Theory I

Albert-Ludwigs-Universität Freiburg

Prerequisites of Theory I

- Programming language, such as C++
- Basic knowledge on data structures, algorithms, logic and mathematics

Course goal

- Get an in-depth knowledge on:
  - design and analyse algorithms
  - the key problems underlying the foundations of programming languages and database systems
- Improve the problem solving ability by doing the exercises

Course load

- One exercise sheet every week
  - exercises posted one week ahead of the lectures about the topic of the exercises
  - exercise classes take place on each Monday (one hour)
  - hand in your solution before the exercise class starts or at the designated lockbox in building 51
- Final exam
- Web page for the lecture:

  https://swt.informatik.uni-freiburg.de/teaching/SS2012/theoryI

Material

Area	Topics
Algorithms and complexity	Search trees, AVL trees
	Hashing chaining, Hashing open addressing
	Dynamic tables, amortized analysis
	Randomized algorithms, primality testing
	Text search
	Edit distance
Principles of programming languages	Basic terms
	Abstract data types
	The word problem
Foundations of database systems	Relational algebra
	Relational calculus
	Formal design

Algorithms and complexity:
1 *Introduction*

Albert-Ludwigs-Universität Freiburg

Problem, algorithm

- Example: sorting problem
  - input: sequence of n numbers $< a_1, a_2, \dots, a_n >$
  - output: permutation $< a'_1, a'_2, \dots, a'_n >$ such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$
- An algorithm describes
  - solution to a problem
  - how to get from the input to the output by a sequence of basic operations
- Length of sequence of basic operations depends on
  - size of the input
  - structure of the input

Applications of algorithms

- The Internet enables people all around the world to quickly access and retrieve large amounts of information. In order to do so, clever algorithms are employed to manage and manipulate this large volume of data.
- Electronic commerce enables goods and services to be negotiated and exchanged electronically. Public-key cryptography and digital signatures core technologies used and are based on numerical algorithms and number theory.
- The Human Genome Project has the goals of identifying all the 100,000 genes in human DNA, determining the sequences of the 3 billion chemical base pairs that make up human DNA, storing this information in databases, and developing tools for data analysis.

Analysis of algorithms

- Issues:
  - correctness
  - time efficiency
  - space efficiency

- Approaches:
  - theoretical analysis
  - empirical analysis

Best case, average case, worst case

Cost $C_{...}(n)$ = number of computation steps for inputs of size n

- Worst case: $C_{worst}(n)$ = maximum cost over inputs of size n
- Best case: $C_{best}(n)$ = minimum cost over inputs of size n
- Average case: $C_{avg}(n)$ = average cost over inputs of size n

Example: sequential search

ALGORITHM $SequentialSearch(A[0..n-1], K)$

//Searches for a given value in a given array by sequential search
//Input: An array $A[0..n-1]$ and a search key K
//Output: The index of the first element of A that matches K
//   or -1 if there are no matching elements
$i \leftarrow 0$
while $i < n$ **and** $A[i] \neq K$ **do**
    $i \leftarrow i + 1$
if $i < n$ **return** i
else return -1

- Worst case?
- Best case?
- Average case?

Type of formula for basic operations's count

- Exact formula

  e.g., $C(n) = n(n-1)/2$

- Formula indicating order of growth with specific multiplicative constant

  e.g., $C(n) \approx 0.5\, n^2$

- Formula indicating order of growth with unknown multiplicative constant c

  e.g., $C(n) \approx cn^2$

Order of growth

- Order of growth of a cost function with input size n

- Example,
  - How much faster will algorithm run on computer that is twice as fast?

  - How much longer does it take to solve problem of double input size?

Values of some important function as *n*

n	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n	$n!$
10	3.3	10^1	$3.3 \cdot 10^1$	10^2	10^3	10^3	$3.6 \cdot 10^6$
10^2	6.6	10^2	$6.6 \cdot 10^2$	10^4	10^6	$1.3 \cdot 10^{30}$	$9.3 \cdot 10^{157}$
10^3	10	10^3	$1.0 \cdot 10^4$	10^6	10^9		
10^4	13	10^4	$1.3 \cdot 10^5$	10^8	10^{12}		
10^5	17	10^5	$1.7 \cdot 10^6$	10^{10}	10^{15}		
10^6	20	10^6	$2.0 \cdot 10^7$	10^{12}	10^{18}		

Table 2.1 Values (some approximate) of several functions important for analysis of algorithms

Asymptotic order of growth

A way of comparing functions that ignores constant factors and small input sizes

- O(*g*(*n*)): class of functions *f*(*n*) that grow <u>no faster</u> than *g*(*n*) e.g., $10\mathrm{n} \in O(n^2)$, $2^{2n} \notin O(2^n)$
- Θ(*g*(*n*)): class of functions *f*(*n*) that grow <u>at same rate</u> as *g*(*n*) e.g., $2^{32}n^3 \in \Theta(n^3)$, $2^{32}n^3 \notin \Theta(n^2)$
- Ω(*g*(*n*)): class of functions *f*(*n*) that grow <u>at least as fast</u> as *g*(*n*) e.g., $n^3 \in \Omega(n^2), n \notin \Omega(n^2)$

Useful summation and formula rules

$\sum_{l \leq i \leq n} 1 = 1+1+\ldots+1 = n - l + 1$

In particular, $\sum_{1 \leq i \leq n} 1 = n - 1 + 1 = n \in \Theta(n)$

$\sum_{1 \leq i \leq n} i = 1+2+\ldots+n = n(n+1)/2 \approx n^2/2 \in \Theta(n^2)$

$\sum_{1 \leq i \leq n} i^2 = 1^2+2^2+\ldots+n^2 = n(n+1)(2n+1)/6 \approx n^3/3 \in \Theta(n^3)$

$\sum_{0 \leq i \leq n} a^i = 1 + a + \ldots + a^n = (a^{n+1} - 1)/(a - 1)$ for any $a \neq 1$

In particular, $\sum_{0 \leq i \leq n} 2^i = 2^0 + 2^1 + \ldots + 2^n = 2^{n+1} - 1 \in \Theta(2^n)$

$\sum(a_i \pm b_i) = \sum a_i \pm \sum b_i \qquad \sum c a_i = c \sum a_i$

$\sum_{l \leq i \leq u} a_i = \sum_{l \leq i \leq m} a_i + \sum_{m+1 \leq i \leq u} a_i$