Programming on one slide

- TRUE/FALSE, boolean operations
- Data: 3, 7, 2.47, 333e-02, ‘a’, “text data”, address ...
- Operations: + - * / >= ...
- Variables: i, j, sum, A[5], B[7,8], a[i].surname, ...

- if-then-else, for/while, ...
- function()
- Input/output
- Standard libraries: java.lang.Math.sqrt()

Goals

- To learn the main concepts and techniques of the algorithm design and analysis – the practical skills and basic theoretical basis
- To be able to choose, (design,) analyze and compare algorithms and data structures
- To learn to learn, use wisely, solve, read, write, and present

Algorithms = basics of CS education

- In your future professional life people EXPECT you to know elementary data structures, algorithms, and be able to think in higher categories.
- We need to develop the language to talk about high-level goals, and you need to be able to implement them – if so needed
- At work no-one is going to teach you mathematics or fundamentals anymore

Algorithms

- Al-Khwārizmī


Example

- Wind has blown away the +, *, (, ) signs
- What’s the maximal possible value?
- Minimal?

\[
\begin{array}{cccccc}
2 & 1 & 7 & 1 & 4 & 3 \\
\end{array}
\]
- 2 1 7 1 4 3
- \((2+1)*7*(1+4)*3 = 21*15 = 315\)
- \(2*1 + 7 + 1*4 + 3 = 16\)

Q: How to maximize the value of any expression?

- 2 4 5 1 9 8 1 2 1 9 8 7 2 4 1 1 2 3 = ?

School mathematics: 11*13

But Al Khwarizmi knew another way to multiply, a method which is used today in some European countries. To multiply two decimal numbers \(x\) and \(y\), write them next to each other, as in the example below. Then repeat the following: divide the first number by 2, rounding down the result (that is, dropping the .5 if the number was odd), and double the second number. Keep going till the first number gets down to 1. Then strike out all the rows in which the first number is even, and add up whatever remains in the second column.

\[
\begin{array}{c}
11 & 13 \\
5 & 26 \\
2 & 52 \text{ (strike out)} \\
1 & 104 \\
\end{array}
\]

143 (answer)

13 x 11

1 1 0 1 \\
\times 1 0 1 1

1 1 0 1 \text{ (1101 times 1)}
1 1 0 1 \text{ (1101 times 1, shifted once)}
0 0 0 0 \text{ (1101 times 0, shifted twice)}
+ 1 1 0 1 \text{ (1101 times 1, shifted thrice)}
\hline
1 0 0 1 1 1 1 \text{ (binary 143)}

\text{Q (n^2)}

Analysis of algorithms

- Theoretical study of computer-program performance and resource usage.
- What else is important than performance?
  - Modularity
  - Correctness
  - Maintainability
  - Functionality
  - Robustness
  - User-friendliness
  - Programmer time
  - Simplicity
  - Extensibility
  - Reliability

- School mathematics: 11*13 is the same as:

\[
\begin{array}{c}
11 & 13 \\
5 & 26 \\
2 & 52 \text{ (strike out)} \\
1 & 104 \\
\end{array}
\]

143 (answer)
Why study algorithms and performance?

• Algorithms help us to understand **scalability**.
• Performance often draws the line between what is feasible and what is **impossible**.
• Algorithmic mathematics provides a **language for talking about program behavior**.
• Performance is the **currency of computing**.
• The lessons of program performance generalize to other computing resources.
• Speed is fun!

Steve Jürvetson on Skype:

• “I remember wondering: how can they be so good?” he told me, speaking about the Estonian core of Skype. “How can such a small group can do so much so quickly, compared to typical development efforts in, for example, Microsoft? I had the impression that maybe coming out of a time of Soviet occupation, when computers were underpowered, you had to know how to really program, effectively, parsimoniously, being very elegant in sculpting the programming code to be tight, effective, and fast. [That’s] not like in Microsoft, which has a very lazy programming environment, where programs are created that have memory leaks and all sorts of problems, that crash all the time and no one really cares—because it’s Microsoft!”


---

**Contents**

• Algorithms and Data structures – basics
  • Abstract data types (+ STL, LEDA, ...)
  • Lists, trees, heaps, graphs (TSP; shortest paths, diameter, ...)
  • Analysis of algorithm complexity
    – $O(n \log n)$, $o(n \log n)$, $\Omega(n \log n)$
    – recurrences $T(n) = 2T(n/2) + n$
    – Turing machines, NP, NP-complete, P-SPACE, ...
    – Very tough (NP) problems (“Monkey problem”; unsolvable problems?)

---

**Contents (2)**

• Basic designs: Dynamic programming, divide and conquer, randomised algorithms, ...
• Deterministic or non-deterministic
• Exact or approximate
• Heuristics (approximations):
  – Greedy Algorithms (packing, set cover, weighted set cover, ...)
  – local search, tabu search
  – Simulated annealing (e.g. TSP),
  – Genetic algorithms (?), ...
  – Monte Carlo, Las Vegas

---

**Contents (3)**

• Sorting, search, dictionary, ...
• Text algorithms
• Numerical algorithms
• Computational geometry

• Online algorithms vs offline
• Parallel algorithms

---

>10 km
Skills/goals

- Bit-level understanding of low-level implementation
- Creative use of existing implementations
- Formal analysis, complexity theory
- Typical heuristics and algorithm design elements

\[
\begin{align*}
\text{SUM}(10 & 7988284666294625736265536 \\
& 98373937710798828466629462573 \\
& 626553698373931117710798828466 \\
& 629462573626553619831111339377 \\
& 107988284666294625736265536983 \\
& 73937710798182666211946257362 \\
& 655369837319377107988284666294 \\
& 6257362655361739377107988284 \\
& 66629462573165536983739377) = ?
\end{align*}
\]

• 67 + 56 = ?

Answer

• 13,000

- Is checking faster than finding the solution?
SAT satisfiability

- Can a boolean formula with variables instantiated to become “TRUE”
  - \((A \lor B) \land (\neg A \lor B)\)
- Check if (some) answer satisfies the formula
  - very quick as compared to finding an answer

Subset sum

- Does some subset of a set sum up to 0?
  - Verifier / witness / certificate
  - Easy to check if someone points out...
  - \(\{2, -4, 3, 8, -1, 9, -6\}\)
- If verification in P time, then problem in NP
- Complementary: prove that none exists!
  - co-NP
  - Every subset is non-0
Main data models

- Linear
- Graph
- Tree
- Relational data model

All fitted on **linear** RAM, disk, tape

Programming languages:

- **Pseudocode**
  - primarily this course will use pseudocode
- Pointers, arrays, memory handling
  - Low- and high level abstractions
- C/C++
- Java, Python, perl, go, ...
  - Explicit data structures and algorithms
- Whichever choice: **you must be able to explain it**
- Clarity and simplicity is a key

“The Textbook”

- **Introduction to Algorithms**, Second Edition
  - Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein
  - http://mitpress.mit.edu/algorithms/
  - http://projects.csail.mit.edu/clrs/
  - http://books.google.com/books?id=HucYVYpFt7M

  - **David Harel, Yishai Feldman**
  - Addison Wesley; 3 edition (June 11, 2004)
  - http://books.google.com/books?id=kx0kRFEYkLC
  - http://www.wisdom.weizmann.ac.il/~dharel/algorithmics.html
- **Foundations of Computer Science: C Edition**
  - Alfred V. Aho, Jeffrey D. Ullman
  - W. H. Freeman (October 15, 1994)

- **The Art of Computer Programming (TAOCP)**
  - Donald E. Knuth.

- **The Algorithm Design Manual**
  - Steven S. Skiena

- **Algorithms [ILLUSTRATED]**
  - Sanjoy Dasgupta, Christos Papadimitriou, Umesh Vazirani

- **Concrete Mathematics: A Foundation for Computer Science (2nd Edition) (Hardcover)**
  - Ronald L. Graham, Donald E. Knuth, Oren Patashnik
  - Hardcover: 672 pages
  - Addison-Wesley Professional;
  - 2 edition (March 10, 1994)

  - Robert Sedgewick
  - Safari Bookshelf ! (UT)

- **Parallel algorithms**
  - An Introduction to Parallel Algorithms Joseph Jaja

- **Heuristics**
  - How to Solve It: Modern Heuristics, By Zbigniew Michalewicz, David B. Fogel
  - Edition 2, Illustrated Published by Springer, 2004

- **Randomized algorithms**

- **Complexity theory**
  - This book is a classic, developing the theory, then cataloging many NP-Complete problems.
Wikipedia:

- ...