Vierde college algoritmiek

23 februari 2026

Complexiteit

- woensdag 25 februari: geen practicumbijeenkomst
- dinsdag 24 februari: wel werkcollege
- volgende week programmeeropdracht 1?

Tijdcomplexiteit ('complexiteit') van een algoritme:

- = hoeveelheid werk verricht door het algoritme
- hangt meestal af van de grootte van de invoer: hoe groter de invoer, hoe groter de complexiteit
- wordt bepaald door het aantal keer dat de basisoperatie wordt uitgevoerd
- het belangrijkst is de (asymptotische) groei
- wordt vaak uitgedrukt in O-notatie (orde van grootte)
- hangt vaak ook af van het soort invoer: worst case, best case, average case

Groei

algoritme	$C(n)$	$n = 10$	$n = 100$	$n = 1000$
algo1	$10n$	100	1000	10.000
algo2	$100n$	1000	10.000	100.000
algo3	$5n^2$	500	50.000	5.000.000

Voorbeeld 1:

```
// invoer: array a[0···n − 1] bestaande uit n reals
// uitvoer: de grootste waarde
```

```
max := a[0];
for i := 1 to n − 1 do
    if ( a[i] > max ) then ⟵ basisoperatie
        max := a[i];
    fi
od
return max;
```

Complexiteit: $C(n) = n - 1 \in \Theta(n)$

Voorbeeld 2:

```
// invoer: array a[0···n − 1] bestaande uit n reals
// uitvoer: true als alle n waarden verschillen
```

for $i := 0$ **to** $n - 2$ **do**
  for $j := i + 1$ **to** $n - 1$ **do**
    if ($a[i] = a[j]$) **then** ⟵ basisoperatie
      return false; **fi**
  od
od
return true;

Best case complexiteit: ...

Worst case complexiteit: ...

Voorbeeld 2:

```
// invoer: array a[0···n−1] bestaande uit n reals
// uitvoer: true als alle n waarden verschillen
```

for $i := 0$ **to** $n-2$ **do**
  for $j := i+1$ **to** $n-1$ **do**
    if ($a[i] = a[j]$) **then** ⟵ basisoperatie
      return false; **fi**
  od
od
return true;

Best case complexiteit: $B(n) = 1 \in \Theta(1)$

Worst case complexiteit:

$$W(n) = \sum_{i=0}^{n-2} (n-1-i) = \frac{1}{2} n(n-1) \in \Theta(n^2)$$

1. Kijk welke parameter(s) de grootte van de invoer bepalen

2. Identificeer een basisoperatie

3. Hangt aantal keer dat basisoperatie wordt uitgevoerd alleen af van grootte invoer? Zo nee, voer dan worst-case, best-case, average-case analyse uit

4. Tel aantal keer dat basisoperatie wordt uitgevoerd (als som, recurrente betrekking)

5. Bepaal zo mogelijk gesloten formule, of orde van grootte

6. Laat constante factoren en kleinere termen weg

O-notatie beschrijft het asymptotisch gedrag:

$t \in O(g) \iff \exists c > 0$ en $\exists n_0 \geq 0$ zodat $\forall n > n_0$:

$$t(n) \leq c \cdot g(n)$$

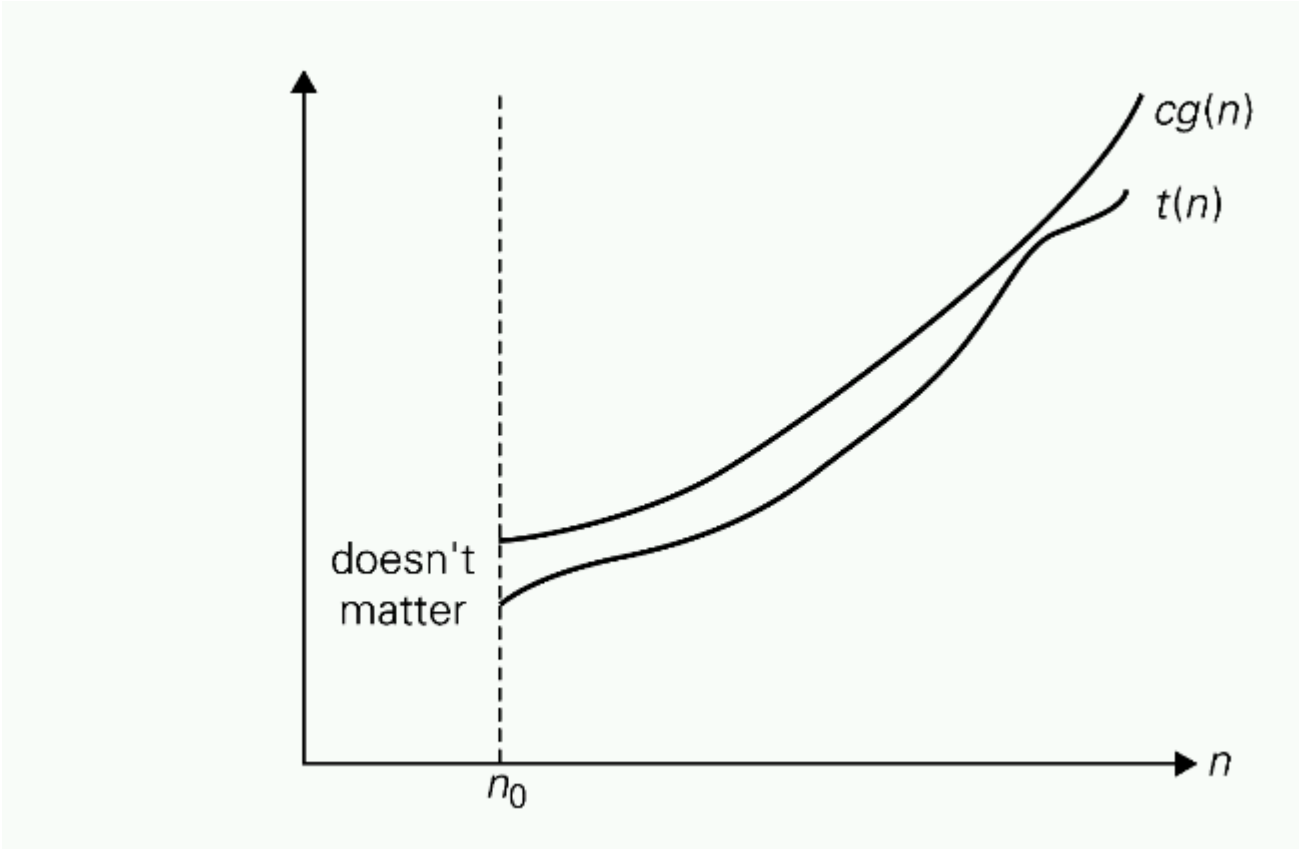
$t \in \Omega(g) \iff \exists c' > 0$ en $\exists n_0 \geq 0$ zodat $\forall n > n_0$:

$$t(n) \geq c' \cdot g(n)$$

$t \in \Theta(g) \iff \exists c_1, c_2 > 0$ en $\exists n_0 \geq 0$ zodat $\forall n > n_0$:

$$c_1 \cdot g(n) \geq t(n) \geq c_2 \cdot g(n)$$

Aangezien zowel t als g in onze toepassingen de complexiteit van een algoritme voorstelt is hier steeds $t > 0$ en $g > 0$.



$t \in O(g) \iff \exists c > 0$ en $\exists n_0 \geq 0$ zodat $\forall n > n_0$:

$$t(n) \leq c \cdot g(n)$$

$t(n)$ groeit niet sneller dan $g(n)$

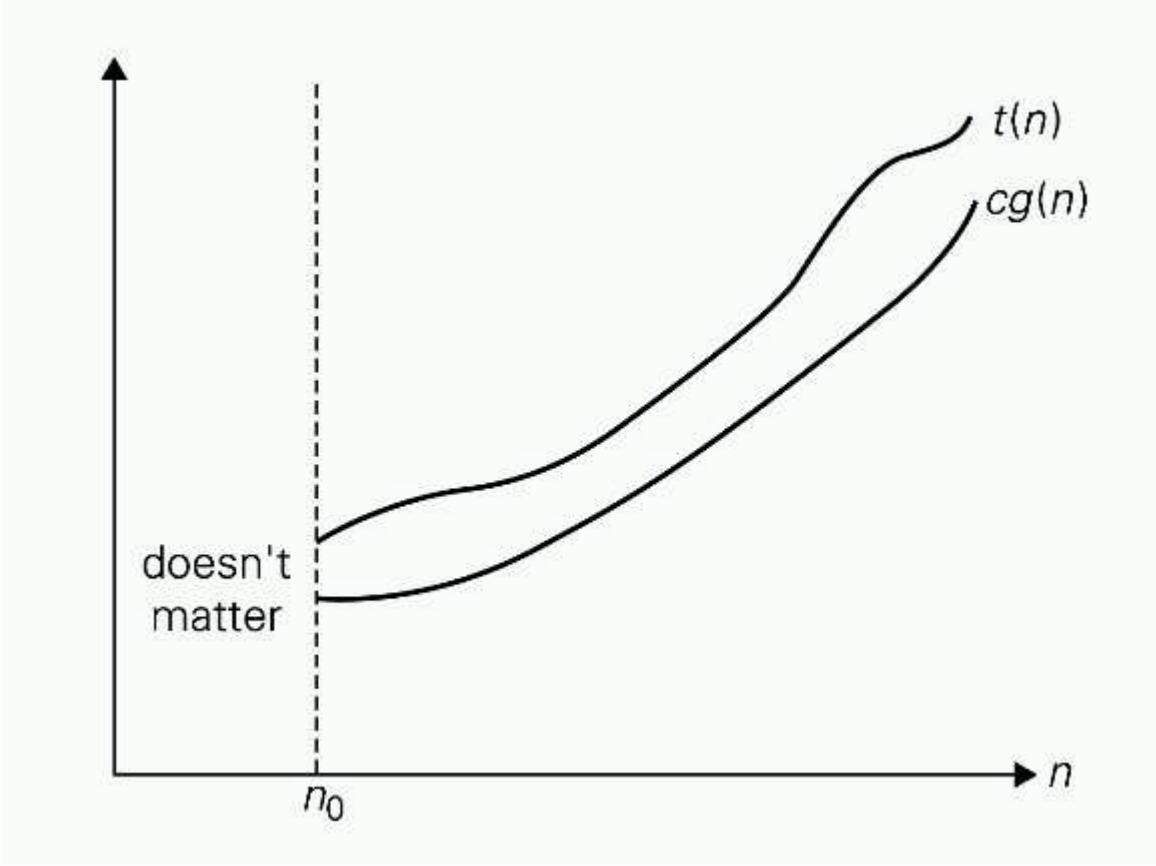
$t \in O(g) \iff \exists c > 0$ en $\exists n_0 \geq 0$ zodat $\forall n > n_0$:

$$t(n) \leq c \cdot g(n)$$

$t(n)$ groeit niet sneller dan $g(n)$

Bijvoorbeeld: $n^2 + 100n \in O(n^2)$
Neem maar $c = 1{,}1$:

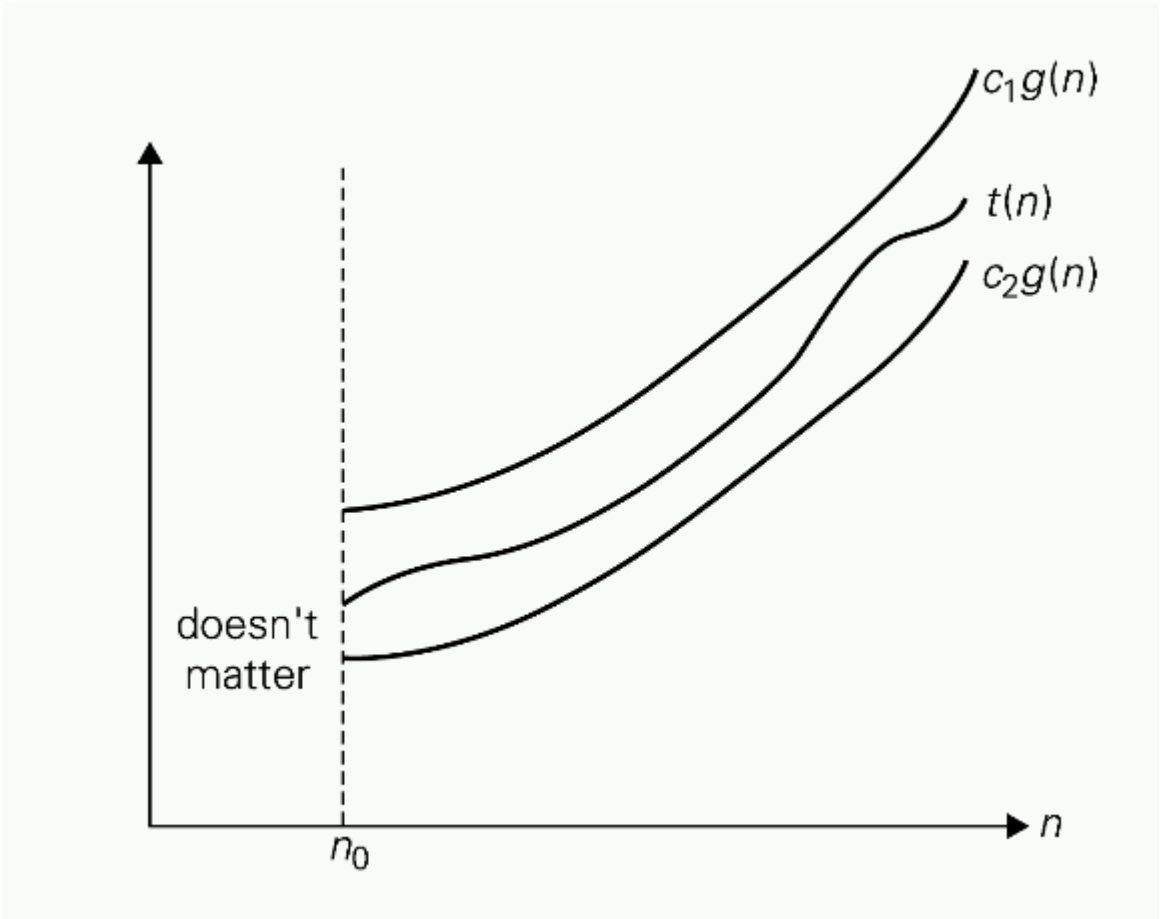
n	$n^2 + 100n$	$c \cdot n^2$
1	101	1,1
10	1100	110
100	20.000	11.000
1000	1.100.000	1.100.000



$t \in \Omega(g) \iff \exists c > 0$ en $\exists n_0 \geq 0$ zodat $\forall n > n_0$:

$$t(n) \geq c \cdot g(n)$$

$t(n)$ groeit minstens zo snel als $g(n)$



$t \in \Theta(g) \iff \exists c_1, c_2 > 0$ en $\exists n_0 \geq 0$ zodat $\forall n > n_0$:

$$c_1 \cdot g(n) \geq t(n) \geq c_2 \cdot g(n)$$

$t(n)$ groeit even snel als $g(n)$ (dus O + Ω)

Stelling:

(1) $$0 < \lim_{n\to\infty} \frac{t(n)}{g(n)} < \infty \iff t \in \Theta(g)$$

(2) $$\lim_{n\to\infty} \frac{t(n)}{g(n)} = 0 \iff t \in O(g), \text{ maar } t \notin \Theta(g)$$

(3) $$\lim_{n\to\infty} \frac{t(n)}{g(n)} = \infty \iff g \in O(t) \text{ maar } g \notin \Theta(t)$$

N	10	50	100	300	1000
$\log_2 N$	3	5	6	8	9
$5N$	50	250	500	1500	5000
$N \cdot \log_2 N$	33	282	665	2469	9966
N^2	100	2500	10.000	90.000	7 cijfers
N^3	1000	125.000	7 cijfers	8 cijfers	10 cijfers
2^N	1024	16 cijfers	31 cijfers	91 cijfers	302 cijfers
$N!$	7 cijfers	65 cijfers	161 cijfers	623 cijfers	onvoorstelbaar
N^N	11 cijfers	85 cijfers	201 cijfers	744 cijfers	onvoorstelbaar

Ter vergelijking:

het aantal protonen in het heelal is een getal met 79 cijfers
het aantal microseconden sinds de Big Bang heeft 24 cijfers

(1) $n(n-2) \in O(n^3)$; $n(n-2) \in O(n^2)$; $n(n-2) \in \Omega(n^2)$

(2) $n \in O(n^2)$, maar NIET $n \in \Theta(n^2)$

(3) $2^n \in O(3^n)$, maar NIET $2^n \in \Theta(3^n)$

(4) $(n^2+1)^{10} \in \Theta(n^{20})$

(5) $\sqrt{10n^2+7n+3} \in \Theta(n)$

(6) $2n \log_2(n+2)^2 + (n+2)^2 \log_2(n/2) \in \Theta(n^2 \log_2 n)$

(7) $2^{n+1} + 3^{n-1} \in \Theta(3^n)$

(8) $\lfloor \log_2 n \rfloor \in \Theta(\log_2 n)$; $\log_{10} n \in \Theta(\log_2 n)$

(9) $2^n \in O(n!)$, maar NIET $2^n \in \Theta(n!)$

n	$\sqrt{n}$	$^{10}\log n$	$^{2}\log n$
10	3,16	1	3,32
100	10,00	2	6,64
1.000	31,62	3	9,97
1.000.000	1.000,00	6	19,93
1.000.000.000	31.622,78	9	29,90

De volgende naamgeving wordt meestal gehanteerd:

1	constant
$\log n$	logaritmisch
n	lineair
$n \log n$	n-log-n
n^2	kwadratisch
n^α $(\alpha > 1)$	polynomiaal
2^n	exponentieel
$n!$, n^n, ...	superexponentieel

Voorbeeld 3:

```
// invoer: array a[0···n − 1] bestaande uit n reals en
//         een reeel getal k
// uitvoer: index i waarvoor a[i] = k; −1 als deze niet
//          bestaat
```

$i := 0$ ⟵ basisoperatie
while ($i < n$ **and** $a[i] \neq k$) **do**
    $i := i + 1$; **od**
if ($i < n$)
    return i; **fi**
return -1;

best case/worst case/average case: $\cdots$

Recursief algoritme voor de berekening van $n!$:

```
int faculteit ( int n ) { // gebruikt: n! = n · (n − 1)!
  if ( n == 0 )
     return 1;
  else          ↓ ——————— basisoperatie
     return n*faculteit(n-1);
}  // faculteit
```

$M(n)$ = aantal vermenigvuldigingen (= aantal recursieve aanroepen - 1) voldoet aan de recurrente betrekking:

$$\begin{cases} M(0) = 0 \\ M(n) = M(n-1) + 1 \quad \text{voor } n > 0 \end{cases}$$

Oplossing: $M(n) = n \longrightarrow$ complexiteit `faculteit` is $\Theta(n)$.

1. Kijk welke parameter(s) de grootte van de invoer bepalen

2. Identificeer een basisoperatie

3. Hangt aantal keer dat basisoperatie wordt uitgevoerd alleen af van grootte invoer? Zo nee, voer dan worst-case, best-case, average-case analyse uit

4. Tel aantal keer dat basisoperatie wordt uitgevoerd (als som, recurrente betrekking)

5. Bepaal zo mogelijk gesloten formule, of orde van grootte

6. Laat constante factoren en kleinere termen weg

Recursief algoritme voor de berekening van $n!$:

```
int faculteit ( int n ) { // gebruikt: n! = n · (n − 1)!
  if ( n == 0 )
     return 1;
  else          ↓ ←———————— basisoperatie
     return n*faculteit(n-1);
}  // faculteit
```

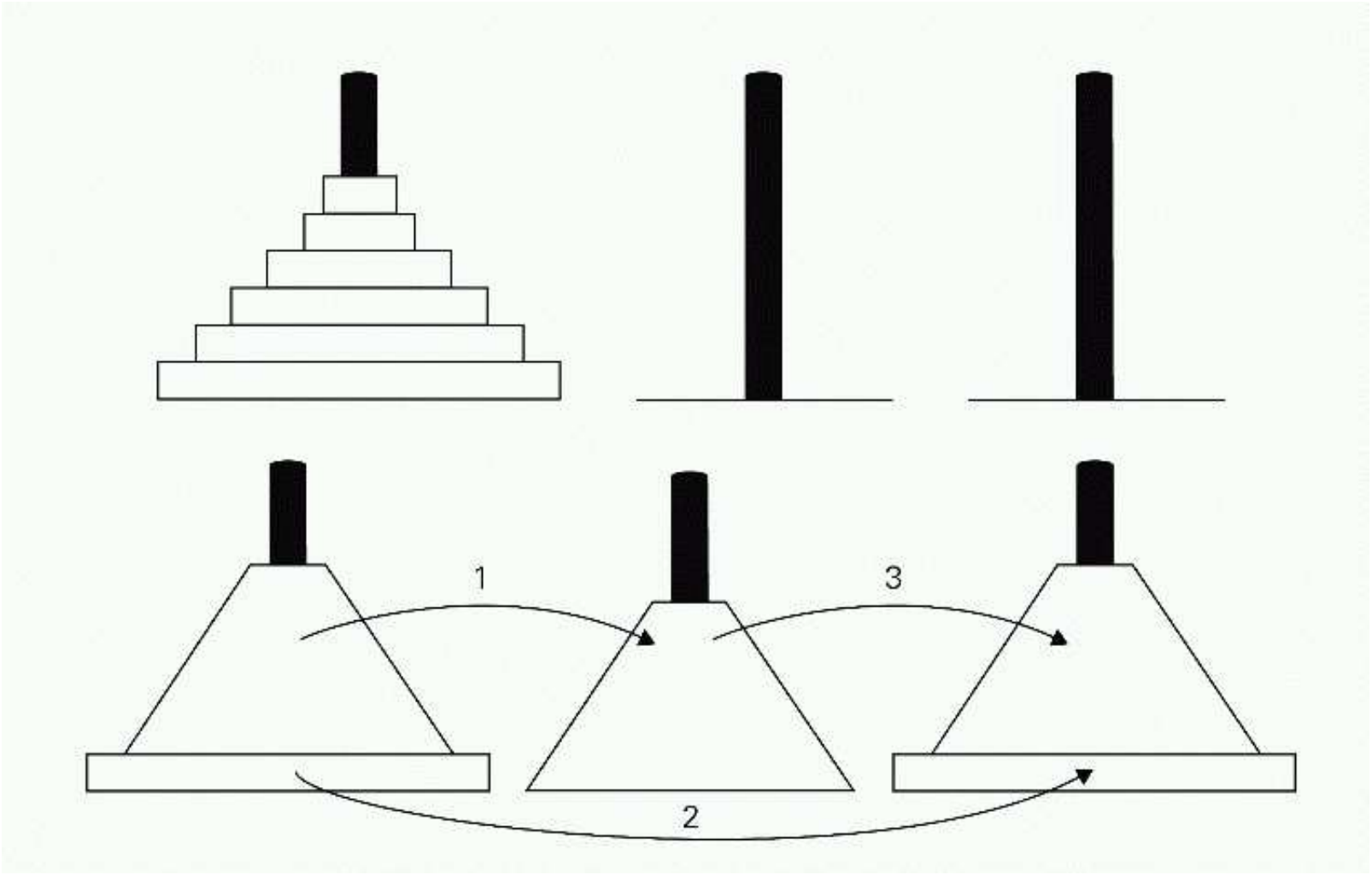
$M(n)$ = aantal vermenigvuldigingen (= aantal recursieve aanroepen - 1) voldoet aan de recurrente betrekking:

$$\begin{cases} M(0) = 0 \\ M(n) = M(n-1) + 1 \quad \text{voor } n > 0 \end{cases}$$

Oplossing: $M(n) = n \longrightarrow$ complexiteit `faculteit` is $\Theta(n)$.

Dit is feitelijk exponentieel in grootte van invoer.

Oplossing Hanoi algemeen



Recursieve oplossing van de Torens van Hanoi

Een recursief algoritme voor het probleem van de Torens van Hanoi (zie Programmeermethoden):

```
// zet toren van n stuks (optimaal) van a naar b via c
// print de zetten
void zet (int n, int a, int b, int c) {
  if ( n > 0 ) {
    zet (n-1, a, c, b);
    cout << "zet van " << a << "naar " << b << endl;
    zet (n-1, c, b, a);
  }  // if
}  // zet
```

- n (het aantal schijven) is een maat voor de grootte van de invoer
- het verzetten van een schijf is de basisoperatie

Laat $M(n)$ = aantal zetten, dan voldoet $M(n)$ aan de recurrente betrekking:

$$\begin{cases} M(0) = \ldots \\ M(n) = \ldots \quad \text{voor } n > 0 \end{cases}$$

Oplossing (zie college):

$M(n) = \ldots.$

- n (het aantal schijven) is een maat voor de grootte van de invoer
- het verzetten van een schijf is de basisoperatie

Laat $M(n)$ = aantal zetten, dan voldoet $M(n)$ aan de recurrente betrekking:

$$\begin{cases} M(0) = 0 \\ M(n) = 2M(n-1) + 1 \quad \text{voor } n > 0 \end{cases}$$

Oplossing (zie college):

$M(n) = 2^n - 1 \longrightarrow$ complexiteit `zet` is $\Theta(2^n)$.

Dit is feitelijk dubbel exponentieel in grootte van invoer.

Boom van recursieve aanroepen $n = 3 \ldots$

We willen een busreis maken langs steden $0, 1, 2, \ldots, n$, in die volgorde. Aangezien meerdere busmaatschappijen op de verschillende (deel)trajecten rijden, zijn de prijzen voor een rit van plaats i naar plaats j (via alle tussenliggende steden) per bus verschillend. Het kan dus voordeliger zijn om in plaats van rechtstreeks met de goedkoopste bus van plaats 0 naar n te reizen, (één of meer keer) over te stappen en met een andere bus verder te gaan.

Laat `prijs`$[i][j]$, de prijs van het goedkoopste buskaartje rechtstreeks van i naar j, gegeven zijn voor alle $i \leq j$. Het probleem is nu om de prijs van de goedkoopste reis van 0 naar n te vinden.

Voorbeeld:

$$\text{prijs} = \begin{pmatrix} 0 & 5 & 10 & 15 \\ - & 0 & 7 & 13 \\ - & - & 0 & 4 \\ - & - & - & 0 \end{pmatrix}$$

De prijs van de goedkoopste busreis van 0 naar 3 is hier. . .

Voorbeeld:

$$\text{prijs} = \begin{pmatrix} 0 & 5 & 10 & 15 \\ - & 0 & 7 & 13 \\ - & - & 0 & 4 \\ - & - & - & 0 \end{pmatrix}$$

De prijs van de goedkoopste busreis van 0 naar 3 is hier 14 (met tussenstop in plaats 2).

Laat kosten(n) de prijs van de goedkoopste busreis van 0 naar n voorstellen, langs alle tussenliggende steden (in oplopende volgorde). Dan geldt:

kosten(n) $= \ldots$

Laat kosten(n) de prijs van de goedkoopste busreis van 0 naar n voorstellen, langs alle tussenliggende steden (in oplopende volgorde). Dan geldt:

$$\text{kosten}(n) = \begin{cases} 0 & \text{als } n = 0 \\ \min_{0 \le k < n}(\text{kosten}(k) + \text{prijs}[k][n]) & \text{als } n \ge 1 \end{cases}$$

Een recursieve functie

```
kosten(n)::
    if n=0 then
       return 0;
    else
       temp := prijs[0][n];   // k = 0
       for k := 1 to n-1 do
           hulp := kosten(k) + prijs[k][n];
           if hulp < temp then
               temp := hulp;
           fi
       od
       return temp;
    fi .
```

Boom van recursieve aanroepen voor $n = 5$...

Basisoperatie...

Aantal keer basisoperatie / recurrente betrekking ...

Laat $A(n)$ het aantal keer zijn dat basisoperatie wordt uitgevoerd.

$$A(n) = \begin{cases} 0 & \text{als } n = 0 \\ n - 1 + A(1) + A(2) + \cdots + A(n-1) & \text{als } n \geq 1 \end{cases}$$

$A(1) = 0, A(2) = 1, A(3) = 3, A(4) = 7, A(5) = 15, \ldots$
$A(n) = 2^{n-1} - 1$ als $n \geq 1$ (met inductie)

- **Lezen/leren bij dit college:**
  slides
  Paragraaf 2.1–5

- **Werkcollege:**
  Dinsdag 24 februari 2026, 11.00–12.45, (Gorlaeus DM.0.17)

- **Opgaven:**
  zie `https://liacs.leidenuniv.nl/~vlietrvan1/algoritmiek/`

- **Volgend college:**
  maandag 2 maart 2026, 11.00-12.45