

datastructuren en algoritmen  
2011 09 05  
college 1

## schema

- praktische zaken
- datastructuren en algoritmen
- analyse van algoritmen
- belangrijke functies
- materiaal

## schema

- praktische zaken
- datastructuren en algoritmen
- analyse van algoritmen
- belangrijke functies
- materiaal

## wie

- **hoorcolleges:**  
Femke van Raamsdonk  
femke at cs.vu.nl  
T446
- **werkcolleges:**  
Dennis Andriesse  
Michel Degenhardt  
Patrick van Rietschoten

## hoorcolleges

- in week 36–42:
  - maandag 13.30-15.15 in M143
  - donderdag 11.00-12.45 in M143
- na 7 hoorcolleges voortentamen  
na totaal 12 colleges vragenuur (uitloop)  
dan tentamen

## practicum

- practicumopdrachten (Java) voor bonus van ten hoogste 0.5
- meer informatie volgt nog !

## werkcolleges

- dinsdag 11.00-12.45 in zalen M607 (groep 1) en C659 (groep 2)  
vrijdag 13.30-15.15 in zalen M607 (groep 1) en P647 (groep 2)
- twee groepen:
  - groep 1: IMM + LI + INF
  - groep 2: BWI + ECT
- drie werkcollegedocenten:  
Dennis Andriesse, Michel Degenhardt, Patrick van Rietschoten

## tentamen en voortentamen

- tentamen op dinsdag 25 oktober 2011, 15.15-18.00  
herkansing op woensdag 11 januari 2011, 18.30-21.15
- voortentamen op dinsdag 2011 09 27, om 11.00-12.45 (let op zalen)  
in plaats van werkcollege 7  
alleen dit jaar (2011-2012) geldig
- practicum voor bonus van ten hoogste 0.5 voor eindcijfer
- eindcijfer: hoogste van

$$\text{Tentamencijfer}, \frac{2 \times \text{Tentamencijfer} + 1 \times \text{Voortentamencijfer}}{3}$$

plus bonus

## materiaal

- boek: Algorithm Design
- webpage van het vak

## voorbeeld algoritme: recept voor cake



- **input:** ingrediënten
- **software:** het recept
- **hardware:** oven
- **output:** cake

## schema

- praktische zaken
- datastructuren en algoritmen
- analyse van algoritmen
- belangrijke functies
- materiaal

## voorbeeld algoritme: Euclid's GCD



bereken de grootste gemene deler (gcd) van twee positieve getallen  $a > b$ :

- als  $b = 0$ , dan is het  $a$
- als  $b \neq 0$ , bereken dan de gcd van  $b$  en  $(a \bmod b)$

## voorbeeld algoritme: Fibonacci's konijnen



- we starten met 1 paar konijnen
- elk konijnenpaar van ten minste 3 jaar oud krijgt per jaar een paar jongen
- ze blijven allemaal leven

$$\begin{aligned}F(1) &= 1 \\F(2) &= 1 \\F(n) &= F(n-1) + F(n-2)\end{aligned}$$

## datastructuren

- een systematische manier om de informatie te organiseren
- mogelijk cruciaal voor de efficiency van een algoritme
- we schrijven ze op als Abstract Data Type (ADT)  
(later meer hierover)

## algoritme: wat is het?

een algoritme is een lijst instructies, de essentie van een programma

belangrijke aspecten van algoritmen:

- **correctheid**  
doet het algoritme wat de bedoeling is?
- **terminatie**  
stopt het algoritme op den duur?
- **efficiency**  
hoeveel tijd en ruimte gebruikt het algoritme?  
dit bestuderen we in dit college !

## schema

- praktische zaken
- datastructuren en algoritmen
- analyse van algoritmen
  - experimenteel
  - stappen tellen
  - asymptotische grenzen
- belangrijke functies
- materiaal

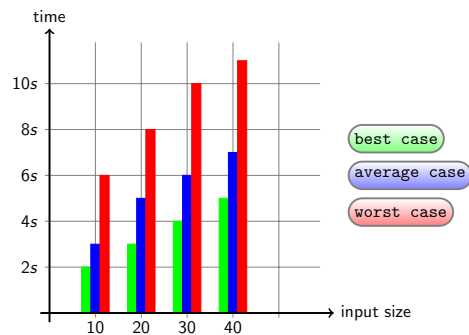
## analyse van algoritmen: complexiteit

algoritmen die hetzelfde 'doen'  
kunnen verschillen in 'hoe handig' ze dat doen

- **tijdscomplexiteit**  
hoeveel tijd gebruikt het algoritme?  
(als functie van de grootte van de input)
- **ruimte-complexiteit**  
hoeveel geheugen gebruikt het algoritme?  
(als functie van de grootte van de input)

## running time van een algoritme

benodigde tijd neemt toe naarmate de input groter is



dit college: voornamelijk worst-case tijdscomplexiteit

## tijdscomplexiteit: factoren

- **input**  
als grotere input dan (meestal) langere running time
- **hardware**  
processor, clock rate, memory, disk, ...
- **software**  
operating system, programming language, compiler, interpreter
- **algoritme**

hier: running time als functie van de grootte van de input

## experimenteel: we gaan meten

gewoon met een klokje

- maar: implementatie nodig
- maar: voor vergelijkingen zelfde hardware en software nodig
- let op: zie geen belangrijke input over het hoofd
- maar: we zien niet waarom het programma snel of langzaam is
- en dan: komt er een nieuwe processor

## theoretisch: we gaan voorspellen

- running time als functie van de grootte van de input
- onafhankelijk van hardware en software
- we vergeten geen inputs
- geen implementatie nodig

## model bij stappen tellen

- **computer model:** Random Access Machine (RAM)
- **algoritme:** beschrijving in pseudo-code

## voorspellen door stappen te tellen

wat is een stap?

een stap is een primitieve operatie zoals:

- een waarde toekennen aan een variable
- een method aanroepen, return van een method
- een arithmetische operatie (+, -, <, ...)
- waarde in array opvragen

## Random Access Machine (RAM)

- CPU met geheugen
- onbegrensd aantal geheugencellen met daarin willekeurig groot getal (of karakter)
- primitieve operaties in constante tijd
- toegang tot geheugencel met primitieve operatie



## pseudo-code

lijkt op programmeertaal maar onafhankelijk van specifieke syntax

- primitieve operaties  
ofwel stappen
- control flow  
if-then-else, while, ...

## stappen tellen: vervolg voorbeeld

init:

$A[0]$  opvragen (1)

$currentMax$  waarde toekennen (1)

begin loop:

$i$  waarde 1 toekennen (1)

$i \leq n - 1$  check  $n$  keer uitvoeren ( $n - 1$  keer true, 1 keer false) ( $n$ )

loop wordt  $n - 1$  keer uitgevoerd met daarin ( $n - 1$  keer):

$A[i]$  opvragen (1)

$currentMax < A[i]$  check (1)

$currentMax$  waarde toekennen (worst-case!) (1)

$A[i]$  waarde opvragen (worst-case!) (1)

$i + 1$  uitrekenen (1)

$i$  nieuwe waarde toekennen (1)

return:  $currentMax$  retourneren (1)

## pseudo-code: een voorbeeld

**Algorithm** arrayMax( $A, n$ ):

**Input:** array  $A$  storing  $n$  integers

**Output:** the maximum element of  $A$

$currentMax := A[0]$

**for**  $i := 1$  **to**  $n - 1$  **do**

**if**  $currentMax < A[i]$  **then**

$currentMax := A[i]$

**return**  $currentMax$

## vervolg telling

- worst-case:

$$\begin{aligned} T(n) &= 2 + 1 + n + (n - 1) \cdot 6 + 1 \\ &= 7 \cdot n - 2 \end{aligned}$$

- best-case:

$$\begin{aligned} T(n) &= 2 + 1 + n + (n - 1) \cdot 4 + 1 \\ &= 5 \cdot n \end{aligned}$$

## primitieve operaties tellen: voors en tegens

- + je kunt het beste, slechtste (!), en gemiddelde (hmm) geval analyseren
- in het echt is niet elke primitieve stap even snel
- heel precies stappen tellen is lastig

## bovengrens met grote- $\mathcal{O}$ : definitie

bovengrens met  $\mathcal{O}$ :

$f(n) \in \mathcal{O}(g(n))$  als

$$\begin{aligned} \exists c > 0 \in \mathbb{R} \\ \exists n_0 > 0 \in \mathbb{N} \\ \forall n \in \mathbb{N} : n \geq n_0 \Rightarrow f(n) \leq c \cdot g(n) \end{aligned}$$

$f(n)$  is **kleiner-gelijk**  $g(n)$  **op een constante na** en **op den duur**

## orde van grootte

wij zijn geïnteresseerd in:

- running time als functie van de grootte van de input
- big picture benadering:  
een 'op den duur' afschatting met een bekende functie
- bijvoorbeeld:  $\text{arrayMax}(A, n)$  groeit evenredig met  $n$
- bijv andere hardware geeft alleen constante factor verschil

## grote- $\mathcal{O}$ : voorbeelden

- $f(n) = 10000 \in \mathcal{O}(1)$
- $f(n) = 5 \cdot n \in \mathcal{O}(n)$
- $f(n) = 5 \cdot n + 100 \in \mathcal{O}(n)$
- $f(n) = 5 \cdot n^2 \in \mathcal{O}(n^2)$
- $f(n) = 8n - 2 \in \mathcal{O}(n)$   
want  $8n - 2 \leq 8n$  voor  $n \geq 1$

## asymptotische analyse

- **worst-case aantal primitieve operaties**  
geef een functie in  $n$  voor het worst-case geval van het aantal primitieve operaties  
(bekijk de pseudo-code)
- geef voor die functie een asymptotische afschatting met de grote- $O$  notatie

## grote- $O$ : conventies

- **we geven een zo scherp mogelijke afschatting**  
 $4n^3 \in \mathcal{O}(n^5)$  is waar maar onscherp  
 $4n^3 \in \mathcal{O}(n^3)$  is goed
- **we geven een zo eenvoudig mogelijke afschatting**  
 $4n^3 \in \mathcal{O}(4n^3)$  is waar maar te complex  
 $4n^3 \in \mathcal{O}(n^3)$  is goed

## asymptotische analyse: voorbeeld

`arrayMax(A, n)` is in  $\mathcal{O}(n)$

## grote- $O$ : rekenregels

- we gebruiken (eenvoudige)stellingen, **rekenregels**, om van een samengestelde functie de grote- $O$  te bepalen.
- **voorbeeld:**  
 $n^3 \in \mathcal{O}(n^3)$  dus  $7n^3 \in \mathcal{O}(n^3)$

## ondergrens met grote-Omega: definitie

ondergrens

$f(n) \in \Omega(g(n))$  als  $g(n) \in \mathcal{O}(f(n))$

oftewel

$$\begin{aligned} \exists c > 0 \in \mathbb{R} \\ \exists n_0 > 0 \in \mathbb{N} \\ \forall n \in \mathbb{N} : n \geq n_0 \Rightarrow & g(n) \leq c \cdot f(n) \\ & \sim \\ & f(n) \geq c' \cdot g(n) \end{aligned}$$

## dubbele grens met grote-Theta: definitie

$f(n) \in \Theta(g(n))$  als

$$f(n) \in \mathcal{O}(g(n))$$

en

$$f(n) \in \Omega(g(n))$$

oftewel

$$\begin{aligned} \exists c, c' > 0 \in \mathbb{R} \\ \exists n_0 > 0 \in \mathbb{N} \\ \forall n \in \mathbb{N} : n \geq n_0 \Rightarrow c \cdot g(n) \leq f(n) \leq c' \cdot g(n) \end{aligned}$$

## meer grenzen

- **kleine-o:**

$f(n) \in o(g(n))$  als:

$$\begin{aligned} \forall c > 0 \in \mathbb{R} \\ \exists n_0 > 1 \in \mathbb{N} \\ \forall n \in \mathbb{N} : n \geq n_0 \Rightarrow f(n) \leq c \cdot g(n) \end{aligned}$$

- **kleine-omega:**

$f(n) \in \omega(g(n))$  als  $g(n) \in o(f(n))$  oftewel:

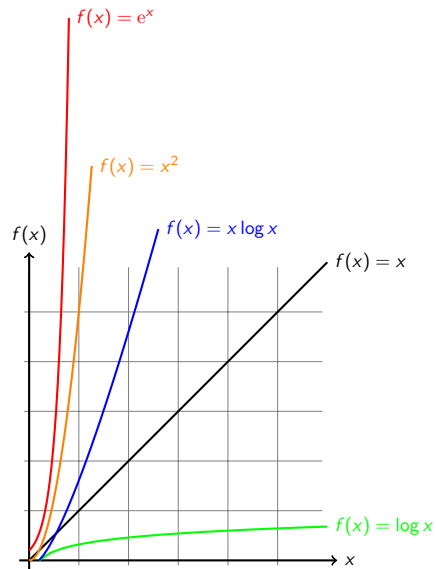
$$\begin{aligned} \forall c > 0 \in \mathbb{R} \\ \exists n_0 > 1 \in \mathbb{N} \\ \forall n \in \mathbb{N} : n \geq n_0 \Rightarrow g(n) \leq c \cdot f(n) \end{aligned}$$

## belangrijke groeifuncties

constante	$f(n) = C$
logaritmisch	$f(n) = \log n$
linear	$f(n) = n$
$(n \cdot \log n)$	$f(n) = (n \cdot \log n)$
kwadratisch	$f(n) = n^2$
cubic (en polynomiaal)	$f(n) = n^3$
exponentieel	$f(n) = b^n$

$\log_a b = c$  precies als  $a^c = b$   
tenzij anders vermeld: log is  $\log_2$

plaatje: hoe snel groeien deze functies?



tabel: hoe snel groeien deze functies?

$n$	$\log n$	$n$	$n \log n$	$n^2$	$n^3$	$2^n$
8	3	8	24	62	512	256
16	4	16	64	256	4096	65536
32	5	32	160	1024	32768	4294967296
$10^3$	10	$10^3$	13000	$10^6$	$10^9$	$10^{300}$
$10^4$	13	$10^4$	$10^5$	$10^8$	$10^{12}$	$10^{3000}$
$10^5$	20	$10^5$	$10^6$	$10^{10}$	$10^{15}$	$10^{3000}$

hoeveel tijd kost een algoritme dus?

$n$	10	100	1000	$10^4$	$10^5$	$10^6$
$\log n$	<	<	<	<	<	0.00002s
$n$	<	<	0.001s	0.01s	0.1s	1s
$n \cdot \log n$	<	<	0.013s	0.1s	1s	10s
$n^2$	<	0.01s	1s	100s	3h	1000h
$n^3$	0.001s	1s	1000s	1000h	100y	$10^5$ y
$2^n$	0.001s	$10^{23}$ y	>	>	>	>

met: 1 stap kost  $1\mu s$  (0.000001s)

< betekent (< 0.001s)

> betekent  $10^{300}$  jaar

schema

- praktische zaken
- datastructuren en algoritmen
- analyse van algoritmen
  - experimenteel
  - stappen tellen
  - asymptotische grenzen
- belangrijke functies
- materiaal

## materiaal

- boek 1.1, 1.2, 1.4, 1.6
- boek 1.3: gebruiken we wel
- boek 1.5: komt later

## extra materiaal

- wiki over de grote O notatie