Complexiteit 2023 — college 1

7 februari 2023

Introductie

Praktische informatie

Docenten
Jeannette de Graaf
Luc Edixhoven

Assistentie
Andy Tatman
Perri van den Berg
Rachel de Jong
Sara Kooistra

`l.j.edixhoven@liacs.leidenuniv.nl`
Maandag en dinsdag: Snellius 131

Praktische informatie

Website

`liacs.leidenuniv.nl/~edixhovenlj/complexiteit`

Materiaal

Dictaat met opgaven (zie website) + sheets

Brightspace

Hoofdzakelijk voor inleveren van opdrachten en voor cijfers

Vragen

Discord en discussieforum op Brightspace, anders e-mail.
Antwoorden binnen 'redelijke termijn'.

Praktische informatie

Rooster

Zie website ;) (en MyTimetable)

Belangrijkst

- hoorcolleges op dinsdagen 09u00–10u45, afwisselende zalen
- werkcolleges op dinsdagen 11u00–12u45, Snellius 302–304, 306–308 en 303
- géén college op 28 maart
- tentamen op maandag 5 juni 2023, 09u00–12u00
- hertentamen op maandag 10 juli 2023, 09u00–12u00

Praktische informatie

Huiswerk

- Vier opdrachten, 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.

Praktische informatie

Eindcijfer

if *tentamencijfer* $\geq 5{,}0$ **then**
  eindcijfer = tentamencijfer + gemiddeld huiswerkcijfer$/10$;
else
  eindcijfer = tentamencijfer;
fi

Vragen tot nu toe?

Waar gaat het vak over?

Analyse van algoritmen

- Algoritmiek (en Datastructuren) (al gehad, enige herhaling)
  - → hoe los je dit probleem op?
- Programmeren en Correctheid (keuzevak jaar 3)
  - → klopt je oplossing?
- Computability (ook dit semester)
  - → kunnen we dit probleem überhaupt oplossen?
- Complexiteit (dit vak)
  - → hoeveel werk kost deze oplossing? (tijd)complexiteit
  - → hoeveel geheugen? ruimtecomplexiteit
  - → en kan het beter? optimaliteit

En waarom is dat nuttig?

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}	$10^{3\,000}$	$10^{500\,000}$	$10^{70\,000\,000}$

En waarom is dat nuttig?

De absurd hoge getallen terzijde: bijvoorbeeld het sorteren van rijtjes is iets dat overal gebeurt en voor gigantische rijtjes, en niet-triviale algoritmen voor sorteren zijn o.a. in n^2, $n\sqrt{n}$ en $n \lg n$.

Voor $n = 10\,000\,000$, met $1\,000\,000$ operaties per seconde:

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

Bonus

n^3 → 32 miljoen jaar

Trix ⤴ zou nu net zijn begonnen aan haar derde rijtje.

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

Opbouw vak

Het vak bestaat uit grofweg twee delen:

1. Complexiteit en optimaliteit van algoritmen

   (d.w.z. tijdcomplexiteit)

2. Complexiteitstheorie: $\mathcal{P}$ vs $\mathcal{NP}$ en een beetje verder

Vandaag: inleiding basisconcepten en 'het vak in vogelvlucht'

Het laatste college is gereserveerd voor tentamenvoorbereiding. Er zit nog wat speling in het rooster aan het einde van het semester.

Hoeveelheid werk

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.
- grotere invoeren kosten meer werk, dus complexiteit wordt gewoonlijk uitgedrukt als functie van de grootte van de invoer

Maximum

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;
```

Maximum: complexiteit

We tellen het aantal operaties:

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

$3n \leq$ totaal aantal operaties $\leq 4n \Longrightarrow \Theta(n)$

Basisoperatie

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

Complexiteit

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
  $\Rightarrow O, \Omega, \Theta$

Grootte van de invoer

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

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
graafproblemen	aantal knopen en/of takken

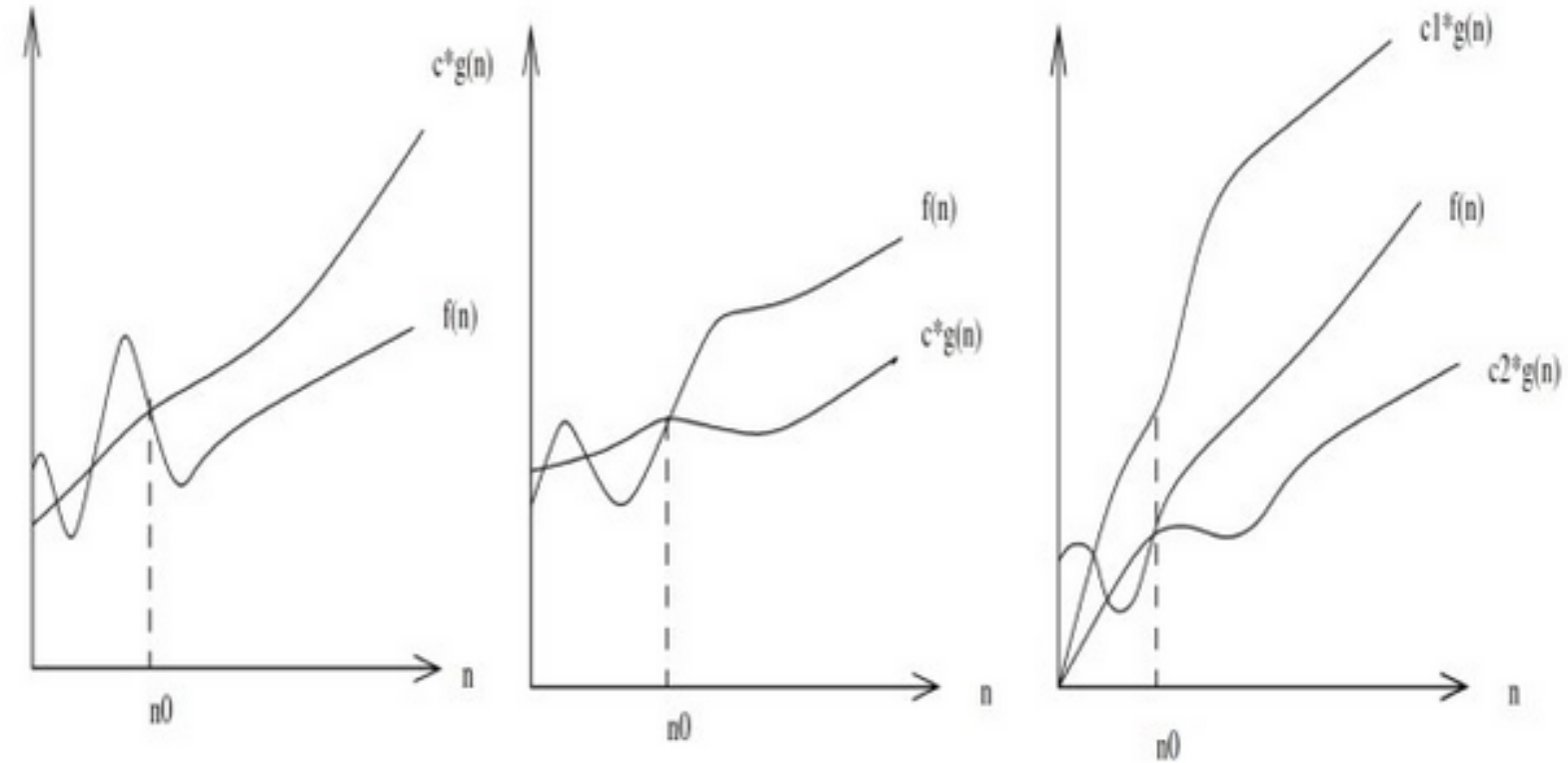
O, Ω en Θ

Gegeven twee functies $f(n)$ en $g(n)$. $O(g)$, $\Omega(g)$ en $\Theta(g)$ zijn verzamelingen van functies:

1. $f \in O(g)$: er bestaan constanten c en n_0 (beide > 0) zodat $0 \leq f(n) \leq c \cdot g(n)$ voor alle $n \geq n_0$: asymptotische bovengrens
2. $f \in \Omega(g)$: er bestaan constanten c en n_0 (beide > 0) zodat $0 \leq c \cdot g(n) \leq f(n)$ voor alle $n \geq n_0$: asymptotische ondergrens
3. $f \in \Theta(g)$: er bestaan constanten c_1, c_2 en n_0 (alle > 0) zodat $0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$ voor alle $n \geq n_0$: asymptotisch gedrag

Officieel dus $f \in O(g)$ en niet $f = O(g)$.

O, Ω en Θ



$f \in O(g)$

f groeit hooguit even hard als g

$f \in \Omega(g)$

f groeit minstens even hard als g

$f \in \Theta(g)$

f groeit even hard als g

O, Ω en Θ

Enkele voorbeelden:

- $\frac{1}{2}n^2 - 3n \in \Theta(n^2)$
- $3n^3 + 6n^2 + 9 \in \Theta(n^3)$
- $42n \in O(n^2)$, 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 $C > 0$); $0 \in O(1)$, maar $0 \notin \Omega(1)$
- $\sum_{i=1}^{n} i \in \Theta(n^2)$
- $\sum_{i=1}^{n} \frac{1}{i} \in \Theta(\lg n)$

Ordes van grootte

Veelgebruikte namen

$\Theta(1)$	**constant**
$\Theta(\lg n)$	**logaritmisch**
$\Theta(n)$	**lineair**
$\Theta(n^2)$	kwadratisch
$\Theta(n^k)$ met $k > 0$	**polynomiaal**
$\Theta(a^n)$ met $a > 1$	**exponentieel**
$\Theta(n!)$, $\Theta(n^n)$, ...	superexponentieel

Ordes van grootte

Gedrag voor grote n

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}	$10^{3\,000}$	$10^{500\,000}$	$10^{70\,000\,000}$

Best, worst, average case

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

- worst case complexiteit: $g(n)$ is het aantal stappen in het *slechtste* geval; het algoritme doet voor elke mogelijke invoer dus hooguit $g(n)$ stappen
- best case complexiteit: $h(n)$ is 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
- average case complexiteit: complexiteit 'gemiddeld' over alle mogelijke invoeren

Best, worst, average case

```
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 $A[1], \ldots, A[n]$ oplopend.
(met $A[i] > 0$ voor $i = 1, \ldots, n$ en alle $A[i]$ verschillend)

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

Best, worst, average case

```
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
```

Best case: $n - 1$ vergelijkingen ($\Theta(n)$) d.e.s.d.a. regel 3 telkens waar is, m.a.w. als A stijgend is

Best, worst, average case

```
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
```

Worst case: $n^2 - 1$ vergelijkingen ($\Theta(n^2)$) d.e.s.d.a. voor alle i $A[i+1]$ kleiner is dan alle voorgaande, m.a.w. als A dalend is

Best, worst, average case

```
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
```

Average case: $\Theta(n^2)$, gemiddeld over alle mogelijke invoeren (aangenomen dat ze allemaal even waarschijnlijk zijn)

Optimaliteit

Bestaat er een efficiënter algoritme voor het probleem?

- heeft te maken met de (inherente) complexiteit van het probleem: soms *kan* het niet beter
- de worst case van een algoritme geeft een bovengrens: het probleem kan worden opgelost in hooguit . . . stappen (namelijk door dat algoritme)
- moeilijker is het bepalen van een (niet-triviale) ondergrens

Optimaliteit

Een simpele (meestal niet al te scherpe) ondergrens voor de (worst case) complexiteit van een probleem kan soms worden verkregen 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!)

Optimaliteit

- 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 die gebaseerd 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. (volgende slide)

Optimaliteit

- 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 ook een ad-hoc argument, zoals voor het vinden van het maximum van n getallen

Recursie

Complexiteit van recursieve algoritmen:

- in het algemeen is hier het aantal recursieve aanroepen een goede maat voor de hoeveelheid werk
- maar je kunt ook hier het aantal keren tellen dat de basisoperatie wordt uitgevoerd
- een en ander leidt tot recurrente betrekkingen

Als voorbeeld bekijken we een recursief algoritme voor het bepalen van het maximum van n getallen in een array A.

Maximum: recursief

```
1  int grootste(int A[ ], n) {
2      if n = 1 then
3          return A[n];
4      else
5          max := grootste(A, n − 1);
6          if A[n] > max then
7              max := A[n];
8          fi
9      fi
10     return max;
11 }
```

$$C(n) = \begin{cases} 0 & n = 1 \\ C(n-1) + 1 & n > 1 \end{cases} \qquad \text{Oplossing} \Rightarrow C(n) = n - 1$$

Verdere wiskundige achtergrond

Raadpleeg het dictaat! o.a.:

Floor en ceiling

$\lfloor x \rfloor$ = het grootste gehele getal $\leq x$

$\lceil x \rceil$ = het kleinste gehele getal $\geq x$

Logaritmen

$\log_b(xy) = \log_b x + \log_b y \qquad \log_b x = \frac{\log_c x}{\log_c b}$

Sommaties

$\sum_{i=1}^{n} i = \frac{1}{2}n(n+1) \qquad \sum_{i=0}^{k} 2^i = 2^{k+1} - 1$

Verdere wiskundige achtergrond

Combinatoriek

Aantal rijtjes ter lengte k van getallen 1 t/m n (met $k \leq n$):

- volgorde van belang, hoeft niet allemaal verschillend: n^k
- volgorde van belang, wel allemaal verschillend: $n \cdot (n-1) \cdot (n-2) \cdots (n-k+1)$
- volgorde van belang, wel allemaal verschillend, $k = n$ — d.w.z. **permutaties**: $n! = n \cdot (n-1) \cdot (n-2) \cdots 2 \cdot 1$
- volgorde niet van belang, wel allemaal verschillend — d.w.z. **combinaties**: $\binom{n}{k} = \frac{n!}{k! \cdot (n-k)!}$

Complexiteitstheorie: $\mathcal{P}$ en $\mathcal{NPC}$

Algemener plaatje: welke problemen zijn 'moeilijk'?

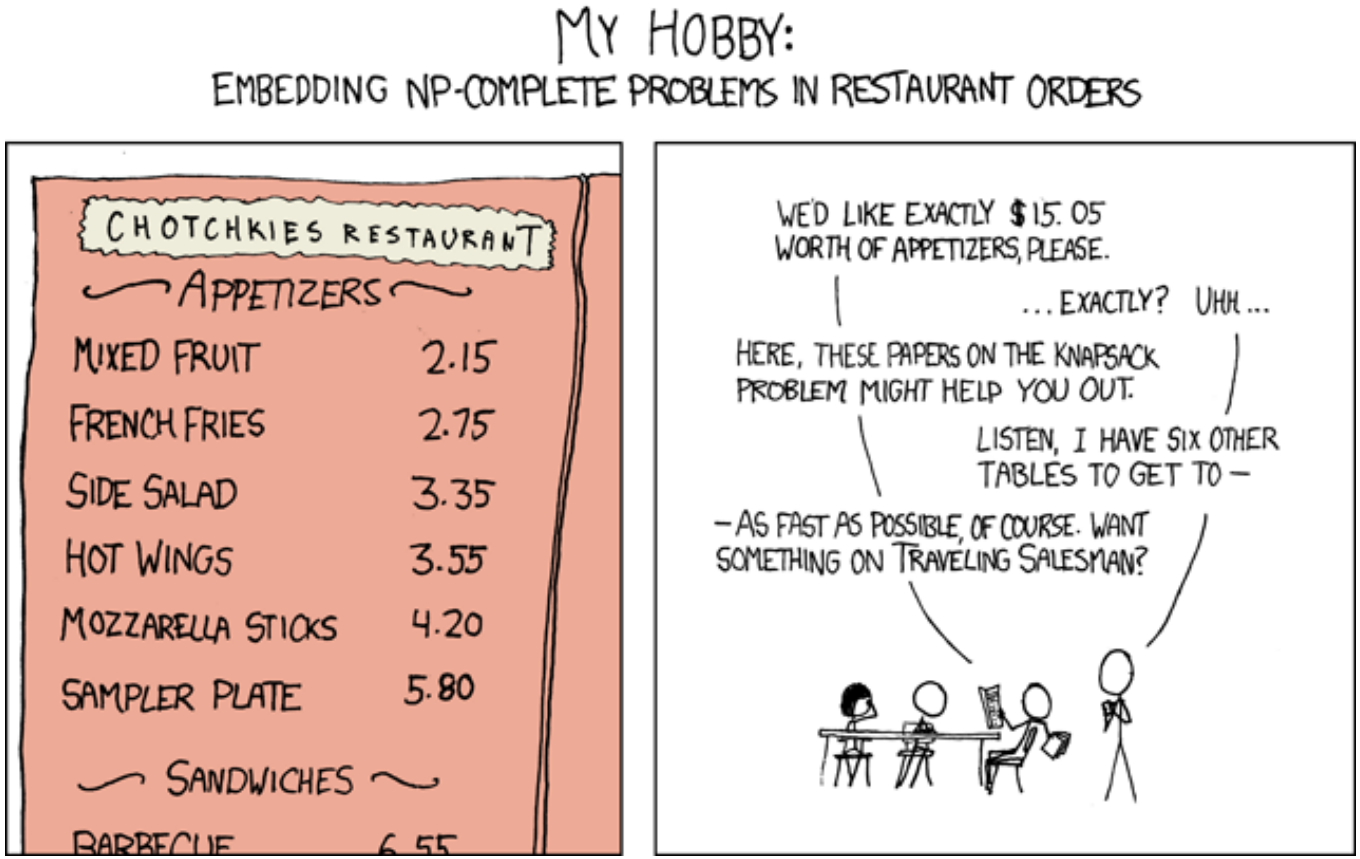
We onderscheiden drie soorten problemen:

- handelbare problemen → in het algemeen (min of meer) efficiënt op te lossen (d.w.z. polynomiale bovengrens): $\mathcal{P}$
- onhandelbare problemen → in het algemeen niet efficiënt op te lossen (bijv. exponentiële ondergrens)
- onbeslisbare problemen → in het algemeen überhaupt niet op te lossen: zie Computability

Complicatie: van veel problemen weten we gewoon niet of ze wel of niet handelbaar zijn: $\mathcal{NPC}$. Hiervoor weten we geen polynomiaal algoritme maar ook geen niet-polynomiale ondergrens.

$\mathcal{NPC}$

Typische problemen in $\mathcal{NPC}$:



`https://xkcd.com/287/`

Ook bijv. SAT en graafkleuring.

Reducties en $\mathcal{NPC}$

Met reducties kan je problemen met elkaar in verband brengen: als $A \leq_P B$ dan is A hooguit polynomiaal moeilijker dan B.

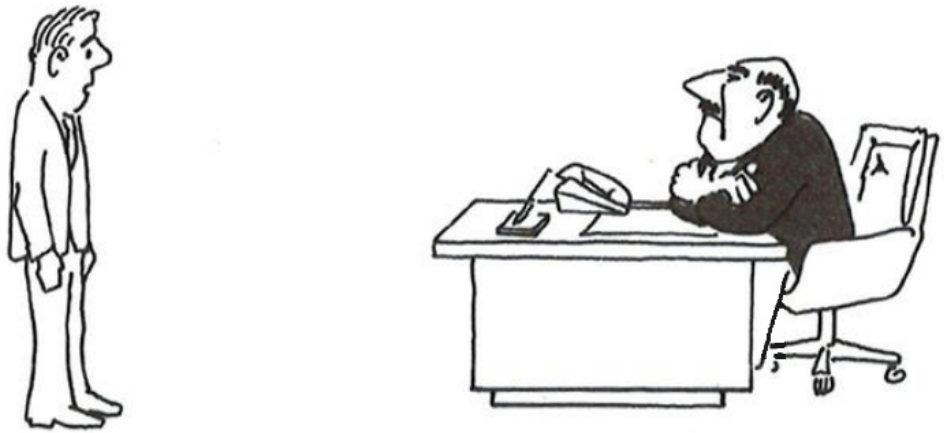
M.a.w.: als B polynomiaal kan, dan A ook! Nu hoef je alleen nog maar te redeneren over B.

Alle problemen in $\mathcal{NPC}$ reduceren naar elkaar. Het oplossen van één is voldoende om voor allemaal uitsluitsel te geven, maar dat is tot nu toe nog niemand gelukt.

$$\mathcal{P} \stackrel{?}{=} \mathcal{NP} \qquad \text{(of: } \mathcal{NPC} \stackrel{?}{\subset} \mathcal{P}\text{)}$$

(`https://www.claymath.org/millennium-problems`)

NPC



"I can't find an efficient algorithm, I guess I'm just too dumb."

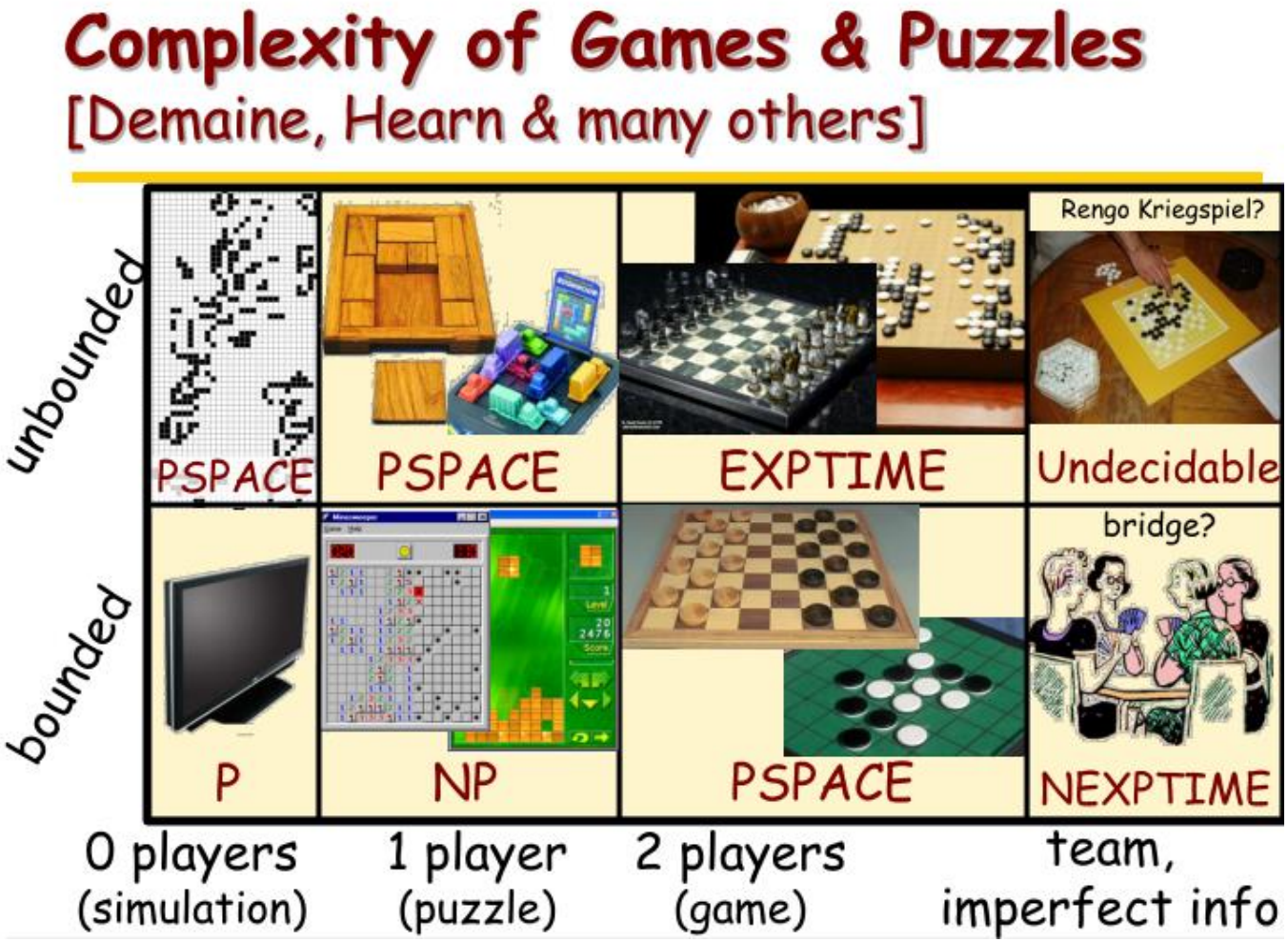
"I can't find an efficient algorithm, because no such algorithm is possible!"

NPC

"I can't find an efficient algorithm, but neither can all these famous people."

En verder?

Afhankelijk van hoe het semester loopt nog wat verdieping aan het einde.



Om te onthouden van vandaag

- basisoperaties
- ↳ complexiteit van algoritmen
  - O, Ω en Θ
  - best, worst en average case
- complexiteit van problemen

Eerste helft semester

- Volgende week: complexiteit en optimaliteit van zoekalgoritmen
- Tot eind maart: technieken voor ondergrenzen, sorteren en een handjevol andere problemen

(Werk)college

Volgende college: dinsdag 14 februari, 9u00–10u45, zaal C1 (Gorlaeus)

Werkcollege: zodadelijk van 11u00 tot 12u45, computerzalen 302–304, 306–308 en 303 (Snellius)
Opgaven uit het dictaat: 1, 2, 3abcde, 4, 6

Huiswerk

Huiswerk 1

Verschijnt vanmiddag (ergens na het werkcollege) op de website.

Inleveren via Brightspace, voor het begin van hoorcollege 4, d.w.z. vóór 28 februari 09u00 (drie weken).