Algoritmes & Complexity

Bachelor IT
Sven Mariën
(sven.marien01@ap.be)

AP HOGESCHOOL ANTWERPEN

AP.BE

Algoritme ???

- **Een algoritme is een stappenplan om een bepaald probleem op te lossen.**

Geschiedenis [bewerken | brontekst bewerken]

Het woord algoritme is een verbastering van het Oudengelse woord *algorism*, dat van het Latijnse woord *algorismus* komt, dat weer voortkomt uit de naam van de Perzische wiskundige Al-Chwarizmi (ca. 780 - ca. 845). Hij was de auteur van het boek *al-Kitab al-mukhtasar fi hisab al-jabr w'al-muqabala* (*Boek van de beknopte rekenkundige algebra en handelsbalans*) dat de algebra in de Westerse wereld introduceerde. Het woord algebra zelf komt van *al-Jabr* uit de titel van het boek. Het woord *algorisme* verwees oorspronkelijk alleen naar de regels voor het rekenen met Arabische cijfers, maar was in de 18e eeuw naar *algoritme* geëvolueerd. Het woord algoritme wordt nu gebruikt voor alle eindige procedures om problemen op te lossen of taken uit te voeren.

- Een **algoritme**:
  - is een ondubbelzinnig beschreven stappenplan dat duidelijk maakt wat er moet gebeuren.
  - verwacht 1 of meerdere inputs (parameters), bv. 2 getallen groter dan 0, een lijst van getallen of woorden,….
  - produceert een set van outputs (resultaten), bv. het grootste van de 2 getallen, een gesorteerde lijst van de getallen, …
  - Het zal gegarandeerd na een bepaalde tijd eindigen en een resultaat teruggeven.

Voorbeeld van een eenvoudig algoritme

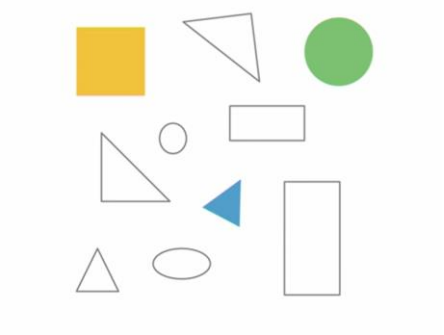
Zoek de hoogste waarde in een lijst van getallen.

- **Inputs**: een lijst van (minstens 1) positieve getallen.
- **Outputs**: een getal, dewelke het hoogste getal in de lijst voorstelt.
- **Algoritme:**
  1. Zet *max* = 0
  2. Voor elk getal *x* in de lijst, vergelijk met *max.* Indien *x* > *max,* stel *max* = *x*
  3. *max* is het hoogste getal in de lijst

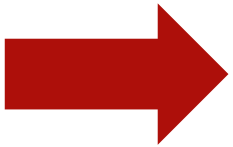
Algoritme vb. 2

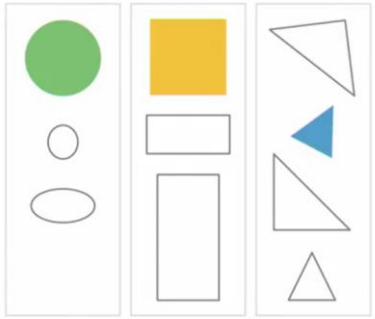
Groeperen van gelijkaardige figuren

- **Inputs**: een lijst van geometrische figuren.
- **Outputs**: een lijst van groepen met de toegekende figuren.



```
for (each shape) {
  if (shape is ellipse) {
    ellipses.add(shape)
  }
  if (shape is rectangle) {
    rectangles.add(shape)
  }
  if (shape is triangle) {
    triangles.add(shape)
  }
}
```

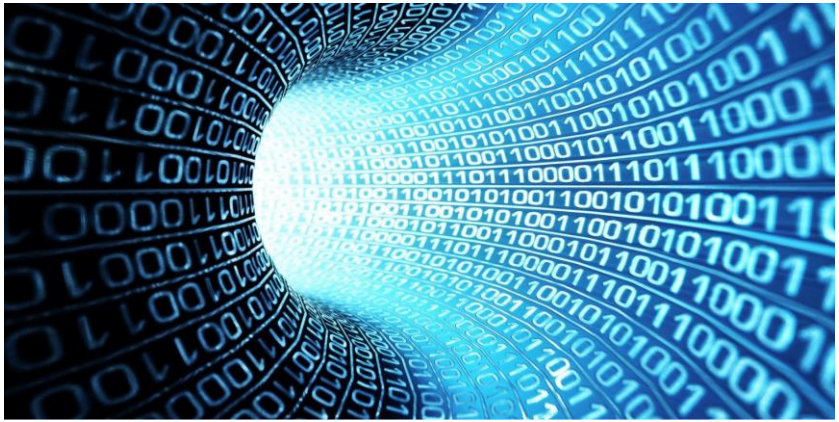


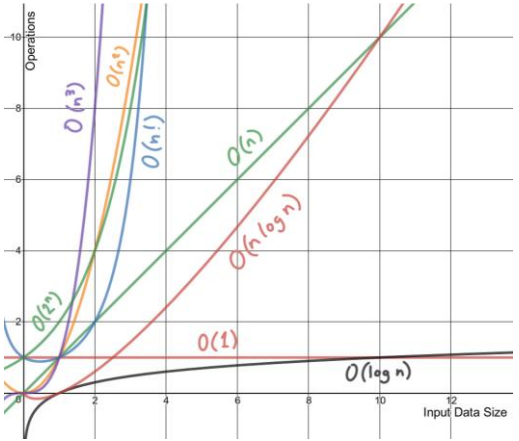
Welke ?

- **Zoekalgoritmes**: zoeken naar een bepaald element in een set van data (bv. woord zoeken in een tekst, bestand zoeken in een folderstructuur op disk,..)
- **Sorteeralgoritmes**: de volgorde van de elementen in een bepaalde volgorde plaatsen (van laag naar hoog,…)
- **Berekeningsalgoritmes**: uit een set van gegevens een andere set van gegevens afleiden (bv. het gemiddelde berekenen, omzetten van een temperatuur van de ene eenheid naar een andere eenheid,...)
- **Filteralgoritmes**: uit een set van gegevens een subset afleiden (bv. alle priemgetallen eruit halen,..)

Hoe een algoritme beoordelen:

- **Time-complexity:** hoeveel **tijd** neemt een algoritme in beslag in **functie van de input**
- **Space-complexity:** hoeveel **geheugen** neemt een algoritme in beslag in **functie van de input**





Time-complexity

- Ik wens de **Time-complexity** te onderzoeken van dit '**ZoekMax**' algoritme.
- Ik meet de tijd nodig om het maximum te zoeken in een lijst van:
  - **100.000** elementen = **0,68 ms**
  - **1.000.000** elem. = **6,9 ms**
  - **10.000.000** elem. = **66 ms**
  - **100.000.000** elem. = **616 ms**
- Wat zeggen deze metingen nu eigenlijk ?
- Kan ik iets doen met de exacte waarden ?
- Zijn deze metingen overal en altijd hetzelfde ? Morgen, overmorgen, op jouw laptop ? Op een server ? ….

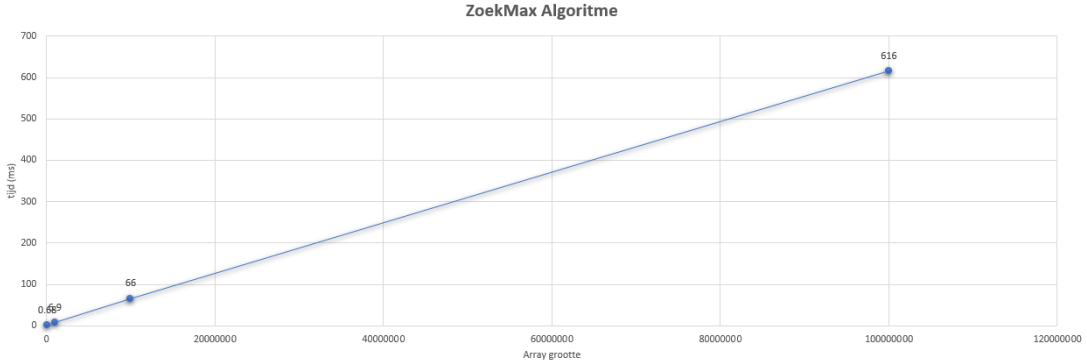
```
/// <summary>
/// Zoek het element met de hoogste waarde in de gegeven lijst
/// </summary>
/// <param name="lijst">een lijst met enkel positieve getallen</param>
/// <returns>de hoogste waarde</returns>
1 reference
public static int ZoekMax(int[] lijst)
{
    var max = 0;
    for (long f = 0; f < lijst.Length; f++)
    {
        if (lijst[f] > max)
            max = lijst[f];
    }
    return max;
}
```

Absolute metingen vergelijken ?

- De absolute waarden gaan vergelijken is niet eenvoudig. Deze zijn immers afhankelijk van verschillende factoren:
  - Hoe krachtig is mijn PC (hardware, CPU, geheugen,..) ?
  - Welk OS gebruik ik (windows, linux, iOS,..) ?
  - Met welke programmeertaal werk ik (c#, python, javascript,..) ?
  - Welke applicaties draaien op dat moment nog in de achtergrond (virus scanner, andere apps,..) ?
  - ….
- Maw. een zelfde algoritme kan binnen 2 jaar totaal andere resultaten geven als ik bv. een nieuwe laptop heb aangekocht.

ZoekMax algoritme

- Als we deze metingen uitzetten in een grafiek dan zien we echter wel een lineair verloop:



- Maw. voor elk bijkomend element is er telkens evenveel extra tijd nodig.

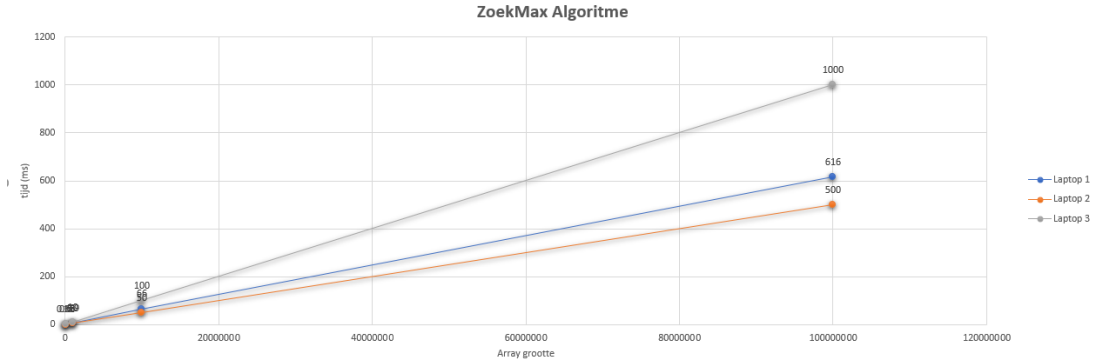
Lineair verloop voor het “ZoekMax” algo

- Als we code bekijken is het ook vrij logisch dat het verloop lineair is.
- Voor elk element worden immers telkens dezelfde instructies uitgevoerd.
  - **“For” lus + “If” statement.**

```
/// <summary>
/// Zoek het element met de hoogste waarde in de gegeven lijst
/// </summary>
/// <param name="lijst">een lijst met enkel positieve getallen</param>
/// <returns>de hoogste waarde</returns>
1 reference
public static int ZoekMax(int[] lijst)
{
    var max = 0;
    for (long f = 0; f < lijst.Length; f++)
    {
        if (lijst[f] > max)
            max = lijst[f];
    }
    return max;
}
```

Vergelijken van het verloop

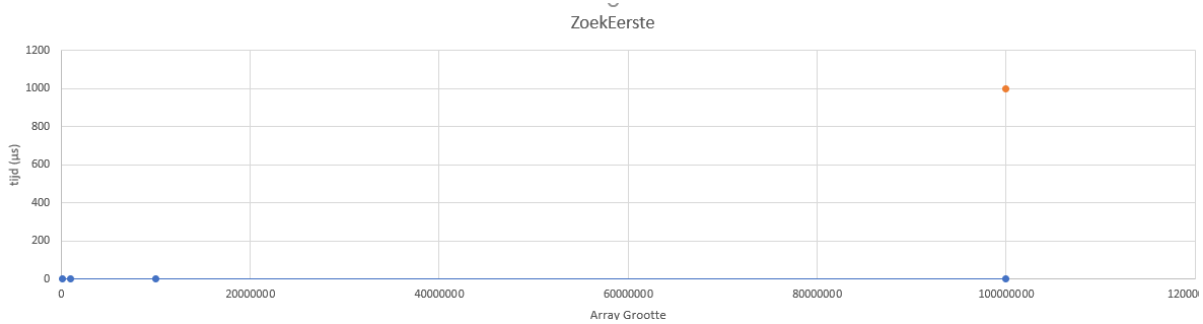
- Als we deze grafiek vergelijken met andere hardware,.. dan zullen de absolute waarden waarschijnlijk verschillen maar zal het verloop heel gelijkaardig & eveneens lineair zijn:



Time complexity, voorbeeld 2

- Hoe verwacht je dat het verloop is van het '**ZoekEerste**' algoritme ?
- En van het '**ZoekLaatste**' ?

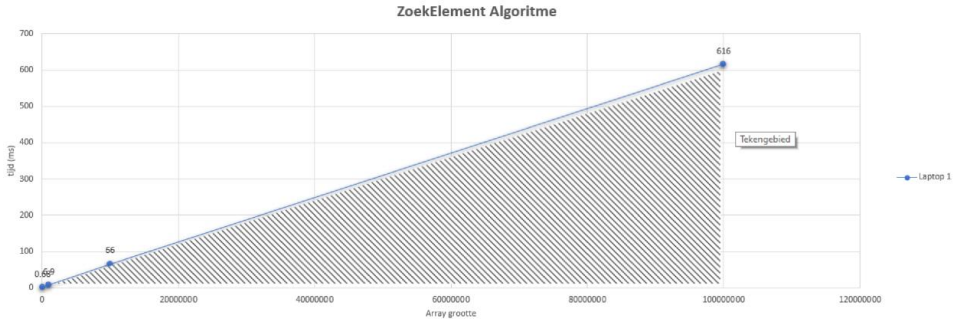
```
/// <summary>
/// Zoek het eerste element in de gegeven lijst van getallen.
/// </summary>
/// <param name="lijst">een lijst met minimaal 1 element</param>
/// <returns>de waarde van het eerste element</returns>
0 references
public static int ZoekEerste(int[] lijst)
{
    return lijst[0];
}

0 references
public static int ZoekLaatste(int[] lijst)
{
    return lijst[lijst.Length - 1];
}
```



Time complexity, voorbeeld 3

- Hoe verwacht je dat het verloop is van '**ZoekElement**' ?
- Gaan we hier steeds de volledige lijst doorlopen ?
- We kunnen dus stellen dat voor een bepaalde grootte van array:
  - De tijd **niet altijd dezelfde** zal zijn.
  - Maar ook afhankelijk is **van de plaats** in de array waar het betreffende element zich bevindt.

```
/// <summary>
/// Zoek in de gegeven lijst naar een element met de opgegeven waarde
/// </summary>
/// <param name="lijst">een lijst met minstens 1 element</param>
/// <param name="waarde">de op te zoeken waarde</param>
/// <returns>TRUE indien de waarde voorkomt in de lijst</returns>
0 references
public static bool ZoekElement(int[] lijst, int waarde)
{
    for (long f = 0; f < lijst.Length; f++)
    {
        if (lijst[f] == waarde)
            return true;
    }
    return false;
}
```



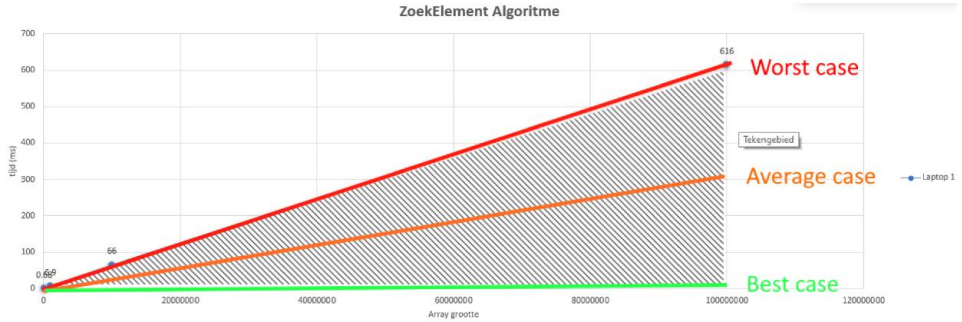
Hoe beoordelen van de complexity ?

- De absolute tijdmetingen zijn van meerdere factoren afhankelijk.
  - Enerzijds van de invoer (grootte array)
  - Anderzijds van hardware, OS, programmeertaal,…
- We gaan bijgevolg geen rekening houden met absolute waarden, maar wel **met het verloop in functie van de invoer**
  - Hiervoor bestaat **een specifieke notatie**
    - Big O => Worst case
    - Big Ω (Omega) => Best case
    - Big Θ (Theta) => Average case

Worst case, best case , average case

- Wij hebben vastgesteld dat bij het '**ZoekElement**' algoritme er voor **een zelfde input** toch **een verschillend verloop** mogelijk was.
  - Big O => Worst case
  - Big Ω (Omega) => Best case
  - Big Θ (Theta) => Average case
- Wij gaan **enkel Big O gebruiken**

```
/// <summary>
/// Zoek in de gegeven lijst naar een element met de opgegeven waarde
/// </summary>
/// <param name="lijst">een lijst met minstens 1 element</param>
/// <param name="waarde">de op te zoeken waarde</param>
/// <returns>TRUE indien de waarde voorkomt in de lijst</returns>
0 references
public static bool ZoekElement(int[] lijst, int waarde)
{
    for (long f = 0; f < lijst.Length; f++)
    {
        if (lijst[f] == waarde)
            return true;
    }
    return false;
}
```



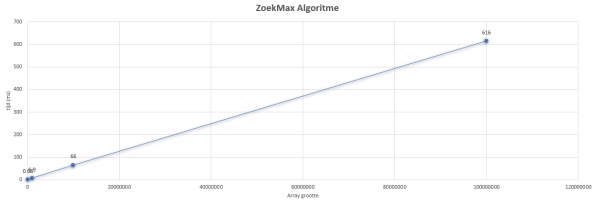
Big-O Notation

- Afhankelijk van het verloop van de benodigde tijd van een algoritme wordt een andere notatie gebruikt.
- Bijvoorbeeld: indien de tijd **lineair** verhoogt in functie van de input (n), dan duiden we dit aan met **O(n)** (wordt uitgesproken als : "**Big o of n**")

Big-O	Name
$O(1)$	Constant
$O(\log n)$	Logarithmic
$O(n)$	Linear
$O(n \log n)$	Log-linear
$O(n^2)$	Quadratic
$O(n^3)$	Cubic
$O(2^n)$	Exponential

Achtergrond bij de Big-O notation

- Voor een lineair verloop, maw. (vergelijking van) een rechte => **y = a.x + b**
  - Of in dit geval => **t = a.n + b**, waarbij:
  - t = tijd
  - n = grootte van de array
  - a en b zijn constant (afhankelijk van hardware, os,..)

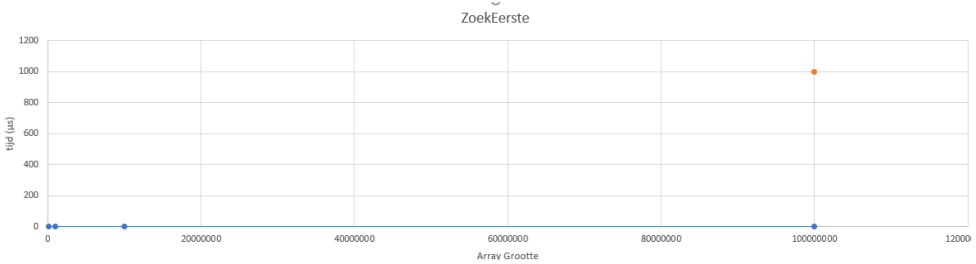


- Stel bijvoorbeeld dat a = 2 en b = 1
  - Voor **n = 100** => t = 2x100 + 1 = **201**
  - Voor **n = 1000000** => t = 2x1000000 + 1 = **2000001**
- b is verwaarloosbaar tov. a.n => we nemen **enkel de term met de hoogste orde** van n
- We laten a wegvallen want a had ook evengoed 1, 3, 4,… kunnen zijn => **we stellen de constanten = 1.**

  We behouden **t = n** oftewel **O(n)** om een lineair verloop aan te geven.

Achtergrond bij Big-o notation

- Voor een constant verloop is de vergelijking: y = a
  - Of in ons geval: t = a, waarbij
  - t = tijd
  - a = constante
- Stel dat a= 10
  - Voor n = 100 => t = 10
  - Voor n = 1000000 => t = 10



- Er is maar **1 term** en we **stellen de constante** wederom **= 1**
- We behouden t = 1 oftewel **O(1)**

Time Complexity overzicht

Algoritme	Verloop	Big-O notatie
ZoekMax	Lineair	**O(n)**
ZoekEerste	Constante	**O(1)**
ZoekLaatste	Constante	**O(1)**
ZoekElement	Lineair	**O(n)**

```
/// <summary>
/// Zoek het element met de hoogste waarde in de gegeven lijst
/// </summary>
/// <param name="lijst">een lijst met enkel positieve getallen</param>
/// <returns>de hoogste waarde</returns>
1 reference
public static int ZoekMax(int[] lijst)
{
    var max = 0;
    for (long f = 0; f < lijst.Length; f++)
    {
        if (lijst[f] > max)
            max = lijst[f];
    }
    return max;
}
```

```
/// <summary>
/// Zoek het eerste element in de gegeven lijst van getallen.
/// </summary>
/// <param name="lijst">een lijst met minimaal 1 element</param>
/// <returns>de waarde van het eerste element</returns>
0 references
public static int ZoekEerste(int[] lijst)
{
    return lijst[0];
}

0 references
public static int ZoekLaatste(int[] lijst)
{
    return lijst[lijst.Length - 1];
}
```

```
/// <summary>
/// Zoek in de gegeven lijst naar een element met de opgegeven waarde
/// </summary>
/// <param name="lijst">een lijst met minstens 1 element</param>
/// <param name="waarde">de op te zoeken waarde</param>
/// <returns>TRUE indien de waarde voorkomt in de lijst</returns>
0 references
public static bool ZoekElement(int[] lijst, int waarde)
{
    for (long f = 0; f < lijst.Length; f++)
    {
        if (lijst[f] == waarde)
            return true;
    }
    return false;
}
```

Space Complexity

- Gebruiken we ook de **Big-O** notatie
- We kijken nu echter naar het **geheugenverbruik** van het algoritme.
- Maw. **welke variabelen** zijn er nodig binnen het algoritme
- We kijken wederom **niet** naar de **absolute waarden** (aantal bytes)
- Wel: Wat is het **verband** tussen de **input en deze variabelen ?**
- Maw. heeft de input een invloed op de grootte van deze variabelen ?

```
/// <summary>
/// Zoek het element met de hoogste waarde in de gegeven lijst
/// </summary>
/// <param name="lijst">een lijst met enkel positieve getallen</param>
/// <returns>de hoogste waarde</returns>
1 reference
public static int ZoekMax(int[] lijst)
{
    var max = 0;
    for (long f = 0; f < lijst.Length; f++)
    {
        if (lijst[f] > max)
            max = lijst[f];
    }
    return max;
}
```

```
/// <summary>
/// Zoek het eerste element in de gegeven lijst van getallen.
/// </summary>
/// <param name="lijst">een lijst met minimaal 1 element</param>
/// <returns>de waarde van het eerste element</returns>
0 references
public static int ZoekEerste(int[] lijst)
{
    return lijst[0];
}

0 references
public static int ZoekLaatste(int[] lijst)
{
    return lijst[lijst.Length - 1];
}
```

```
/// <summary>
/// Zoek in de gegeven lijst naar een element met de opgegeven waarde
/// </summary>
/// <param name="lijst">een lijst met minstens 1 element</param>
/// <param name="waarde">de op te zoeken waarde</param>
/// <returns>TRUE indien de waarde voorkomt in de lijst</returns>
0 references
public static bool ZoekElement(int[] lijst, int waarde)
{
    for (long f = 0; f < lijst.Length; f++)
    {
        if (lijst[f] == waarde)
            return true;
    }
    return false;
}
```

Time & Space Complexity overzicht

Algoritme	Verloop tijd	Time Complexity	Verloop geheugen	Space Complexity
ZoekMax	Lineair	**O(n)**	Constante	**O(1)**
ZoekEerste	Constante	**O(1)**	Constante	**O(1)**
ZoekLaatste	Constante	**O(1)**	Constante	**O(1)**
ZoekElement	Lineair	**O(n)**	Constante	**O(1)**

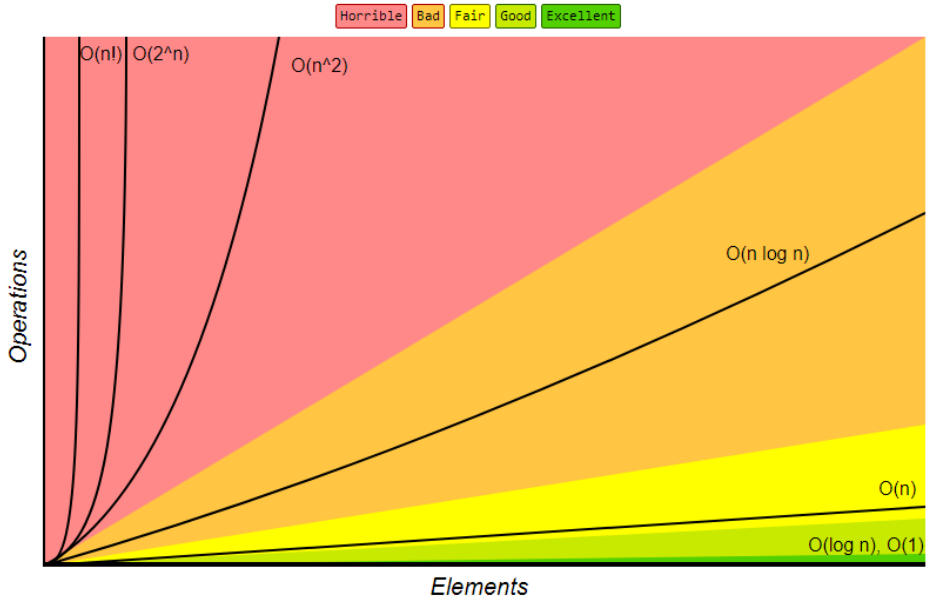
```
/// <summary>
/// Zoek het element met de hoogste waarde in de gegeven lijst
/// </summary>
/// <param name="lijst">een lijst met enkel positieve getallen</param>
/// <returns>de hoogste waarde</returns>
1 reference
public static int ZoekMax(int[] lijst)
{
    var max = 0;
    for (long f = 0; f < lijst.Length; f++)
    {
        if (lijst[f] > max)
            max = lijst[f];
    }
    return max;
}
```

```
/// <summary>
/// Zoek het eerste element in de gegeven lijst van getallen.
/// </summary>
/// <param name="lijst">een lijst met minimaal 1 element</param>
/// <returns>de waarde van het eerste element</returns>
0 references
public static int ZoekEerste(int[] lijst)
{
    return lijst[0];
}

0 references
public static int ZoekLaatste(int[] lijst)
{
    return lijst[lijst.Length - 1];
}
```

```
/// <summary>
/// Zoek in de gegeven lijst naar een element met de opgegeven waarde
/// </summary>
/// <param name="lijst">een lijst met minstens 1 element</param>
/// <param name="waarde">de op te zoeken waarde</param>
/// <returns>TRUE indien de waarde voorkomt in de lijst</returns>
0 references
public static bool ZoekElement(int[] lijst, int waarde)
{
    for (long f = 0; f < lijst.Length; f++)
    {
        if (lijst[f] == waarde)
            return true;
    }
    return false;
}
```

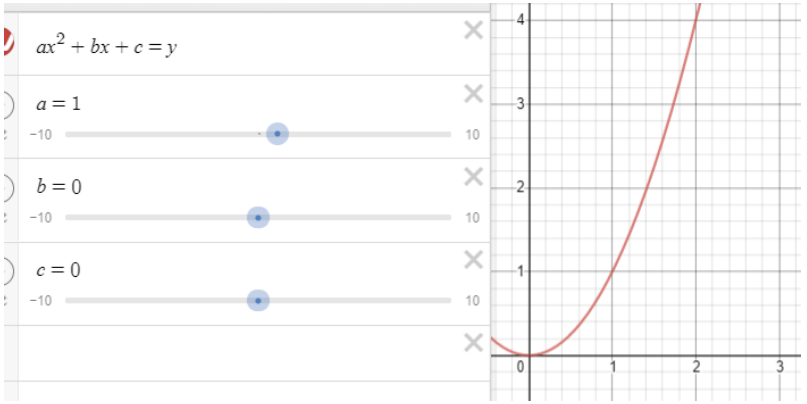
Big-O overzicht

Big-O	Name
O(1)	Constant
O(log n)	Logarithmic
O(n)	Linear
O(n log n)	Log-linear
O(n^2)	Quadratic
O(n^3)	Cubic
O(2^n)	Exponential



$O(n^2)$?

- Kwadratisch verloop
- n verhogen geeft een tijds en/of geheugen verhoging van n^2
- Bv. een 2-dim array
  - n = 3 => 9 elementen
  - n = 5 => 25 elementen
- Bv. een for lus in een for lus
  - n = 3 => 9 elem. opvullen
  - n = 5 => 25 elem. opvullen
- Hier zien we dus een
  - Time complexity = $O(n^2)$
  - Space complexity = $O(n^2)$

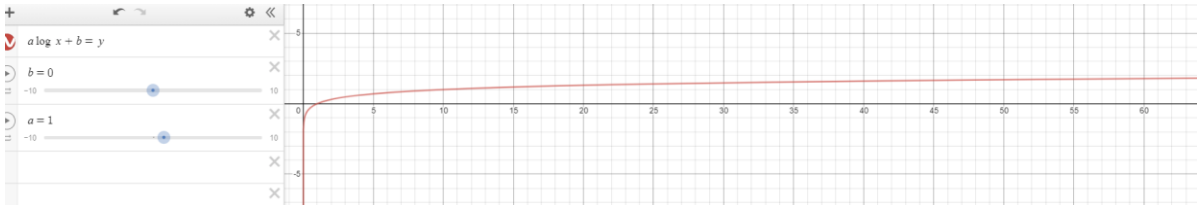



```
/// <summary>
/// Maak een 2 dim. array aan en vul deze met de gegeven waarde
/// </summary>
/// <param name="grootte">grootte van beide dimensies van de array</param>
/// <param name="inhoud">waarde waarmee de array dient te worden opgevuld</param>
/// <returns>de aangemaakte array</returns>
0 references
public static int[,] GenereerEnVul2DimensioneleArray(int grootte, int inhoud)
{
    int[,] result = new int[grootte, grootte];
    for(int f = 0; f < result.GetLength(0);f++)
    {
        for(int g = 0; g < result.GetLength(1);g++)
        {
            result[f, g] = inhoud;
        }
    }
    return result;
}
```

O (log n) ?

- Logaritmisch verloop
- Zoekalgoritmes die het zoekbereik steeds halveren.
- Vb. zoeken in telefoongids:
  - Start in het midden
  - Groter of kleiner ?
  - 1^e of 2^e helft nemen
  - Terug in het midden starten
  - Enz…
- Na “enkele” stappen kom je tot het gezochte item.

- For a phone book of 3 names it takes 2 comparisons (at most).
- For 7 it takes at most 3.
- For 15 it takes 4.
- …
- For 1,000,000 it takes 20.



AP HOGESCHOOL
ANTWERPEN