graph algorithms efficiently
    - E.g., exploration, shortest paths, weakly/strongly connected components

Overview: String matching

- Given a string, say "<u>Today, Thomas wants to eat pizza</u>", find all occurrences of substring "<u>Thomas</u>"
- More general: The goal is to find words (approximately) in documents
- Approximately?
  - How about "<u>Today, Tomas wants to eat pizza</u>"?
- What you will learn in this course?
  - Understand the basic algorithms for string matching
  - Distinguish their strengths and weaknesses
  - How to use these algorithms in your own programs

Overview: Greedy algorithms

- Greedy algorithms: Aim to find a globally optimal solution to a problem, while always choosing a locally optimal solution at each step
  - Example?
- What you will learn in this course?
  - Understand the concept of greedy algorithms
  - Analyze two problem examples in detail
  - Get an intuition about when to use greedy algorithms (and when not to use them)

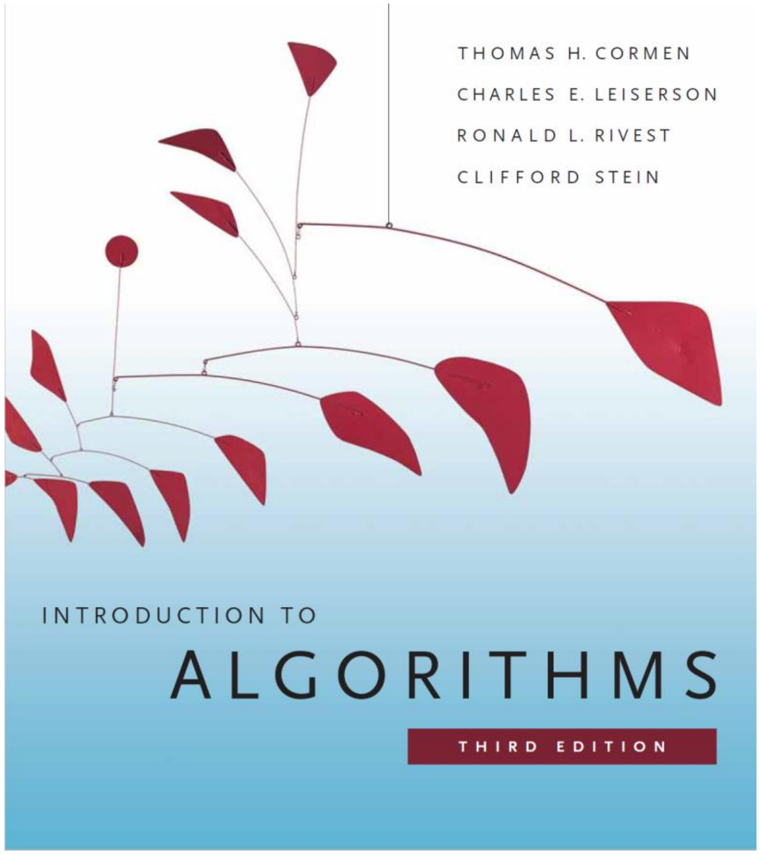
Overview: Dynamic programming

- Dynamic programming is a design paradigm
  - Solve large problems by breaking it down into smaller sub-problems
  - Once you have to solve the same sub problem again, simply look up the result of a previous computation, instead of recomputation
- What you will learn in this course?
  - Understand the concept of dynamic programming
  - Analyze two problem examples in detail
  - Get an intuition about when to use dynamic programming (and when not to use them)

Administrative issues

Course material

- The course is mainly designed around the sections in the book below:



THOMAS H. CORMEN
CHARLES E. LEISERSON
RONALD L. RIVEST
CLIFFORD STEIN

INTRODUCTION TO
ALGORITHMS
THIRD EDITION

Administrative

- How frequently meet for lab classes?
  - Mixture of programming tasks and paper-and-pencil
- Examination type:
  - TBD, depends on number of students
  - Probably closed-book examination
- Final score:
  - Based on Lecture/Lab class attendance (30%) and performance in the final examination (70%)

Advices

- The course starts quite easy going, but becomes quite involved after the first 4 weeks
- Reading the course book (“Introduction to Algorithms”) is not mandatory, but can improve your understanding significantly
  - For more background on Python just refer to the Internet

What should you know after this lecture?

- This course targets to teach you on the intersections of:
  - Algorithms, data structures, and programming
- Computer Science is not about computers only
- Algorithms are at the core of many CS concepts
- Algorithms have properties
  - Correctness
  - Efficiency / Time complexity
- It is, in general, useless to design bad algorithms
- Algorithms (abstract) vs. implementations (concrete)

What will you learn in the next few lectures?

- During the next few weeks we will look at the programming language Python
  - Understand Syntax and Semantics
  - Design small programs by yourself
  - Debug and maintain programs
- Afterwards we will start with the theoretical part

- Next week: No lecture, because of national holiday

Thank you very much