data structures and algorithms
2018 09 03
lecture 1
overview

- practicalities
- introduction
- sorting
- insertion sort
overview

- practicalities
- introduction
- sorting
- insertion sort
who and when

lectures
Mondays 13.30-15.15 and Thursdays 11.00-12.45
Femke van Raamsdonk
f.van.raamsdonk at vu.nl, T446

exercise classes
Tuesdays 13.30-15.15 and Fridays 11.00-12.45
four groups
Bogdan Ghita, Geoffrey Frankhuizen, Ionut Boicu, Petar Vukmirovic
in which group are you? see canvas!
which group is in which room? see canvas!
excellent book! Introduction to Algorithms by Cormen, Leiserson, Rivest, Stein

slides via canvas

exercise sheets via canvas (most exercises from the book)

some solutions via canvas
exam and grade

**midterm** in week 4 of the course

**exam** in week 8 of the course

**resit** of the exam in January

midterm is recommended but not obligatory

it contributes for 25% to the grade if better than the exam
contact

email at f.van.raamsdonk at vu.nl

refer to the course in the subject

no email via canvas

some things can wait
overview

- practicalities

- introduction

- sorting

- insertion sort
some problems cannot be solved

some problems cannot be solved efficiently

some problems can be solved efficiently

for some problems we do not know whether they can be (efficiently) solved

if $P \neq NP$ then the NP-complete problems cannot be efficiently solved
what this course is about

we will study basic data structures and analysis and design of algorithms

prerequisite: elementary programming

but this is not a programming course

prerequisite: elementary (discrete) mathematics and graph theory

but this is not a pure theory course
examples of algorithms

baking a cake

knitting
example of an algorithm: Euclid’s gcd

compute the greatest common divisor of two non-negative numbers $a \geq b$:

- if $b = 0$ then return $a$
- if $b \neq 0$, then compute the gcd of $b$ and $(a \mod b)$

the second line contains a recursive call
an algorithm is a list of instructions, the essence of a program

what are important aspects?

- correctness
  does the algorithm meet the requirements?

- termination
  does the algorithm eventually produce an output?

- efficiency or complexity
  how much time and memory space does it use?
complexity

algorithms that ’do’ the same may differ in performance

time complexity: how much time does the algorithm use?
time as function of the input

space complexity: how much space does the algorithm use?
space as function of the input

we focus on time complexity
how do we describe the time complexity of an algorithm?
how do we describe an algorithm?
our model

computer:
one-processor Random Access Machine (RAM)
Central Processing Unit with unlimited memory
elementary operations take constant (little) time

algorithm:
description in pseudocode, independent of specific syntax

data structure:
specification as Abstract Data Type (ADT)

(worst-case) time complexity:
(upper bound on) running time as function of the input size

theoretical analysis:
independent of hardware, software, and for all inputs
we ‘count’ elementary steps

elements of elementary steps:
comparison of two integers, addition, array assignment, ...

let $n$ be the size of the input,
and let a function of $n$ give the number of steps

compare the growth of different functions:

<table>
<thead>
<tr>
<th>$n$</th>
<th>log $n$</th>
<th>$n$</th>
<th>$n \log n$</th>
<th>$n^2$</th>
<th>$n^3$</th>
<th>$2^n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>3</td>
<td>8</td>
<td>24</td>
<td>62</td>
<td>512</td>
<td>256</td>
</tr>
<tr>
<td>16</td>
<td>4</td>
<td>16</td>
<td>64</td>
<td>256</td>
<td>4096</td>
<td>65536</td>
</tr>
<tr>
<td>32</td>
<td>5</td>
<td>32</td>
<td>160</td>
<td>1024</td>
<td>32768</td>
<td>4294967296</td>
</tr>
<tr>
<td>$10^3$</td>
<td>10</td>
<td>$10^3$</td>
<td>13000</td>
<td>$10^6$</td>
<td>$10^9$</td>
<td>$10^{300}$</td>
</tr>
<tr>
<td>$10^4$</td>
<td>13</td>
<td>$10^4$</td>
<td>10^5</td>
<td>$10^8$</td>
<td>10^12</td>
<td>10^{3000}</td>
</tr>
<tr>
<td>$10^5$</td>
<td>20</td>
<td>$10^5$</td>
<td>10^6</td>
<td>$10^{10}$</td>
<td>10^{15}</td>
<td>10^{3000}</td>
</tr>
</tbody>
</table>
elementary steps matter for time

elementary steps (operations) are performed really fast nevertheless they are relevant for the time complexity assumption: our computer performs $10^9$ operations per second

compare the time

<table>
<thead>
<tr>
<th>number of operations</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^3$</td>
<td>0.0001s</td>
</tr>
<tr>
<td>$10^6$</td>
<td>0.001s</td>
</tr>
<tr>
<td>$10^7$</td>
<td>0.01s</td>
</tr>
<tr>
<td>$10^8$</td>
<td>0.1s</td>
</tr>
<tr>
<td>$10^{12}$</td>
<td>16.7min</td>
</tr>
<tr>
<td>$10^{14}$</td>
<td>27h47min</td>
</tr>
<tr>
<td>$10^{18}$</td>
<td>33yr</td>
</tr>
</tbody>
</table>
approach: worst-case asymptotic time complexity

we write algorithms in pseudocode

we give a function $T(n)$ that gives the time complexity for input size $n$

we want to know what happens to $T$ if $n$ becomes (very) large in worst-case scenario
overview

- practicalities
- introduction

- sorting
- insertion sort
sorting: specification

**input**: a finite sequence of elements

**output**: an ordered or sorted permutation of the input-sequence

elements are usually integers or natural numbers

sorted means usually in *increasing order with respect to* \( \leq \)

a sequence is usually given by an array

an element may occur more than once

the input-sequence is assumed to be an array

convention (of the book): an array starts at index 1
insertion sort: idea and example

you may think of sorting a hand of cards

the sequence consists of a sorted part followed by a non-sorted part

initially: the sorted part consists only of the first element

loop: while the non-sorted part is non-empty
insert (how?) the first element of the non-sorted part
in the correct position of the sorted part

[5, 3, 4, 7, 1]
[3, 5, 4, 7, 1]
[3, 4, 5, 7, 1]
[3, 4, 5, 7, 1]
[1, 3, 4, 5, 7]
Algorithm insertionSort(A, n):

for $j := 2$ to $n$ do

key := $A[j]$

$i := j - 1$

while $i \geq 1$ and $A[i] > key$ do

$A[i + 1] := A[i]$

$i := i - 1$

$A[i + 1] := key$
insertion sort: worst-case time complexity

test for-loop: \( n \)

assignment \( key: n - 1 \)

assignment \( i: n - 1 \)

worst case: \( A[i] > key \) always succeeds

for fixed \( j \): we do \( j \) times the while-test
and \( \sum_{j=2}^{n} j = \frac{1}{2}n(n + 1) - 1 \)

for fixed \( j \): we do \( j - 1 \) times the assignment \( A[i + 1] \)
and \( \sum_{j=2}^{n} (j - 1) = \frac{1}{2}(n - 1)n \)

for fixed \( j \): we do \( j - 1 \) times the assignment \( i \)
and \( \sum_{j=2}^{n} (j - 1) = \frac{1}{2}(n - 1)n \)

assignment \( A[i + 1]: n - 1 \) times
insertion sort: worst-case time complexity

so $T(n)$ is of the form $an^2 + bn + c$ for some $a, b, c$

we have $an^2 \leq T(n)$

we can find a positive real number $d$ such that $T(n) \leq dn^2$ for large $n$

this is the intuition of $T$ is in $\Theta(n^2)$

eventually the growth of $T$ is similar to the growth of $n^2$
insertion sort: order of growth

we do not care about the exact time an elementary step takes

we are moreover mainly interested in the rate of growth of the running time

the time needed for insertion sort is quadratic in input size $n$

so worst-case complexity in $\Theta(n^2)$

this is an asymptotic approximation where constant factors become irrelevant
summary

book more or less chapters 1 and 2

we continue with chapter 2 in lecture 2

slides lecture 1 via canvas

exercise sheet 1 via canvas

exercise class 1 tomorrow at 13.30

group 1 in WN-P647

group 2 in HG-08A20

group 3 in WN-S655

group 4 in HG-15A16