



EMS30 Week 7 Les 2: Algoritmen, performance en big-O-notatie

Leerdoelen week 7 les 2. Je leert hoe:

- je de orde (big-O-notatie) van een algoritme kunt bepalen;
- verschillende implementaties van algoritmen die functioneel gelijk zijn, toch in orde kunnen verschillen;
- verschillende datastructuren geïmplementeerd kunnen worden en hoe dit de orde van de algoritmen die je op deze datastructuren kunt uitvoeren beïnvloed.

- De big O notatie wordt gebruikt om de executietijd en het geheugengebruik van algoritmen met elkaar te **vergelijken**.
- De O staat voor '**order of magnitude**'.
- $f(n)=O(n^2)$ betekent dat functie $f(n)$ '**van boven**' **asymptotisch** begrensd wordt door $g(n)=n^2$.
- Dit betekent dat $f(n) \leq a \cdot n^2$ voor **grote** waarden van n . Waarbij $a \geq 0$ (je mag a zelf kiezen).

Big-O-notatie voorbeelden

Wat is de big O notatie voor:

- $f(n) = 2n^3 + 4n + 2 \log_2 n + 100.$ $= O(n^3)$
- $f(n) = 4 \log_2 n.$ $= O(\log n)$
- $f(n) = \ln n.$ $= O(\log n)$
- $f(n) = 1000.$ $= O(1)$
- $f(x) = (10x^4 + 4x^2) / (5x^3 + 12x^2 + 7x).$ $= O(x)$

Big-O-notatie voorbeeld

Wat is de executietijd van het onderstaande algoritme **max** in **big-O-notatie**: $T(n) = O(n)$

```
int max(const vector<int>& v) {  
    assert(v.size() > 0);  
    auto max {v[0]};  
    for (auto elm: v) {  
        if (elm > max) {  
            max = elm;  
        }  
    }  
    return max;  
}
```

Elk element moet 1x
bekeken worden

Kan het **beter**?

... als je $\frac{1}{2}$ **v.size()** processoren hebt ?

$O(\log n)$

... als de vector al **gesorteerd** is?

$O(1)$

Big-O-notatie voorbeeld

Wat is de executietijd van het onderstaande algoritme **maxprod** in **big-O-notatie**: $T(n) = O(n^2)$

```
int maxprod(const vector<int>& v) {  
    auto n {v.size()};  
    assert(n > 0);  
    auto maxp {v[0] * v[0]};  
    for (decltype(n) i {0}; i < n; ++i) { n keer  
        for (auto j {i}; j < n; ++j) { gemiddeld 1/2n keer  
            if (v[i] * v[j] > maxp) { n · 1/2n = 1/2n2 keer  
                maxp = v[i] * v[j];  
            }  
        }  
    }  
    return maxp;  
}
```

Big-O-notatie voorbeeld

Wat is de executietijd van het onderstaande algoritme **maxprod** in **big-O-notatie**: $T(n) = O(n)$

```
int maxprod(const vector<int>& v) {  
    auto n {v.size()};  
    assert(n > 0);  
    auto max {v[0]};  
    for (auto elm: v) { n keer  
        if (elm > max) {  
            max = elm;  
        }  
    }  
    return max * max;  
}
```

Maximum van alle
mogelijke producten is
 \max^2

Big-O-notatie

Orde	$n = 100$	$n = 10000$	$n = 1000000$	$n = 100000000$
$O(1)$	1 ms	1 ms	1 ms	1 ms
$O(\log n)$	1 ms	2 ms	3 ms	4 ms
$O(n)$	1 ms	0,1 s	10 s	17 min
$O(n \cdot \log n)$	1 ms	0,2 s	30 s	67 min
$O(n^2)$	1 ms	10s	28 uur	761 jaar
$O(n^3)$	1 ms	17 min	32 jaar	31710 eeuw
$O(10^n)$	1 ms	∞	∞	∞

Je ziet dat een $O(n^2)$ algoritme niet bruikbaar is voor hele grote hoeveelheden data en dat een $O(10^n)$ algoritme sowieso niet bruikbaar is.

Allemaal al bedacht in de jaren '60.

- *The Art of Computer Programming* van Donald Knuth
https://en.wikipedia.org/wiki/The_Art_of_Computer_Programming
- Stack (`stack`)
- Queue (`queue`)
- Dynamic Array (`vector`)
- Sorted Dynamic Array (`vector`)
- Doubly Linked List (`list`)
- Search Tree (`set`, `map`, `multiset` en `multimap`)
- Hash Table (`unordered_set`, `unordered_map`, ...)
- Binary Heap (`priority_queue`)

- Ding om data ‘gestructureerd’ in op te slaan.
- Een **template** is erg geschikt voor het implementeren van een datastructuur.
- Basisbewerkingen:
 - **insert, remove, find**
 - empty, full, size
- Een bepaalde datastructuur kan op verschillende manieren geïmplementeerd worden.

Zie dictaat tabel 7.2.

naam	insert	erase	find
stack			
queue			
dynamic array			
sorted dyn. array			
doubly linked list			

Zie dictaat tabel 7.2.

naam	insert	erase	find
search tree			
hash table			
binary heap			

- [Binary search tree](#): [AVL-tree](#), [Red-black tree](#), ...
- [Hash table](#)
- [Binary heap](#)

Aan de slag!

Aan de slag met [Opdrachten_Week_7_Les_2.pdf](#)

