



Stof

- slides college 1
- boek 1.1, 1.2, 1.4, 1.6

Aantekeningen

Enkele aantekeningen, alles staat ook in het boek en op de slides.

Over de grote- O -notatie. Algoritmen in $O(n^k)$ voor zekere $k \in \mathbb{N}$ heten *polynomiaal* en gelden als redelijk efficiënt. Algoritmen in $O(k^n)$ voor zekere $k \in \mathbb{N}$ heten *exponentieel* en gelden als zeer inefficiënt.

Over logaritmen. De inverse van optellen is aftrekken: $(m+n) - n = m$. De inverse van vermenigvuldigen is delen: $\frac{m \times n}{n} = m$. De operatie exponentiëren (m^n) is niet symmetrisch. Er zijn twee inverse operaties. De ene is worteltrekken: $\sqrt[n]{m^n} = m$. De andere is de logaritme: $\log_m(m^n) = n$. In het vervolg schrijven we \log voor \log_2 .

Over logaritmen en de grote- O -notatie. Voorbeelden:

- $n \notin O(\log n)$ want (intuïtief): n groeit sneller dan de logaritme.
- $\log n \in O(n)$.
- $\log n \in O(\sqrt{n})$.
- Er geldt zelfs: $\log n \in O(n^{\frac{1}{100}})$.

Er is de volgende hiërarchie in complexiteitsklassen (in oplopende complexiteit): logaritmische functies (in klassen van de vorm $O(\log n)$), polynomiale functies (in klassen van de vorm $O(n^k)$ voor zekere $k \in \mathbb{N}$), exponentiële functies (in klassen van de vorm $O(k^n)$ voor zekere $k \in \mathbb{N}$), factoriële functies (in klassen van de vorm $O(n!)$).

Er geldt $\log_2 n \in \Theta(\log_3 n)$. Meer in het algemeen geldt:

$$\log_k n \in O(\log_l n) \text{ voor alle } k, l \in \mathbb{N}.$$

Waarom geldt dit? Zij $k, l \in \mathbb{N}$. Om te beginnen: $k^{((\log_k l)(\log_l n))} = (k^{\log_k l})^{\log_l n} = l^{\log_l n} = n$. Dus $(\log_k l)(\log_l n) = \log_k n$. Dus $\frac{\log_k n}{\log_l n} = \log_k l$. We concluderen: $\log_k n \in O(\log_l n)$ voor alle $k, l \in \mathbb{N}$.

Over algoritmes en de grote- O -notatie. De intuïtie is dat \mathcal{O} een bovengrens geeft voor de hoeveelheid werk die een algoritme kost. Iets preciezer: Stel dat een algoritme A voor iedere input van grootte $\leq n$ ten hoogste $f(n)$ tijds-eenheden nodig heeft om de output te berekenen, voor een functie $f : \mathbb{N} \rightarrow \mathbb{R}_{>0}$. Dan heeft A tijdscomplexiteit $O(f(n))$. NB: het gaat hier om ‘worst case’ tijdscomplexiteit. Je kunt ook kijken naar ruimte-complexiteit.

Opgaven

1. Het kan zijn dat een algoritme niet altijd de hele input nodig heeft om een (correcte) output te geven. Geef een voorbeeld van zo’n algoritme.
2. Gegeven een lijst met 25000 namen, alfabetisch geordend. We gaan de lijst sequentieel doorzoeken, op zoek naar naam N . We beschouwen het vergelijken van een naam uit de lijst met N als een stap. Hoeveel stappen hebben we nodig in het worst-case, best-case, average-case geval? (Is het een ‘eerlijke’ average case, eigenlijk?)
3. Gegeven is het volgende algoritme:

Algorithm search(A, n, x):

Input: array A storing $n \geq 1$ integers

Output: returns **true** if A contains x and **false** otherwise

for $i := 0$ **to** $n - 1$ **do**

if $A[i] = x$ **then**

return true

done

return false

Tel het aantal primitieve operaties (zoals in de slides van college 1) voor het worst-case en voor het best-case geval.

4. (Dit is ongeveer opgave R-1.7 uit het boek.)

In de volgende tabel staat links vertikaal een functie van n , denk voor n aan de grootte van de input. Horizontaal bovenaan staat de tijd die we bereid zijn te wachten op de output. Vul in hoe groot de input maximaal mag zijn, ongeveer, om op tijd de output af te kunnen lezen.

	1 seconde	1 uur	1 maand	1 eeuw
$\log n$	$\sim 10^{3000000}$			
n				
$n \log n$				
n^2				
n^3				
2^n				
$n!$		12		

5. Laat zien: als $f_1(n) \in \mathcal{O}(g_1(n))$ en $f_2(n) \in \mathcal{O}(g_2(n))$ dan $f_1(n) + f_2(n) \in \mathcal{O}(g_1(n) + g_2(n))$.
6. Geef een voorbeeld van het volgende: $f_1(n) \in \mathcal{O}(g_1(n))$ en $f_2(n) \in \mathcal{O}(g_2(n))$ maar $f_1(n) - f_2(n) \notin \mathcal{O}(g_1(n) - g_2(n))$.
7. Laat zien:
- (a) als $f(n) = 3 \cdot n + 5$ dan $f(n) \in \mathcal{O}(n)$,
 - (b) als $f(n) = 3 \cdot n^2$ dan $f(n) \notin \mathcal{O}(n)$,
 - (c) als $f(n) = 3 \cdot n^2$ dan $f(n) \in \mathcal{O}(n^2)$,
 - (d) als $f(n) = 3 \cdot n^2 + 2 \cdot n + 1$ dan $f(n) \in \mathcal{O}(n^2)$.
8. Laat zien:
- (a) $5n^2 + 3n \log n + 2n + 5 \in \mathcal{O}(n^2)$,
 - (b) $20n^3 + 10n \log n + 5 \in \mathcal{O}(n^3)$,
 - (c) $3 \log n + 2 \in \mathcal{O}(\log n)$,
 - (d) $2^{n+2} \in \mathcal{O}(2^n)$,
 - (e) $2n + 100 \log n \in \mathcal{O}(n)$.
9. (Deze opgave is ongeveer opgaven R-1.10-R-1.14 uit het boek.)
Geef van de volgende loopjes de tijdscomplexiteit in termen van grote- \mathcal{O} .

Algorithm Loop1(n):

```

s := 0
for i := 1 to n do
    s := s + i

```

Algorithm Loop2(n):

```

p := 1
for i := 1 to 2n do
    p := p · i

```

Algorithm Loop3(n):
 $p := 1$
 for $i := 1$ **to** n^2 **do**
 $p := p \cdot i$

Algorithm Loop4(n):
 $s := 0$
 for $i := 1$ **to** $2n$ **do**
 for $j := 1$ **to** i **do**
 $s := s + i$

Algorithm Loop5(n):
 $s := 0$
 for $i := 1$ **to** n^2 **do**
 for $j := 1$ **to** i **do**
 $s := s + i$

10. Gegeven is het volgende algoritme:

Algorithm find(k, A, n):
 for $i = 0$ **to** $n - 1$ **do**
 if $A[i] = k$ **then**
 return true
 done
 return false

Wat is de worst-case tijdscomplexiteit in termen van \mathcal{O} ?

11. Gegeven is het volgende algoritme:

```

Algorithm maxSubarray( $A, n$ ):
  for  $left := 0$  to  $n - 1$  do
     $sum := 0$ 
    for  $right := left$  to  $n - 1$  do
       $sum := sum + A[right]$ 
      if  $sum > max$  then
         $max := sum$ 
    done
  done
  return  $max$ 

```

Wat is de worst-case tijdscomplexiteit in termen van \mathcal{O} ?

12. Is een algoritme in $\mathcal{O}(n)$ *altijd* sneller dan een algoritme in $\mathcal{O}(n^2)$?
13. Beschouw de volgende definitie van machtsverheffen (power):

$$p(x, n) = \begin{cases} 1 & \text{als } n = 0 \\ x \cdot p(x, n - 1) & \text{als } n > 0 \end{cases}$$

Geef volgens deze definitie een algoritme **Power** in pseudo-code, en laat zien dat **Power** $\in \mathcal{O}(n)$.

14. Beschouw de volgende definitie van machtsverheffen (power):

$$q(x, n) = \begin{cases} 1 & \text{als } n = 0 \\ x \cdot q(x, \frac{n-1}{2})^2 & \text{als } n > 0 \text{ oneven} \\ q(x, \frac{n}{2})^2 & \text{als } n > 0 \text{ even} \end{cases}$$

Geef volgens deze definitie een algoritme **Qower** in pseudo-code, en laat zien dat **Qower** $\in \mathcal{O}(\log n)$.

15. Los de volgende recurrente betrekking voor de complexiteit van een algoritme voor de torens van Hanoi op:

$$T(n) = \begin{cases} 1 & \text{als } n = 1 \\ 2T(n - 1) + 1 & \end{cases}$$