Eerste college Complexiteit*

7 februari 2025

Introductie

*officieel heet dit vak Complexity

- docenten: Jeannette de Graaf en Mark van den Bergh
  j.m.de.graaf@liacs.leidenuniv.nl, Gorlaeusgebouw BM.2.09
  m.j.h.van.den.bergh@liacs.leidenuniv.nl, Gorlaeusgebouw BM.2.01

- assistentie: Pjotr Beerens, Perri van den Berg, Angela van Delft en Niels Heslenfeld

- website: liacs.leidenuniv.nl/~graafjmde/COMP/

- materiaal: dictaat met opgaven (zie website) + sheets

- Brightspace: voor inleveren huiswerkopdrachten en cijfers

- vragen: via Brightspace, e-mail of in persoon

- Rooster: zie website / MyTimetable
- hoorcollege:
  vrijdag 11.00–12.45 in Gorlaeusgebouw BW.0.40
- werkcollege:
  vrijdag 13.15–15.00 in Gorlaeusgebouw, zalen BW.0.29 en BW.0.30
- geen college en geen werkcollege op 28 maart, 18 april
- tentamen: woensdag 28 mei 2025, 09.00–12.00 uur
- hertentamen: dinsdag 1 juli 2025, 09.00–12.00 uur

- Vier* huiswerkopdrachten, individueel in te leveren
- Samenwerken mag (met mate), overschrijven niet
- Voor elke huiswerkopdracht krijg je een cijfer
- Het gemiddeld huiswerkcijfer telt mee als bonus indien het tentamencijfer $\geq 5,0$ is

*maximaal

Het eindcijfer wordt als volgt samengesteld:

if tentamencijfer $\geq 5,0$ **then**

    eindcijfer = tentamencijfer + (gemiddeld huiswerkcijfer) / 10 ;

else

    eindcijfer = tentamencijfer ;

fi

- Algoritmiek/Datastructuren (al gehad):
  $\rightarrow$ hoe los je het probleem op?

- Program Correctness (keuzevak in jaar 3):
  $\rightarrow$ is je oplossing correct?

- Computability (berekenbaarheid) (ook dit semester):
  $\rightarrow$ kunnen we dit probleem wel oplossen?

- Complexiteit:

  $\rightarrow$ Analyse van algoritmen: hoe efficiënt is je oplossing?

  $\rightarrow$ Complexiteitsklassen: $\mathcal{P}$ versus $\mathcal{NP}$

Complexiteit:

→ (tijd)complexiteit: hoeveelheid werk

→ ruimtecomplexiteit: hoeveelheid geheugen

→ optimaliteit: kan het nog beter?

Complexiteit = tijdcomplexiteit = hoeveelheid werk

→ een maat voor de hoeveelheid werk moet iets vertellen over de efficiëntie van de gebruikte methode

→ die maat moet onafhankelijk zijn van de gebruikte computer, programmeertaal, implementatiedetails etc.

→ de complexiteit hangt gewoonlijk af van de grootte van de invoer: hoe groter de invoer, hoe groter de complexiteit

→ complexiteit wordt gewoonlijk uitgedrukt als functie van de grootte van de invoer

Opmerking: we zullen hier altijd sequentiële algoritmen bekijken.

Functie van de invoergrootte

n	10	100	1.000	100.000	10.000.000
$\lg n$	3	6	9	16	23
$\sqrt{n}$	3	10	32	316	3 162
n	10	100	1.000	100.000	10.000.000
$n \lg n$	33	664	9.966	1.660.964	$\approx 10^8$
n^2	100	10.000	1.000.000	10^{10}	10^{14}
n^{10}	10^{10}	10^{20}	10^{30}	10^{50}	10^{70}
2^n	1.024	$\approx 10^{30}$	$\approx 10^{301}$	$\approx 10^{30.102}$	veel
$n!$	3.628.800	$\approx 10^{157}$	$\approx 10^{2.567}$	$\approx 10^{456.573}$	veel
n^n	10^{10}	10^{200}	$10^{3.000}$	$10^{500.000}$	$10^{70.000.000}$

Met 1.000.000 operaties per seconde duren 10^{30} operaties al ongeveer $3 \cdot 10^{16}$ jaar. De oerknal is ruim 10 miljard $= 10^{10}$ jaar geleden.

De absurd hoge getallen terzijde: bijvoorbeeld het sorteren van rijtjes is iets dat overal gebeurt en meestal voor gigantisch grote rijtjes, bijvoorbeeld bestaande uit $n = 10.000.000$ items. We zullen sorteeralgoritmen zien met een complexiteit van o.a. n^2, $n\sqrt{n}$ of $n \lg n$.

Voor $n = 10.000.000$, met 1.000.000 operaties per seconde duurt het sorteren daarmee ruwweg:

- $n \lg n \rightarrow 4$ minuten
- $n\sqrt{n} \rightarrow 9$ uur
- $n^2 \rightarrow 3$ jaar

Een sorteeralgoritme met complexiteit n^3 zou er zelfs 32 miljoen jaar over doen om één rijtje te sorteren.

Trix (66 miljoen jaar oud) zou nu net zijn begonnen aan haar derde rijtje.

(https://topstukken.naturalis.nl/object/trix)

Gegeven een array A ($A[1], A[2], \ldots, A[n]$, ongesorteerd) met $n \geq 1$ gehele getallen.
Gevraagd het maximum van deze getallen.

Een algoritme in pseudocode dat het maximum vindt:

```
(1)   max := A[1];
(2)   index := 2;
(3)   while index ≤ n do
(4)       if max < A[index] then
(5)           max := A[index];
(6)       fi
(7)       index := index + 1;
(8)   od
(9)   return max;
```

We tellen het aantal operaties:

(1)	$max := A[1]$;	1 x
(2)	$index := 2$;	1 x
(3)	**while** $index \leq n$ **do**	n x
(4)	**if** $max < A[index]$ **then**	$n-1$ x
(5)	$max := A[index]$;	$\leq n-1$ x
(6)	**fi**	
(7)	$index := index + 1$;	$n-1$ x
(8)	**od**	
(9)	**return** max;	1 x
		$(\leq)\ 4n$

aantal vergelijkingen ($max < A[index]$) $= n-1 \approx n =$ aantal vergelijkingen ($index \leq n$) $\approx 4n$ $\Theta(n)$

Om de complexiteit van een algoritme te bepalen tellen we het aantal keer dat een geschikte basisoperatie wordt uitgevoerd.

- identificeer een operatie die fundamenteel is voor het algoritme (dus geen boekhoudoperaties zoals tellerophogingen)
- het totale aantal uitgevoerde operaties moet ruwweg evenredig zijn (in orde van grootte) met het aantal basisoperaties, ofwel:
- alle andere operaties worden (in orde van grootte) hooguit even vaak uitgevoerd als de basisoperatie
- de basisoperatie is dus maatgevend voor de complexiteit van het algoritme

Basisoperatie: voorbeelden

probleem	basisoperatie (meestal)
X zoeken in een array	vergelijking van X met een array-element (en/of vergelijking tussen array-elementen onderling)
twee polynomen vermenigvuldigen	vermenigvuldiging van twee getallen (en/of optelling)
een array sorteren	vergelijking van twee array-elementen
graafprobleem	bezoek aan een knoop en/of doorlopen van een tak/pijl

De complexiteit van een algoritme hangt af van de grootte van de invoer, n: $f(n)$

- bepaal een geschikte basisoperatie

- tel het aantal keer dat de basisoperatie wordt uitgevoerd $\Rightarrow f(n)$

Interessant om te weten is hoe hard $f(n)$ groeit:

- van belang is de orde van grootte van $f(n)$ voor *grote* n $\Rightarrow O, \Theta, \Omega$

De complexiteit hangt af van de grootte van de invoer.
Wat wordt bedoeld met invoergrootte?

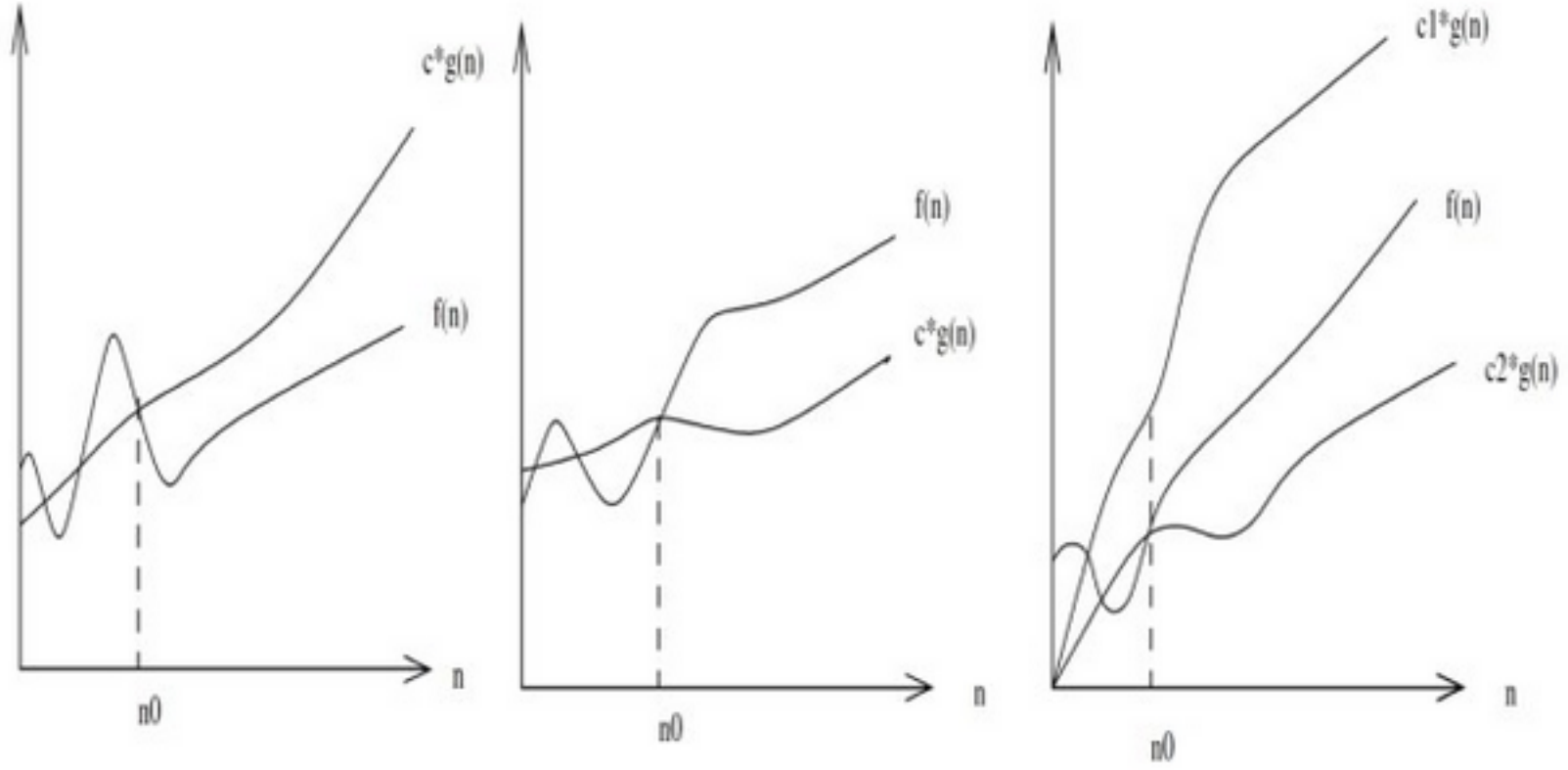
probleem	(maat voor de) grootte van de invoer
X zoeken in een array	aantal array-elementen
twee polynomen vermenigvuldigen	graad van de polynomen (= aantal coëfficiënten)
een array sorteren	aantal array-elementen
graafprobleem	aantal knopen en/of takken

$f, g : \mathbb{N} \longrightarrow \mathbb{R}$ (meestal $\mathbb{R}^+$)

1. $f \in O(g)$: er bestaan constanten c en n_0 (beide > 0) zodat $0 \leq f(n) \leq cg(n)$ voor alle $n \geq n_0$: f groeit hooguit even hard als g. asymptotische bovengrens

2. $f \in \Omega(g)$: er bestaan constanten c en n_0 (beide > 0) zodat $0 \leq cg(n) \leq f(n)$ voor alle $n \geq n_0$: f groeit minstens even hard als g. asymptotische ondergrens

3. $f \in \Theta(g)$: er bestaan constanten c_1, c_2 en n_0 (alle > 0) zodat $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ voor alle $n \geq n_0$: f groeit even hard als g. asymptotisch gedrag



$f \in O(g)$ $f \in \Omega(g)$ $f \in \Theta(g)$

Opmerking

Vaak zie je ook wel de wat slordige notatie met $f = O(g)$ in plaats van $f \in O(g)$. Hiermee wordt hetzelfde bedoeld, maar deze laatste notatie is wiskundig netter. Immers, $O(g)$ is eigenlijk een klasse van functies h die voldoen aan: $0 \leq h(n) \leq cg(n)$ vanaf zekere n.
Analoog voor Θ en Ω.

We gebruiken bij dit college dan ook de notaties $f \in O(g)$, $f \in \Theta(g)$ en $f \in \Omega(g)$.

Zie het dictaat paragraaf 2.3 voor meer.

$\frac{1}{2}n^2 - 3n \in \Theta(n^2)$ $\qquad$ $3n^3 + 6n^2 + 9 \in \Theta(n^3)$

$42n \in O(n^2)$, maar $42n \notin \Omega(n^2)$

$2^n \in O(3^n)$, maar $2^n \notin \Omega(3^n)$

$\log_7 n \in \Theta(\lg n)$ $(\lg = \log_2)$

$C \in \Theta(1)$ (voor $C > 0$) $\qquad$ $0 \in O(1)$, maar $0 \notin \Omega(1)$

$\sum_{i=1}^{n} i \in \Theta(n^2)$ $\qquad$ $\sum_{i=1}^{n} \frac{1}{i} \in \Theta(\lg n)$

$\sum_{i=1}^{n} \frac{1}{2^i} \in \Theta(1)$ $\qquad$ $\sum_{i=1}^{n} \lg i = \lg n! \in \Theta(n \lg n)$*

*voor het bewijs: zie dictaat paragraaf 2.3

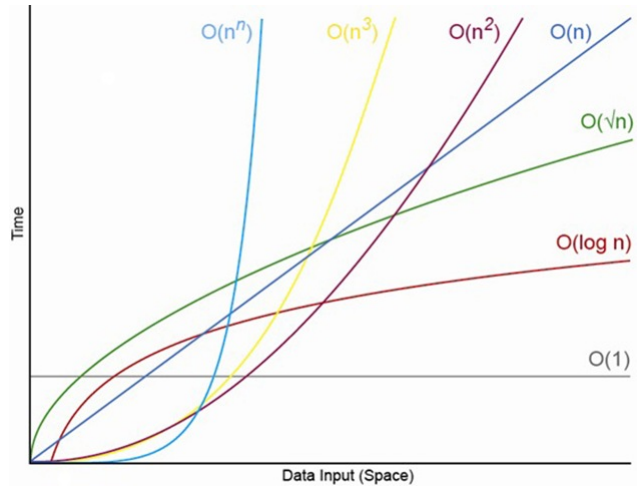
Naamgeving

$\Theta(1)$: constant

$\Theta(\lg n)$: logaritmisch

$\Theta(n)$: lineair

$\Theta(n^2)$: kwadratisch



$\Theta(n^k)$ met $k > 0$: polynomiaal

$\Theta(2^n)$, $\Theta(a^n)$ met $a > 1$: exponentieel

$\Theta(n!)$, $\Theta(n^n)$ $\cdots$: superexponentieel

Polynomiaal: (meestal) binnen redelijke tijd klaar

Exponentieel: zeker niet in acceptabele tijd klaar

n	10	20	50	100	300
n^2	$\frac{1}{10000}$ sec	$\frac{1}{2500}$ sec	$\frac{1}{400}$ sec	$\frac{1}{100}$ sec	$\frac{9}{100}$ sec
n^5	$\frac{1}{10}$ sec	3,2 sec	5,2 min	2,8 uur	28,1 dag
2^n	$\frac{1}{1000}$ sec	1 sec	35,7 jaar	400 biljoen eeuwen	75 cijfers eeuwen
n^n	2,8 uur	3,3 biljoen jaar	70 cijfers eeuwen	185 cijfers eeuwen	728 cijfers eeuwen

executietijd bij miljoen stappen per seconde

Merk op: de oerknal was ruim 10 miljard jaar geleden: dat is een getal van 11 cijfers.

De complexiteit van een algoritme hangt af van de soort invoer: worst case, average case, best case

- worst case complexiteit $g(n)$ = het aantal stappen in het *slechtste* geval: het algoritme doet voor elke mogelijke invoer dus hooguit $g(n)$ stappen
- best case complexiteit $h(n)$ = het aantal stappen in het *beste* geval: het algoritme doet voor elke mogelijke invoer dus minstens $h(n)$ stappen
- worst case geeft dus een bovengrens, best case een ondergrens voor het aantal uitgevoerde stappen
- average case complexiteit = de complexiteit gemiddeld over alle mogelijke invoeren

Voorbeeld: sorteeralgoritme

```
(1)   A[0] := 0; i := 1;
(2)   while i < n do
(3)       while i < n and A[i] < A[i + 1] do
(4)           i := i + 1;
(5)       od
(6)       if i < n then
(7)           wissel(A[i], A[i + 1]);
(8)           i := i − 1;
(9)       fi
(10)  od
```

Dit algoritme sorteert $A = A[1], \ldots, A[n]$ oplopend.
Aanname: $A[i] > 0$ voor $i = 1, \ldots, n$ en $n > 1$ en alle $A[i]$ verschillend.

Basisoperatie is de vergelijking $A[i] < A[i + 1]$ op regel 3. (Waarom?)

Vraag: Hoeveel arrayvergelijkingen doet dit algoritme in de best (resp. worst) case en voor wat voor invoerrijtjes komt dat voor?

- Te beginnen bij $i = 1$ vergelijken we telkens $A[i]$ met zijn rechterbuur. Zolang $A[i] < A[i+1]$ lopen we door. Zodra we een paar tegenkomen dat verkeerd staat (dat wil zeggen $A[i] > A[i+1]$) verwissel je die, doe je een stapje terug in regel (8) en kom je weer in regel (3) terecht. Als de huidige twee array-elementen weer verkeerd staan doe je nog een stap terug, etcetera. Dit levert je dus één of meer extra arrayvergelijkingen op.

- Het beste geval komt dus voor dan en slechts dan als je nooit in regel (8) terecht komt en i direct doorloopt tot $i = n$ en dan stopt. Dit is het geval d.e.s.d.a. voor elke $i = 1, \ldots, n-1$ geldt dat $A[i] < A[i+1]$, ofwel d.e.s.d.a. A al oplopend gesorteerd is. In dat geval doen we $n - 1$ arrayvergelijkingen.

- Het slechtste geval komt juist voor als we zo vaak mogelijk in regel (8) komen, dus als je steeds helemaal terug moet lopen en elementen verwisselen tot $i = 0$, waar je dan constateert dat $0 = A[0] < A[1]$ en je weer naar rechts zal lopen tot waar je begonnen was.
  Zie voor verdere uitleg het college.

- Merk op dat dit algoritme een niet al te slimme variant op Insertion sort is.

Best case:

$n - 1$ vergelijkingen
komt voor d.e.s.d.a. $A[i] < A[i+1]$ voor alle i,
m.a.w. als A stijgend is

Worst case:

$n^2 - 1$ vergelijkingen
komt voor d.e.s.d.a. voor alle i $A[i+1]$ kleiner
is dan *alle* voorgaande, m.a.w. als A dalend is

Average case:

$\Theta(n^2)$
gemiddeld over alle mogelijke invoeren (onder
de aanname dat ze alle even waarschijnlijk zijn)

→ Hier komen we bij het tweede college op terug ←

Bestaat er een efficiënter algoritme voor het probleem?

- dit heeft te maken met de complexiteit van het probleem: soms *kan* het niet beter

- de worst case complexiteit van een algoritme dat het probleem oplost levert een bovengrens voor de complexiteit van het probleem: het probleem kan opgelost worden in hooguit ... stappen (namelijk door dat algoritme)

- moeilijker is het bepalen van een (niet-triviale) ondergrens: hoeveel stappen zijn ten minste nodig om het probleem op te lossen?

Een simpele (meestal niet al te scherpe) ondergrens voor de (worst case) complexiteit van een probleem kan soms verkregen worden door het aantal invoer/uitvoer-elementen te tellen:

- ▷ het optellen van twee $n \times n$ matrices is $\Omega(n^2)$ (scherp)
- ▷ het handelsreizigersprobleem met n steden is $\Omega(n^2)$ (niet scherp)
- ▷ het zoeken naar X in een gesorteerd array met n elementen is $\Omega(n)$ (fout!)

→ Hier komen we bij het tweede college op terug ←

- een betere ondergrens op de complexiteit van een probleem vind je i.h.a. door te kijken naar de basisoperatie

- je bewijst zo (een ondergrens op) de complexiteit van een probleem binnen een bepaalde klasse van algoritmen, bijvoorbeeld algoritmen die gebaseerd zijn op het doen van arrayvergelijkingen

- bewijs stellingen die een ondergrens opleveren voor het aantal (basis)operaties dat nodig is om het probleem op te lossen, d.w.z.

$\rightarrow$ Hier komen we bij het tweede college op terug $\leftarrow$

- laat zien dat elk algoritme voor het probleem minstens ... elementaire (basis-) stappen moet doen in de ... case (meestal worst case)

- technieken voor het bewijzen van ondergrenzen zijn o.a. beslissingsboomargumenten en adversary-argumenten

- soms is een ad-hoc argument voldoende, zoals het bewijs uit het ongerijmde voor het vinden van het maximum van n getallen (zie college 2)

Bestudeer hoofdstuk 2 uit het dictaat!!!

- Floor en ceiling
- Logaritmen
- Sommaties
- n-faculteit
- Combinatoriek (tellen)
- O, Θ, Ω -notatie

- Het college gaat vooral over de *analyse* van algoritmen: hoeveel tijd/stappen heeft het algoritme nodig (complexiteit) en kan dat niet sneller (optimaliteit)?

- We abstraheren van concrete machines en implementaties en kijken naar het aantal basisoperaties

- Complexiteit van algoritmen

  ▷ Worst case, average case, best case

  ▷ O (grote-O), Θ en Ω

- Complexiteit van problemen / optimaliteit

- De laatste colleges: Complexiteitstheorie ($\mathcal{P}$, $\mathcal{NP}$, $\mathcal{NPC}$)

Volgende college: vrijdag 14 februari, 11.00–12.45 uur,
Gorlaeusgebouw zaal BW.0.40

Straks werkcollege: vrijdag 7 februari, 13.15–15.00,
Gorlaeusgebouw BW.0.29 en BW.0.30
Opgaven: 1, 2, 3abcd, 4, 6 (dictaat)

Thuis: bestudeer hoofdstuk 2 van het dictaat tot 2.4