



Uitwerkingen

1. Bijvoorbeeld het algoritme: vind een getal met een bepaalde eigenschap ($= 2, > 2, \dots$) in een rijtje getallen. Of: kijk of een graaf een cykel bevat.
2. Worst-case: 25000 stappen want N was net de laatste.
Best-case: 1 stap want N was net de eerste.
Average-case: 12500 stappen. Dit is niet helemaal 'eerlijk' want namen zijn niet homogeen verdeeld in de alfabetische ordening.

3. We tellen het aantal elementaire stappen in het algoritme $\text{search}(A, n, x)$. Het worst-case geval is als x niet voorkomt in A . We doen dan de volgende elementaire stappen:

1 voor de assignment $i = 0$,
 $n + 1$ keer de test $i \leq n - 1$ van grootte 1 (met n keer true en 1 keer fail),
1 voor de return aan het eind,
 n keer de loop met daarin
1 voor opvragen $A[i]$,
1 voor de vergelijking $A[i] = x$,
1 voor het ophogen van de index i .

Totaal: $T(n) = 1 + (n + 1) + 1 + n \cdot (1 + 1 + 1) = 4n + 3$.

Voor het best-case geval: dat is als $A[0] = x$. We doen dan de volgende elementaire stappen:

1 voor de assignment $i = 0$,
1 voor de test $i \leq n - 1$,
1 voor opvragen $A[i]$,
1 voor de vergelijking $A[i] = x$,
1 voor de return.

Totaal: $T(n) = 5$.

4. De (nog niet) ingevulde tabel:

	1 seconde	1 uur	1 maand	1 eeuw
$\log n$	$\sim 10^{300000}$			
n				
$n \log n$				
n^2				
n^3				
2^n				
$n!$		12		

5. Als $f_1(n) \in \mathcal{O}(g_1(n))$ dan $\exists n_1 \in \mathbb{N} \exists c_1 > 0 : n \geq n_1 \Rightarrow f_1(n) \leq c_1 \cdot g_1(n)$.
 Analoog geldt: als $f_2(n) \in \mathcal{O}(g_2(n))$ dan $\exists n_2 \in \mathbb{N} \exists c_2 > 0 : n \geq n_2 \Rightarrow f_2(n) \leq c_2 \cdot g_2(n)$.
 Neem nu $n^* := \max\{n_1, n_2\}$ en $c^* := \max\{c_1, c_2\}$.
 Dan geldt: als $n \geq n^*$ dan $f_1(n) \leq c_1 \cdot g_1$ dus ook $f_1(n) \leq c^* \cdot g_1$, en $f_2(n) \leq c_2 \cdot g_2$ dus ook $f_2(n) \leq c^* \cdot g_2$. Dus $(f_1 + f_2)(n) \leq c^* \cdot ((g_1 + g_2)(n))$ voor $n \geq n^*$. Dus $(f_1 + f_2)(n) \in \mathcal{O}((g_1 + g_2)(n))$.
6. $n^2 + n \in \mathcal{O}(n^2)$ en $n^2 \in \mathcal{O}(n^2)$ maar $n^2 + n - n^2 = n \notin \mathcal{O}(n^2 - n^2) = \mathcal{O}(0)$.
7. (a) als $f(n) = 3 \cdot n + 5$ dan $f(n) \in \mathcal{O}(n)$ want:
 neem $c = 8$, dan geldt voor alle $n \geq 1$: $f(n) = 3 \cdot n + 5 \leq c \cdot n$.
- (b) als $f(n) = 3 \cdot n^2$ dan $f(n) \notin \mathcal{O}(n)$ want:
 $\frac{3n^2}{n} = 3n$ wordt niet begrensd door een constante.
- (c) als $f(n) = 3 \cdot n^2$ dan $f(n) \in \mathcal{O}(n^2)$ want:
 neem $c = 3$, dan geldt voor alle $n \geq 0$: $f(n) = 3 \cdot n^2 \leq c \cdot n^2$.
- (d) als $f(n) = 3 \cdot n^2 + 2 \cdot n + 1$ dan $f(n) \in \mathcal{O}(n^2)$ want:
 neem $c = 6$, dan geldt voor alle $n \geq 1$: $f(n) = 3 \cdot n^2 + 2 \cdot n + 1 \leq c \cdot n^2$.
 ($f(2) = 17 \leq 24 = c \cdot n^2$.)
- (Er zijn in het algemeen ook andere c 's en n_0 's die goed werken.)
8. (a) $5n^2 + 3n \log n + 2n + 5 \in \mathcal{O}(n^2)$:
 voor $n \geq 2$ geldt dat $5n^2 + 3n \log n + 2n + 5 \leq (5 + 3 + 2 + 5)n^2$, dus neem $c = 15$.
- (b) $20n^3 + 10n \log n + 5 \in \mathcal{O}(n^3)$:
 voor $n \geq 1$ geldt dat $20n^3 + 10n \log n + 5 \leq 35n^3$, dus neem $c = 35$.
- (c) $3 \log n + 2 \in \mathcal{O}(\log n)$:
 voor $n \geq 2$ geldt dat $3 \log n + 2 \leq 5 \log n$ (nb: $\log 1 = 0$).
- (d) $2^{n+2} \in \mathcal{O}(2^n)$:
 voor $n \geq 1$ geldt dat $2^{n+2} = 2^n \cdot 2^2 = 4 \cdot 2^n$ dus neem $c = 4$.
- (e) $2n + 100 \log n \in \mathcal{O}(n)$:
 voor $n \geq 2$ geldt dat $2n + 100 \log n \leq 102n$ dus neem $c = 102$.
9. Het algoritme Loop1 is in $\mathcal{O}(n)$.
 Het algoritme Loop2 is in $\mathcal{O}(n)$. (De eerste intuïtie is misschien $\mathcal{O}(2n)$ maar dan is het beter om $\mathcal{O}(n)$ te gebruiken.)
 Het algoritme Loop3 is in $\mathcal{O}(n^2)$.
 We bekijken eerst een aangepaste versie van Loop4 waarbij we de index i laten lopen van 1 tot en met n . De hoeveelheid werk is dan $1 + 2 + \dots + n = \frac{n(n+1)}{2} = \frac{1}{2}n^2 + \frac{1}{2}n$. (We gebruiken Theorem 1.13.) De aangepaste versie van Loop4 is in $\mathcal{O}(n^2)$.
 Voor de oorspronkelijke versie van Loop4, waarbij de index i van 1 tot en met $2n$ loopt, is de hoeveelheid werk $1 + 2 + \dots + 2n = \frac{2n(2n+1)}{2} = 2n^2 + n$. Dus ook de oorspronkelijke versie van Loop4 is in $\mathcal{O}(n^2)$.

De hoeveelheid werk is $1 + 2 + \dots + n^2 = \frac{n^2(n^2+1)}{2} = \frac{1}{2}n^4 + \frac{1}{2}n^2$. Dus Loop5 is in $\mathcal{O}(n^4)$.

10. De worst-case tijdscomplexiteit is in $\mathcal{O}(n)$. (De best-case tijdscomplexiteit is in $\mathcal{O}(1)$.)
11. De worst-case tijdscomplexiteit is in $\mathcal{O}(n^2)$.
12. Nee, het kan gebeuren dat een algoritme in $\mathcal{O}(n^2)$ beter is dan een algoritme in $\mathcal{O}(n)$.

Voorbeeld: een algoritme A met tijdscomplexiteit gegeven door de functie $f(n) = 100 \cdot n$ is in $\mathcal{O}(n)$, en een algoritme B met tijdscomplexiteit gegeven door de functie $g(n) = n^2$ is in $\mathcal{O}(n^2)$. Op den duur is, voor grotere inputs, is A beter. Maar voor de kleine input met $n = 2$ geldt $f(n) = 200 > 4 = g(n)$. Dus voor een *kleine* input kan een algoritme in $\mathcal{O}(n^2)$ toch beter zijn dan een algoritme in $\mathcal{O}(n)$.

Ander voorbeeld: bubbelsort is in $\mathcal{O}(n^2)$ maar voor het specifieke geval van een input die al gesorteerd is, is het aantal stappen in $\mathcal{O}(n)$. Meer in het algemeen: voor een input emphmet een specifieke eigenschap kan een algoritme in $\mathcal{O}(n^2)$ toch beter zijn dan een algoritme in $\mathcal{O}(n)$.

13. Pseudo-code:

Algorithm Power(x, n):

Input: positive integers x and n

Output: the value x^n

if $n = 0$ **then**

return 1

if $n > 0$ **then**

$y :=$ Power($x, n - 1$)

return $x \cdot y$

14. nog toevoegen

Pseudo-code:

Algorithm $\text{Qower}(x, n)$:

Input: positive integers x and n

Output: the value x^n

if $n = 0$ **then**

return 1

if n *nodd* **then**

$y := \text{Qower}(x, \frac{n-1}{2})$

return $x \cdot y \cdot y$

else

$y := \text{Qower}(x, \frac{n}{2})$

return $y \cdot y$

15. We lossen de recurrente betrekking

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

voor de torens van Hanoi op als volgt:

$$\begin{aligned} T(n) &= 2T(n-1) + 1 \\ &= 2(2T(n-2) + 1) + 1 \\ &= 4T(n-2) + 2 + 1 \\ &= 4(2T(n-3) + 1) + 2 + 1 \\ &= 8T(n-3) + 4 + 2 + 1 \\ &= \dots \\ &= 2^i T(n-i) + 2^{i-1} + \dots + 2^0 \end{aligned}$$

Substitueer: $n = i + 1$ oftewel $i = n - 1$. Dan vinden we:

$$\begin{aligned} T(n) &= 2^{n-1} + 2^{n-2} + \dots + 2^0 \\ &= 2^n - 1. \end{aligned}$$